Data-Limited Methods Toolkit (DLMtool)

User Guide

2017-05-01

Contents

Chapter 1

Introduction

As many as 90% of the world's fish populations have insufficient data to conduct a conventional stock assessment (Costello et al. 2012). Although a wide range of data-limited management procedures (MPs; stock assessments, harvest control rules) have been described in the primary and gray literature (Newman et al. 2015), they have not been readily available or easily tested to determine their efficacy for specific fisheries.

For many fishery managers and stakeholders, the path forward has been unclear and laden with myriad questions, such as: How do these MPs perform comparatively? What are the performance trade-offs? What MPs are appropriate for a given fishery? What is the value of collecting additional data? What is an appropriate stop-gap management approach as more data are collected?

1.1 Data-Limited Methods Toolkit

The Data-Limited Methods Toolkit (DLMtool), a collaboration between the University of British Columbia's (UBC) Institute for Oceans and Fisheries and the Natural Resources Defense Council (NRDC), is aimed at addressing these questions by offering a powerful, transparent approach to comparing, selecting, and applying various data-limited management methods. DLMtool uses management strategy evaluation (MSE) and parallel computing to make powerful diagnostics accessible.

A streamlined command structure and operating model builder allow for rapid simulation testing and graphing of results. The package is relatively easy to use for those inexperienced in R, however, complete access and control is available to more experienced users.

While DLMtool includes over 89 management procedures it is also designed to be extensible in order to encourage the development and testing of new methods. The package is structured such that the same management methods that are tested by the MSE can be applied to provide management recommendations from real data.

Easy incorporation of real data is a central advantage of the software. A set of related functions automatically detect what management procedures can be applied with currently available data, and what additional data are needed to use currently unavailable methods.

The Toolkit has been developed in collaboration with fisheries scientists around the globe. New features and functions have been added to the software package to meet the needs or the particular fisheries and management contexts where it has been applied. To date, the Toolkit has been used for management or academic research in over 25 fisheries, including by the National Marine Fisheries Service in the U.S. Mid-Atlantic and Caribbean regions, and by the California Department of Fish & Wildlife.

1.2 Management Strategy Evaluation

At the core of the Data-Limited Methods Toolkit is an integrated management strategy evaluation (MSE) function. Management strategy evaluation is a computer simulation approach for testing prospective management options over a wide range of possible realities for the fishery and the population. Ideally, management options can be identified that are robust and perform well over all credible scenarios for the fishery.

It is extremely difficult, perhaps impossible, to conduct large-scale experiments to evaluate directly the trade-offs associated with fisheries management. Even among well-studied fisheries, considerable uncertainty often exists regarding stock status and the dynamics of the fishery, and it can be difficult to attribute particular outcomes to distinct management actions. The mathematical description of fish population dynamics and the interaction with different exploitation patterns, first developed by Beverton and Holt (1957), together with the advent of powerful and affordable computers, has allowed the development of the MSE approach (Butterworth, 2007; Punt et al. 2014).

Management strategy evaluation was originally developed by the International Whaling Commission as a tool to evaluate the various trade-offs involved the management of marine mammals, and to guide the decision-making process for selecting an appropriate management strategy. Since its development in the mid-1970s, MSE has become widely used in fisheries science and is routinely applied to evaluate the trade-offs in alternative management strategies of many of the world's fisheries.

An MSE is usually comprised of three key components:

1. an ***operating model*** that is used to simulate the stock and fleet dynamics,
2. an assessment method and harvest control rule model (interchangeably referred to as ***management procedures***, or ***management strategies***) that use the simulated fishery data from the operating model to estimate the status of the (simulated) stock and provide management recommendations (e.g., a total allowable catch (TAC) or effort control), and
3. an ***observation model*** that is used to generate the simulated observed data that would typically be used in management (i.e., with realistic imprecision and bias).

The management recommendations by each management procedure are then fed-back into the operating model and projected forward one-time step. The process of simulating the population dynamics of the fishery along with the management process that feeds back and impacts the simulated fish population is known as ***closed-loop simulation***.

A benefit of closed-loop simulation is that it allows the direct comparison and evaluation of alternative management strategies against perfect knowledge of the simulated system; something that is impossible in the real world (Walters and Martell, 2004). With the aid of computer simulation, it is possible to run many hundreds of simulation runs for each management procedure being evaluated - each representing a different possible simulated future of what could happen to the fishery under various management strategies - and to take into account the uncertainty in knowledge of the stock and fishery (i.e., errors in observation), as well as the uncertainty in future environmental and ecological conditions that are likely to affect the stock dynamics.

Through these simulations, MSE reveals the relative impacts of specified management approaches to their fishery decades into the future and enables managers to choose the approach that best achieves their management objectives, as articulated through a set of well-defined performance metrics.

1.3 How does Management Strategy Evaluation Differ from Stock Assessment?

Stock assessments are intended to provide one-off management advice, such as a catch limit (e.g. 20,000 tonnes), based on historical data. However, a stock assessment on its own provides no knowledge of the expected performance of the assessment, harvest control rule, or management system in general.

In an assessment setting there is no way to know whether a simpler assessment using other data might provide more robust performance (e.g. less overfishing, more yield) over a time horizon that managers are considering (e.g. the next 30 years). Management strategy evaluation tests a range of management approaches (of which an assessment linked to a harvest control rule is one such approach) and offers a scientific basis for selecting a management approach. MSE does not provide a catch-limit in tonnes, it identifies a modus operandi that will provide the desired management performance (it is analogous to selecting a suitable airplane via flight simulation testing rather than actually flying a plane to a specific destination).

The advantage of MSE over stock assessment is that it is possible to consider a much wider range of uncertainty in stock dynamics, fleet dynamics, and data collection, which often better represents the state of knowledge (particularly for data-limited stocks). No matter how much uncertainty is factored into the MSE, a single management approach may be selected that can provide management advice.

MSE was specifically introduced in controversial fishery settings where it was not possible to decide the 'best' representation of the state of nature. In the end, MSE was used to circumvent this problem by including all possible states of nature, often revealing that the disputes were in fact inconsequential all along.

1.4 Overview of the User Guide

This User Guide is designed for new users to become familiar with the features of the DLMtool, and become comfortable with using the Toolkit to run a management strategy evaluation, select an appropriate management procedure for a particular fishery, and apply the management procedure to current fishery data to obtain specific management advice.

The DLMtool is developed with R software, and users of the Toolkit will need to have the R software installed on their machine. Familiarity with the R programming language is beneficial, but an in-depth understanding of R is not required to use the DLMtool.

1.5 Acknowledgements

Thanks to the many people who have alerted us to issues or bugs, provided suggestions for improvements, or asked the tricky, but important, questions that have helped us continue to develop the DLMtool.

This User Guide has been developed with the bookdown package.

Developers:

- Thomas Carruthers, University of British Columbia (UBC) Institute for the Oceans and Fisheries
- Adrian Hordyk, University of British Columbia (UBC) Institute for the Oceans and Fisheries

Collaborators:

- Doug Butterworth, University of Cape Town
- Campbell Davies, Commonwealth Scientific and Industrial Research Organisation (CSIRO)
- Helena Geromont, University of Cape Town
- William Harford, National Oceanic and Atmospheric Administration (NOAA)
- Richard Hillary, Commonwealth Scientific and Industrial Research Organisation (CSIRO)
- Quang Huynh, Virginia Institute of Marine Science (VIMS)
- Laurie Kell, International Commission for the Conservation of Atlantic Tuna (ICCAT)
- Toshihide Kitakado, University of Tokyo
- Skyler Sagarese, University of Miami Rosenstiel School of Marine and Atmospheric Science (RSMAS)
- Liz Brooks, National Oceanic and Atmospheric Administration (NOAA)
- Robyn Forrest, Canadian Department of Fisheries and Oceans
- Chris Grandin, Canadian Department of Fisheries and Oceans
- California Department of Fish and Wildlife

Funders:

- David & Lucille Packard Foundation
- Gordon & Betty Moore Foundation
- Kingfisher Foundation
- Natural Resources Defense Council
- Resources Legacy Fund

1.6 Bug Reports

The package is subject to ongoing development and testing. If you find a bug or a problem please contact us so that it can be fixed!

Fundamentally, the package is stochastic so if you run into problems with the code, please report it (along with a random seed) and, in the meantime, simply try running it again: the problem may be attributable to a rare combination of sampled parameters.

1.7 Version Notes

The current version of the DLMtool package, available for download from CRAN, is 4.1.

1.7.1 New Additions to Version 3.2.2

A number of additional plotting functions, and a few new MPs have been added in this version. Also a few minor changes to improve performance and reliability of the model.

For improved stability, especially with large files, the `runMSErobust` function has been changed so that it now uses the `saveRDS` function to write the MSE objects to disk. MSE objects saved with this version of the function need to be loaded with `readRDS`.

The `plotFun` function can be used to print out all available plotting functions for objects of class `MSE` or `DLM_data`

MSE Object

Pplot and Kplot functions have been modified for extra control of various features of the plots.

Two functions, `barplot.MSE` and `boxplot.MSE`, have been added to plot the MSE object. Call them using the generic `barplot` or `boxplot` functions, and see `?barplot.MSE` and `?boxplot.MSE` for information on the arguments for the function.

New functions: `Jplot`, `Splot`, `Cplot`, and `VOIplot` have been added.

Data Object

`boxplot.Data` has been added and can be used to plot boxplots of the TACs recommendations from different methods. Call with `boxplot(DLM_data)`

New MPs

Three new input control methods, developed by Helena Geremont, have been added to the package: `EtargetLopt`, `L95target`, and `minlenLopt1`.

1.7.2 New Additions to Version 3.2.1

A new slot (Effort) has been added to the MSE object. This stores the fishing effort for each year, simulation, and MP in the projection years. The addition of the new slot may cause a warning message to be thrown up if an MSE object from a previous version of the DLMtool is loaded.

You can update the old MSE object by adding an empty `Effort` slot: `MSEobj <- updateMSE(MSEobj)`.

There were some issues with a couple of the input control MPs, which have now been addressed (thanks, Helena, for identifying these). There was also a problem with effort and selectivity being calculated for the input controls, which has also been fixed.

1.7.3 New Additions to Version 3.2

A number of small but important bugs have been fixed, with special thanks to Liz Brooks, Helena Geromont, and Bill Harford for alerting us to some of these issues.

Quang Huynh has recoded the mean length methods in C++, and they now run much faster, so should pass the time-limit constraint.

A new function (`runMSErobust`) has been added which is a wrapper for the `runMSE` function. In time, this may replace `runMSE` as the primary function to use when running a MSE. `runMSErobust` splits large simulations into a series of smaller packets and stitches them together to return an MSE object. This has the benefit of increasing speed and efficiency, particularly for runs with large number of simulations. The function also checks for errors and re-starts the MSE if the model crashes.

A set of functions `OM_xl` and `Fease_xl` have been added. These are used to read in operating model and feasibility parameters from an Excel spreadsheet rather than a CSV file. These are essentially wrappers for the `new` function, but allow you to store all operating model tables in a single spreadsheet rather than a number of CSV files. This is mainly useful if you are working on multiple species/stocks.

The size limit feature has been updated to include an upper slot limit. See `slotlim` for an example MP. The slot limit is specified as the last element in the input control vector. Similar to the lower size limit, all individuals above the slot limit experience no fishing mortality.

A number of new MPs have been added. There are now 63 output and 22 input control MPs in the DLMtool.

A new function `makePerf` has been added. This function takes an OM object, and returns the same OM object with no process or observation error. This is useful for testing the performance of methods under perfect conditions, to see if they work as expected. And for debugging!

Two new plotting functions have been added: `wormplot` which creates worm plots of the likelihood of meeting biomass targets in future years, and `VOIplot` which is another value of information plot, similar to the `VOI` function, and shows how observation and operating model parameter values affect trends in long-term yield and biomass.

Coming soon: bag limit MPs for recreational fisheries

1.7.4 Notes from Version 3.1

In order to simulate fisheries that have experienced important shifts in historical length selectivity, this can now be user specified using a graphical user interface (the 'ChooseSelect' function) or by manually editing a series of new slots in the Fleet object (`SelYears`, `AbsSelYears`, `L5Upper`, `L5Lower`, `LFSUpper`, `LFSLower`, `VmaxUpper`, `VmaxLower`).

Persistent shifts in stock productivity are a particular concern for fishery management. These can now be generated in the toolkit using a new function `SetRecruitCycle` that generates cyclical pattern in recruitment strength.

Length-based spawning potential ratio (SPR) MPs have been added. Currently these methods are slow and often don't pass the time constraint.

Two features have been added to allow MPs to return additional information for future reference. (1) The DLM_data object that MPs operate on now has a miscellaneous slot `Misc`. (2) MPs can now return a list. The first position is the management recommendation (e.g. TAC) the second is information that is stored in the `Misc` slot that can be used by the MP in the next iteration. This can be useful for storing information that is time-consuming to calculate each year.

A new generic trade-off performance plot `TradePlot` has also been added.

1.7.5 A Note on Version 2.11

Operating model effort is now simulated by a time-series of year vertices and relative magnitude of effort at each vertex. It follows that the slot `Fleet@Fgrad`, and has been replaced by three slots with vectors of equal length: `Fleet@EffYear`, `Fleet@EffUpper` and `Fleet@EffLower`. These effort trajectories can now be specified by a new graphical interface (function `ChooseEffort()`) which uses points to determine the three slots described above.

Operating model fleet selectivity has been robustified to prevent users from specifying length at first capture (`Fleet@L5`) and length at full selection (`Fleet@LFS`) that are unrealistically high. According to our view of reality these now have upper limits of L50 and maximum length, respectively.

A function `DOM()` has been added that evaluates how often one MP outperforms another across simulations. It is possible that an MP could have higher average performance but perform worse on higher fraction of simulations. The `DOM()` function provides a diagnostic to analyze this.

An additional function `Sub()` has been added which allows users to subset an MSE object according to either (or both) a vector of MPs and simulations. This means you no longer have to rerun everything to provide results for a smaller number of MPs or particular simulations.

1.7.6 A Note on Bug Fixes in 2.1.1 and 2.1.2

A bug was found in which length at first capture was being sampled from a uniform distribution `U(LB,UB*2)` rather than `U(LB,UB)`. When depletion could not be simulated by even very high fishery catchabilities an error could occur after more than 10 attempts to find a suitable value of depletion. Length composition simulation in 2.1.1 was not correctly implemented leading to minor biases.

1.7.7 A Note on Version 2.1

In response to popular demand, simulation and data are entirely length-based now. It follows that many objects that worked with 2.0 will no longer be compatible. In most cases it is very quick to make files/objects compatible with version 2.1, but nonetheless we apologize if this is frustrating!

The DLMtool package is stochastic, so if you run into problems with the code, please report them (along with a random seed). In the meantime, simply try running it again; the problem may be attributable to a rare combination of sampled parameters.

Be warned that if you abort a parallel process (e.g. `runMSE()`) half-way through, you are in the lap of the gods! It will often be necessary to restart the cluster `sfInit()` or even restart R.

The package is not designed for very short lived stocks (that live for less than 5 years) due to the problems with approximating fine-scale temporal dynamics with an annual model. Technically, you could just divide all your parameters by a sub-year resolution, but the TAC would be set by sub-year and the data would also be available at this fine-scale, which is highly unlikely in a data-limited setting.

1.7.8 New to Version 2.1

(1) The DLMtool has moved to a length-based simulator (maturity, fisheries selectivity by length)

(2) The spatial targeting function has been removed for the moment as its implementation was flawed , so could not distribute fishing correctly with respect to both density and the amount of the resource among the two areas.

(3) `Tplot2` adds a different set of trade-offs including long-term and short-term probability of achieving 50% of FMSY yield and average annual variability in yields.

(4) Version 2.0 did not include observation error in estimates of current stock abundance and depletion (only biases were simulated). Many thanks to Helena Geromont for spotting this. This has now been corrected.

(5) `DLM_data` objects now have a slot `LHYear` which is a numeric value corresponding with the last historical year. This is needed for some MPs that want to run off only the past data rather than the updated (projected, closed-loop simulation) data.

(6) Post-MSE, you can now run a Convergence function `CheckConverg()` to see if performance metrics are stable.

(7) The package now contains `CSRA`, a tool for calculating very rough estimates of current depletion and fishing mortality rate from mean catch data.

(8) `getAFC` now can be used for converting length estimates to age estimates through a stochastic growth model.

(9) The value of information function (`VOI`) contained bugs in version 2.0. This now has been fixed.

(10) Users can now send their own parameter values to the `runMSE` function allowing outputs from stock assessments or correlated parameters (e.g. K and age at maturity) values.

(11) After deliberation, Pope's approximation has been used to account for intra-year mortality (i.e., TACs are taken from biomass at the start of the year subject to half of natural mortality rate). This is probably a reasonable approximation in a data-limited setting: alternative structural assumptions for M are eclipsed by uncertainty in M itself and other operating model parameters such as selectivity and bias in observation of data such as annual catches.

(12) The simulation of length composition data was bugged in version 2.0. The variability in length at age was taken from the observation model. Using the perfect information observation model therefore led to no variability in length at age and hence very odd length composition data. This has been solved; now a fixed 10% CV in length-at-age is assumed (normally distributed).

(13) A bug with Delay-Difference MPs has been fixed (`DD` and `DD4010`) in which stochastic TACs were sampled when reps =1. This should just be the mean estimate. The result is that DD is much less variable between years but comes with less contrast in the data. In addition to the much less variable catch recommendations, long-term mean performance of the MP is reduced while medium-term performance has been improved.

(14) In the move to length-based inputs it is possible to prescribe wild biases for maximum length and length at maturity. In this version these sampled biases are not correlated so it is possible to create simulated data sets where maximum length is lower than length-at-95% maturity and length-at-50% maturity. We put a hard ceiling on this such that length at 95 percent maturity must be below 90 percent of maximum length and length-at-50% maturity must be below 90% of length-at-95% maturity. This isn't great and this will be improved for v2.11.

(15) The package now works without initiating a cluster `sfInit()`.

(16) A simple modification to `DCAC` has been added `EDCAC` (Harford and Carruthers, 2015) that better accounts for absolute stock depletion.

(17) Three new slots are available to run MPs on that related to mean length of catches (`ML`), modal length of captures (`Lc`), and the mean length of catches of fish over `Lc` (`Lbar`)

1.7.9 New to Version 2.0

(1) Much has changed in the DLMtool terminology to make it more generally applicable. For example, OFL (overfishing limits, FMSY x current biomass), now belongs to a larger class of TACs (Total Allowable Catches).

(2) There are now just two classes of DLM MPs, DLM_output (MPs linked to output controls e.g. TACs) and DLM_input (MPs linked to input controls such as time-area closures, age selectivity and effort). The new DLM_input function classes have four components, fractional reallocation of spatial effort, fraction of effort in final historical year prescribed in the current year, spatial limits on fishing mortality and a user-defined age-selectivity curve. For example, given an hypothetical stock with 8 age classes a DLM_input method might return a vector c(0.5, 0.8, 0,1, 0,0,0,0,1,1,1,1). This is interpreted as a 50% reallocation (Allocation = 0.5) of spatial effort, with a total effort that is 80% of historical levels (Effort = 0.8) with a closure in area 1 and full fishing in area 2 (Spatial = c(0,1)) and knife-edge selectivity at age class 5 (Selectivity = c(0,0,0,0,1,1,1,1)) [note that Selectivity has changed in newer versions of the package]. To demonstrate this new feature there are four new input controls, current effort (curE), 75% of current effort (curE75), age selectivity that matches the maturity ogive (matagelim) and a marine reserve in area 1 (area1MR) [note that matagelim has changed to matlenlim in recent versions].

(3) A 'dumb' MP has been added: Mean Catch Depletion (MCD) that simply calculates a TAC based on mean catches and depletion. This is to demonstrate the (theoretically) very high information content of a reliable estimate of current stock depletion.

(4) A better length composition simulator has been added. Note that this still renews the normal length structure between ages and does not properly simulate the higher mortality rate of larger, faster growing fish (a growth type group simulator is on its way).

(5) Help documentation has been much improved including complete guides for `Fleet`, `Stock`, `Observation` and `MSE` objects. Eg `class?MSE`.

(6) Minor bugs have been found with the help of Helena Geromont including a problem with update intervals of 1 and low simulated steepness values.

(7) Reliability is much improved following a full combinatorial test of all Fleet, Stock, Observation objects against all MPs.

(8) A dedicated Value of information function is now available for MSE objects: `VOI(MSEobject)` which is smarter than the former version which was included in plot(MSE object class).

(9) Plotting functions have been improved, particularly `Tplot`, `Kplot`, `Pplot` and `plot(DLM_data object class)`.

(10) `SPmod` has been robustified to stop strongly negative surplus production estimates from leading to erratic behavior.

(11) The butterfish stock type now has less variable recruitment and slightly lower natural mortality rate as previous values were rather extreme and led to data generation errors (with natural mortality rate as high as 0.9, butterfish is right at the limit of what can be simulated reasonably with an annual age-structured operating model).

Chapter 2

First Time Working With R?

This section is designed for first-time users of the DLMtool, or users who may not have a lot of experience with R.

You should be able to skip this section if you are familiar with R and RStudio, installing new R packages, and entering R commands into the R console.

To get started with the DLMtool you will need at least two things:

1. A current version of the R software installed on your machine.
2. The latest version of the DLMtool package.

2.1 R and RStudio

2.1.1 The R Software

The R software can be freely downloaded from the CRAN website and is available for all operating systems. Updated versions of R are released frequently, and it is recommended that you have the latest version installed.

If you are using Windows OS, you can uses the `installr` package and the `updateR()` function to update and install the latest version. Alternatively, head to the CRAN website to download the latest version of R.

2.1.2 RStudio

RStudio is a freely available integrated development environment (IDE) for R. It is not essential that you use RStudio, but it can make things a lot easier, especially if you are new to R. This User Guide assumes that you are using RStudio to operate the DLMtool.

It is important to be aware that RStudio and R are two different pieces of software that must be installed separately. We recommend installing the R software before downloading and installing RStudio.

2.2 Installing the DLMtool Package

If this is the first time you are using DLMtool, you will need to install the DLMtool package from CRAN.

2.2.1 Installing DLMtool Using R Console

This can be done by running the command:

```
install.packages("DLMtool")
```

A prompt may appear asking you to select a CRAN mirror. It is best to pick the mirror that is the closest geographical distance.

2.2.2 Installing DLMtool Using RStudio

An alternative method to install the DLMtool package is to click the *Packages* tab in the lower right panel in RStudio, and click *Install*. Check that *Repository (CRAN, CRANextra)* is selected in the *Install from:* drop-down menu, type **DLMtool** into the *packages* dialog box, and click *Install*.

The DLMtool package relies on a number of other R packages, which the installation process will automatically install. The number of packages that are installed, and the time it takes, will depend on what packages you already have installed on your system (and your download speed).

2.2.3 Updating the DLMtool Package

You will only need to install the DLMtool package once. However, the DLMtool package is updated from time to time, and you will need to re-install from CRAN for each new version.

This can be done by using the `update.packages` command:

```
update.packages("DLMtool")
```

2.2.4 Loading the DLMtool Package

Once installed, the DLMtool package can be loaded into R by typing in the command line:

```
library(DLMtool)
```

or locating the *DLMtool* package in the list of packages in RStudio and checking the box.

2.3 Assumed Knowledge

This User Guide assumes that you are using RStudio with an up-to-date version of R and the latest version of the DLMtool installed.

You can check your version of R by typing `version` into the R console:

```
version
```

```
##                  _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          3
## minor          3.3
## year           2017
## month          03
## day            06
## svn rev        72310
## language       R
## version.string R version 3.3.3 (2017-03-06)
## nickname       Another Canoe
```

You can also find the version of DLMtool (or any other package) by typing:

```
packageVersion('DLMtool')
```

```
## [1] '4.1'
```

The DLMtool package has been designed so that it is accessible for all users and does not assume a high level of knowledge of R. The functions and User Guide have been constructed in such a way that a user with little experience with R should be able to run the MSE and apply the methods to their data.

No programming experience is required to use the package. However, users of the DLMtool should have some familiarity with R, and be comfortable with using the command line. The User Guide attempts to explain the use of the DLMtool in easy to follow steps, but familiarity with the most common R functions is assumed.

The package is fully extensible, and more experienced R users are able to design their own management procedures, develop new plotting functions, and other customizations.

2.4 A Brief Note on S4 Methods

The core functions of DLMtool are *S4 Classes*. Many R users may not have worked with S4 methods before.

R has three different object oriented (OO) systems, the most common of which is known as **S3**. S3 is known as a generic-function OO, and is a casual system with no formal definition of classes. **S4** works similar to S3, but is more formal and uses classes with a more rigid definition.

It is not essential to understand the difference between S3 and S4, or why one is preferred over the other, to use the DLMtool. The most important thing that you need to know how to access the information in S4 classes.

If you have work with R in the past, you are probably familiar with using the **$** symbol to access elements in a data frame or list. S4 classes contain a named list of **slots** which are analogous to a standard R list. However, the slots in a S4 class differ in two important ways:

1. The type of content in each slot (e.g., character, numeric, matrix) is determined in the class definition, and cannot be changed. In other words, you are not able to put content of class `character` into a slot that is expecting information of class `numeric`. This is what is meant by the S4 system being more strict than S3.

2. The slots are accessed with the @ symbol. This is essentially the same as the **$** symbol in S3 classes. You will see examples of this throughout the User Guide.

The main thing to note here is that when you see the `@` symbol being used, it refers to some particular information (a *slot*) being accessed from a larger collection of data (the *object*).

For further information on the S3 and S4 systems see Advanced R.

2.5 Getting Help

2.5.1 Additional Help on the DLMtool

This User Guide aims to explain the workings of the DLMtool, and address the most common questions and issues associated with the package.

Additional help material for the DLMtool package and functions can be obtained in the usual way:

```
help(DLMtool)
```

Documentation for each function can be obtained by typing a ? symbol followed by the function name. For example:

```
?runMSE
```

Information on the DLMtool classes can be found by first typing `class` followed by the ? symbol and the class name. For example:

```
class?Data
```

2.5.2 Questions on R-related Problems

Although the User Guide attempts to address the most common issues, undoubtedly there will be times where you have problems with your R code. R has a somewhat annoying habit of returning cryptic error messages, that are sometimes indecipherable, especially to those who are new to the software.

Most coding problems with the R language are the result of a missing parenthesis, an extra or missing comma or quotation mark, or some other minor typo that stops your code from running.

There are a number of resources available on the Internet that are devoted to dealing with questions and problems with R programming. StackOverflow is great place to start searching for answers to your R-related problems.

There is a high chance that in the past someone has posted the exact question that you are dealing with, and one or several kind souls have provided helpful solutions. If not, you can post your own question. But be aware, the StackOverflow community is made up entirely of people who volunteer their time to help others, and they sometimes have little patience for question that don't demonstrate a proper search for already posted answers to the problem.

Chapter 3

Prerequisites to Using DLMtool

At the start of every session there are a few things to do: 1. Load the DLMtool library. 2. Make the data available. 3. Set up parallel computing.

The following steps must be run every time the DLMtool is used in a new R session.

3.1 Load the Library

At the beginning of every session you must load the DLMtool library:

```
library(DLMtool)
```

3.2 Set up parallel processing

```
setup()
```

```
## Warning in searchCommandline(parallel, cpus = cpus, type = type,
## socketHosts = socketHosts, : Unknown option on commandline:
## rmarkdown::render_site(output_format
```

```
## R Version:  R version 3.3.3 (2017-03-06)
```

```
## snowfall 1.84-6.1 initialized (using snow 0.4-2): parallel execution on 8 CPUs.
```

3.3 Summary

The above steps need to be done each time you start a new instance of R/RStudio and use the DLMtool. It is recommended that you start a new instance of R (by restarting RStudio) each time you begin a new analysis.

Chapter 4

Setting up the MSE Operating Model

4.1 Overview

The Operating Model (OM) is the main component of the MSE framework. The OM is used to simulate the population and fleet dynamics, the collection of data, and the application of a harvest control rule.

The class *OM* is used to define the parameters for the operating model. Remember, for help on classes you can type `class` followed by ? and the class name. For example, to see the help file for the *OM* class, type the following into the R console:

```
class?OM
```

4.2 The Operating Model Object

Objects of class *OM* contain all of the parameters required to run the MSE. Because this is a simulation, there must be values for all parameters.

You can see from the help file (`class?OM`) that objects of class *OM* can be constructed by using the `new()` function, and are built from three objects:

1. Stock,
2. Fleet,

3. Obs (Observation) and 4 Imp (Implementation).

Stock, ***Fleet***, ***Obs*** and ***Imp*** are all classes in the DLMtool, and, as the names suggest, contain the parameters for the fish stock, the fishing fleet, the observations of the data, and the implementation of the management recommendations respectively.

The *Stock*, *Fleet*, *Obs*, and *Imp* objects are contained within separate parameter tables, which are stitched together to form the *Operating Model* parameters.

You can look at the help files for each the these three classes for information on the **slots** for each object. Here we will briefly go through the contents of each.

4.2.1 Stock Object

First we will look at the contents of the *Stock* object. There are a number of stocks built into the DLMtool, and we will explore one of these. Later we will look at how to develop our own *Stock* objects, or modify an existing *Stock* object.

One of the *Stock* objects that is built into the DLMtool is `Albacore`.
If you type `Albacore` into the R console (or into your R script and click *Run*) you will see a large amount of information printed out in the R console.

Let's take a look at the contents of the Albacore Stock object:

```
slotNames(Albacore)
```

```
##  [1] "Name"         "maxage"       "R0"           "M"
##  [5] "Msd"          "Mgrad"        "h"            "SRrel"
##  [9] "Linf"         "K"            "t0"           "Ksd"
## [13] "Kgrad"        "Linfsd"       "Linfgrad"     "recgrad"
## [17] "a"            "b"            "D"            "Perr"
## [21] "Period"       "Amplitude"    "Size_area_1"  "Frac_area_1"
## [25] "Prob_staying" "AC"           "L50"          "L50_95"
## [29] "Source"
```

The output tells us that there are 29 slots in the `Albacore` Stock object. Each of these slots contains information relating to stock that is used in the MSE.

We can examine the information that is stored in the slots using the @ symbol. For example, the name of the species in the Stock object is:

```
Albacore@Name
```

```
## [1] "Abacore"
```

The maximum age parameter is:

```
Albacore@maxage
```

```
## [1] 15
```

The values for the natural mortality (M) parameter for this stock are:

```
Albacore@M
```

```
## [1] 0.35 0.45
```

Note that the natural mortality parameter (M) has two values, while the maximum age ($maxage$) only has one value.

The MSE in the DLMtool is a stochastic model, and almost all parameters are drawn from a distribution. By default this distribution is assumed to be uniform, and the two values for the M parameter represent the lower and upper bounds of this uniform distribution.

Some parameters, such as maximum age ($maxage$), species name ($Name$), or initial recruitment ($R0$) have only a single value and are fixed in the MSE.

You can see more information on the content of the *Stock* object by using the help function:

```
class?Stock
```

4.2.2 Fleet Object

While the *Stock* object contains all the information relating to the fish stock that is being modeled, the *Fleet* object is populated with information relating to the fishing fleet and historical pattern of exploitation.

Like the *Stock* objects, there are a number of *Fleet* objects that are built into the DLMtool. Here we will look at the `Generic_fleet` object.

```
slotNames(Generic_fleet)
```

```
##  [1] "Name"        "nyears"      "Spat_targ"   "Esd"         "qinc"
##  [6] "qcv"         "EffYears"    "EffLower"    "EffUpper"    "SelYears"
## [11] "AbsSelYears" "L5"          "LFS"         "Vmaxlen"     "L5Lower"
## [16] "L5Upper"     "LFSLower"    "LFSUpper"    "VmaxLower"   "VmaxUpper"
## [21] "isRel"       "CurrentYr"
```

There are 22 slots in the *Fleet* object. The parameters in the *Fleet* object relate to the exploitation pattern of the stock.

For example, the number of years that the stock has been exploited is specified in the `nyears` slot:

```
Generic_fleet@nyears
```

```
## [1] 50
```

As another example, the smallest length at full selection is specified in the `LFS` slot:

```
Generic_fleet@LFS
```

```
## [1] 0.75 1.10
```

Note that by default the values in the `LFS` (and the `L5` [smallest length at 5% selectivity]) slots are specified as multiples of the length of maturity (e.g., `Albacore@L50`). This is necessary because the *Fleet* objects built into the DLMtool are all generic, in the sense that they can be used with any *Stock* object.

You will notice that the `isRel` slot in the `Generic_fleet` object is set to “TRUE”. This means that the selectivity parameters are relative to the length of maturity in the *Stock* object. Absolute values for the selectivity parameters can be used, for example by specifying `LFS` and `L5` to, say, 100 - 150 and 50 - 70 respectively. The `isRel` parameter must then be set to “FALSE”, so that the Operating Model knows that these selectivity values are in absolute terms, and does not multiply them by the length of maturity (strange things may happen if the model assumes that the size of first capture is 50 to 70 times greater than the size of maturity!).

Note that all the parameters in the *Fleet* object have two values, representing the minimum and maximum bounds of a uniform distribution (with some exceptions that will be discussed in more detail later).

More information on the *Fleet* object can be found by typing:

```
class?Fleet
```

4.2.3 Observation Object

The third component for the *Operating Model* is the *Obs* object. This object contains all the information relating to how the fishery information is generated inside the model.

Why do we need a *Obs* table?

Although the MSE may be conditioned on real data and information about the fishery, all *data* is generated inside the model. Because it is a simulation model and the data was generated by a computer, rather than some unobserved real world process, the *fishery data* is known perfectly.In the real world, however, all data sources and parameter estimates are subject to some observation error. The degree of uncertainty may vary between different data types, and between different data-limited fisheries.

The advantage of the MSE process is that the performance of a management procedure using the realistically noisy simulated data can be compared to the performance under conditions of perfect knowledge. This comparison, which unfortunately is never possible in the real world, can reveal important information about the robustness (or sensitivity) of certain methods to variability and error in particular data types. This knowledge can help to prioritize research to reduce uncertainty in the parameters and data sets that are most crucial to the performance of the method.

Like the other two objects, there are a number of built-in *Obs* objects in the DLMtool.

Let's take a look at the `Imprecise_Unbiased` object:

```
slotNames(Imprecise_Unbiased)
```

```
##  [1] "Name"       "LenMcv"     "Cobs"       "Cbiascv"    "CAA_nsamp"
##  [6] "CAA_ESS"    "CAL_nsamp"  "CAL_ESS"    "CALcv"      "Iobs"
## [11] "Mcv"        "Kcv"        "t0cv"       "Linfcv"     "LFCcv"
## [16] "LFScv"      "B0cv"       "FMSYcv"     "FMSY_Mcv"   "BMSY_B0cv"
## [21] "rcv"        "Dbiascv"    "Dcv"        "Btbias"     "Btcv"
## [26] "Fcurbiascv" "Fcurcv"     "hcv"        "Icv"        "maxagecv"
## [31] "Reccv"      "Irefcv"     "Crefcv"     "Brefcv"     "beta"
```

There are 35 slots in *Obs* objects, each with information relating to the uncertainty of a data type.

For example, the `LenMcv` slot defines the uncertainty (coefficient of variability) in the length of maturity:

```
Imprecise_Biased@LenMcv
```

```
## [1] 0.2
```

This means that the assumed length of maturity that is generated by the Operating Model, and used in the simulated application of a management procedure, is not the 'true' value set in the *Stock* object, but a value sampled with a 20% coefficient of variation.

More information on the *Obs* object can be found by typing:

```
class?Obs
```

4.2.4 Implementation Object

The final component for the *Operating Model* is the *Imp* object. This object contains all the information relating to how the management recommendation is actually implemented in the fishery, i.e., the implementation error. The `Imp` object includes slots for the over or under catch of TAC, implementation error in total allowable effort, variability in size limits, and discard mortality.

The `Imp` object is a recent addition to DLMtool and may be expanded in future developments.

4.3 Existing Stock, Fleet, Obs, and Imp Objects

As mentioned earlier, there are several *Stock*, *Fleet*, *Obs* and *Imp* objects that are built into the DLMtool. A list of available *Stock*, *Fleet*, *Obs*, and *Imp* objects can be found using the `avail` function.

For example, here we will print out a list of the 12 *Stock* objects that are built into the DLMtool:

```
avail("Stock")
```

```
##  [1] "Albacore"           "Blue_shark"         "Bluefin_tuna"
##  [4] "Bluefin_tuna_WAtl" "Butterfish"         "Herring"
##  [7] "Mackerel"           "Porgy"              "Rockfish"
## [10] "Snapper"            "Sole"               "Toothfish"
```

We can do the same thing for the 15 *Fleet* objects (`avail("Fleet")`) and the 5 *Obs* objects (`avail("Obs")`).

4.4 Creating the Operating Model Object

The built-in *Stock*, *Fleet*, *Obs*, and *Imp* objects means that it is straightforward to create an *Operating Model* and run the MSE using the DLMtool.

The first step is to create the Operating Model object. This is done using the `new` command, and specifying that we want to create a new object of class `OM`.

Here we will create a Operating Model object called `myOM`, and use the `Blue_shark` *Stock* object, the `Generic_fleet` *Fleet* object, the `Imprecise_Biased` *Observation* object, and the default `Perfect_Imp` *Imp* object:

```
myOM <- new('OM', Blue_shark, Generic_fleet, Imprecise_Biased, Perfect_Imp)
```

If you use the `slotNames` function on the `myOM` object that was just created, you will see that it contains all of the information from the *Stock*, *Fleet*, and *Observation* objects.

We have now created a *Operating Model* object and can begin the MSE. If you wish, you can jump ahead to the Management Strategy Evaluation section (although you may wish to read the Management Procedures section first).

The remainder of this section will describe how to modify the existing *Stock*, *Fleet*, *Obs*, and *Imp* objects, as well as create new objects.

4.5 Customizing the Operating Model

The *Stock*, *Fleet*, *Obs*, and *Imp* objects that are built into the DLMtool can be used for running an MSE and exploring the output. More often, however, users of the DLMtool will be interested in performing an MSE for their own particular fishery.

To do this, the parameters of the *Operating Model* (or more specifically the *Stock*, *Fleet*, *Obs* and *Imp* objects from which it is built) must be specified in such a way that reflects the characteristics of the fishery.

There are a number of ways of doing this.

4.5.1 Creating New Objects

One method is to create empty *Stock*, *Fleet*, *Obs*, and *Imp* objects and manually populate them. This can be done using the `new` command:

```
myStock <- new("Stock")
myFleet <- new("Fleet")
myObservation <- new("Obs")
myImp <- new("Imp")
```

This code creates new *Stock*, *Fleet*, *Obs*, and *Imp* objects that are empty and contain no information.

You can check this by looking at a slot in one of the objects, for example:

```
myStock@M
```

```
## numeric(0)
```

The object can then be manually populated. For example,

```
myStock@M <- c(0.1, 0.25)
```

Here we set value of the natural mortality parameter (`M`) to range from 0.1 to 0.25.

The MSE requires values for each parameter (or slot) in the Operating Model object, and the three objects that were just created would have to be manually populated.

This is possible to do, but perhaps a little tedious. More importantly, it is easy to introduce errors using this method.

For example, let's try to populate the slot that contains information on the asymptotic size (`Linf`)

```
myStock@Linf <- 100
```

This seemed to work okay. However, if we try to run an MSE using this *Stock* object we will be confronted with an error. This is because the Operating Model requires **two** values (lower and upper bounds) for the `Linf` parameter. The DLMtool isn't smart enough (yet!) to detect this error before it is too late.

4.5.2 Importing Objects

There may be cases where creating a blank object is useful. More likely, however, is that users will import an object from a file, or modify a built-in object.

This can be done by using the `new` command and specifying the location of a comma-separated-variable (CSV) file that contains the relevant information.

4.5.2.1 Importing a Stock Object from a CSV File

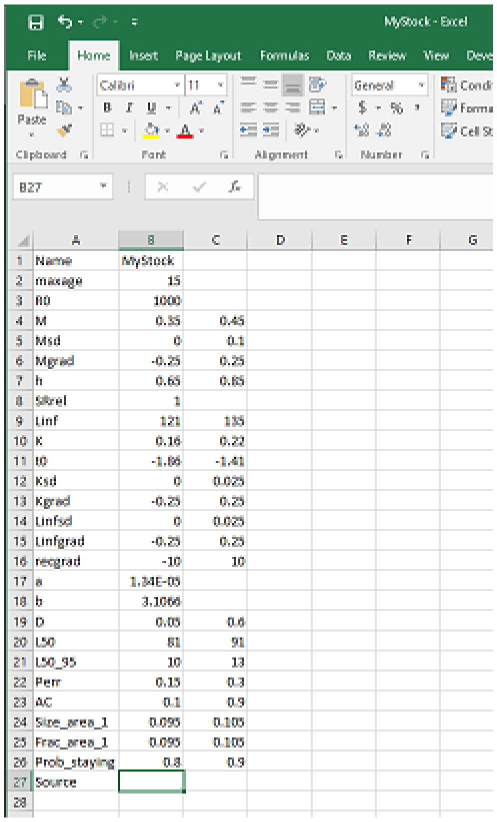
If we have a CSV file named *MyStock.csv* and stored in the directory *CSVs* (located within the working directory), we can import the *Stock* object using the following code:

```
myStock <- new("Stock", "CSVs/MyStock.csv")
```

The `myStock` object is now populated with the contents of the *MyStock.csv* (You may notice it is almost identical to the `Albacore` object that is in the DLMtool).

To import objects in this way, the CSV file must be set up in a specific format. The image below shows the contents of the *MyStock* CSV file. The first column contains the names of the slots in the *Stock* object. It is important that the text in the first column of the CSV file is *exactly* the same as the names slot of the slots in the *Stock* object (you can use `slotNames` to check this).

The second and third columns contain the values for each parameter. Note that some parameters, such as the length-weight relationship parameters (`a` and `b`), the maximum age (`maxage`), and the initial recruitment (`R0`) only require a single value.



	A	B	C
1	Name	MyStock	
2	maxage	15	
3	R0	1000	
4	M	0.35	0.45
5	Msd	0	0.1
6	Mgrad	-0.25	0.25
7	h	0.65	0.85
8	SRrel	1	
9	Linf	121	135
10	K	0.16	0.22
11	t0	-1.86	-1.41
12	Ksd	0	0.025
13	Kgrad	-0.25	0.25
14	Linfsd	0	0.025
15	Linfgrad	-0.25	0.25
16	recgrad	-10	10
17	a	1.34E-05	
18	b	3.1066	
19	D	0.05	0.6
20	L50	81	91
21	L50_95	10	13
22	Perr	0.15	0.3
23	AC	0.1	0.9
24	Size_area_1	0.095	0.105
25	Frac_area_1	0.095	0.105
26	Prob_staying	0.8	0.9
27	Source		
28			

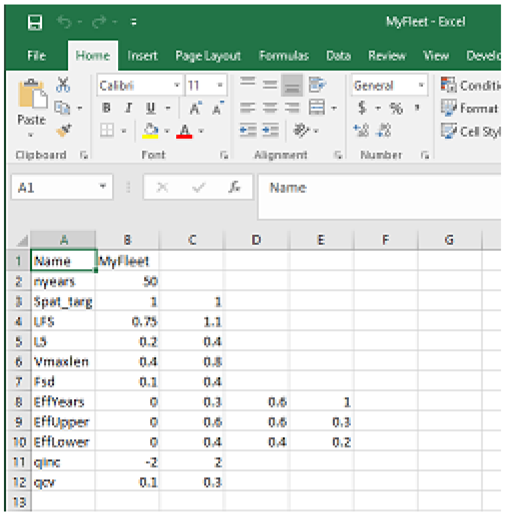
Click here for a larger version of the image.

4.5.2.2 Importing a Fleet Object from a CSV File

A *Fleet* object can be imported in a similar way:

```r
myFleet <- new("Fleet", "CSVs/myFleet.csv")
```

Note here again that the text in the first column must exactly match the names of the slots in the *Fleet* object.



	A	B	C	D	E
1	Name	MyFleet			
2	nyears	50			
3	Spat_targ	1	1		
4	LFS	0.75	1.1		
5	L5	0.2	0.4		
6	Vmaxlen	0.4	0.8		
7	Fsd	0.1	0.4		
8	EffYears	0	0.3	0.6	1
9	EffUpper	0	0.6	0.6	0.3
10	EffLower	0	0.4	0.4	0.2
11	qinc	-2	2		
12	qcv	0.1	0.3		
13					

Click here for a larger version of the image.

Note: If the DLMtool cannot find the CSV file in the specified directory, a warning message will appear and a blank object will be created.

4.5.2.3 Importing an Observation Object from a CSV File

To import a *Obs* object from a CSV file you follow the same procedure:

```
myObs <- new("Obs", "CSVs/myObs.csv")
```

Note, however, that the *Obs* object is large (35 parameters) and it is often easiest to modify an existing *Obs* object.

We can now stitch the three objects together into a new Operating Model:

```
myOM <- new("OM", myStock, myFleet, myObs, myImp)
```

In this example we did not include our newly made *Imp* object. By default the `new("OM")` command includes the `Perfect_Imp` Imp object.

4.6 Modifying Existing Objects

Often the simplest way to creating a custom *Operating Model* object is to modify the existing objects in the DLMtool.

For example, suppose we wish to do a MSE for a species that we know has a similar life history to herring. Using the existing `Herring` *Stock* object is a good place to start.

First we create a new object by copying the existing `Herring` object:

```
MyStock <- Herring
```

Then we can modify the parameters so that it better suits our particular species. For example, we may wish to modify the asymptotic size (`Linf`), the von Bertalanffy K parameter (`K`), and the length at maturity (`L50` and `L50_95`):

```
MyStock@Linf <- c(60, 80)
MyStock@K <- c(0.2, 0.25)
MyStock@L50 <- c(30, 40)
MyStock@L50_95 <- c(10, 15)
```

We should also change the `Name` and `Source` for our stock:

```
MyStock@Name <- "Herringish"
MyStock@Source <- "None. I made this up"
```

You can continue modifying the remaining parameters as required.

The *Fleet* and *Obs* objects can be modified in from existing objects in a similar way. First we copy existing objects, and then modify them to suit our needs:

```
MyFleet <- FlatE_Dom # Dome-shaped selectivity and flat effort trajectory
MyFleet@L5 <- c(15, 20) # change the selectivity pattern
MyFleet@LFS <- c(25, 30)
MyFleet@isRel <- "FALSE" # remember to change this

MyObs <- Imprecise_Unbiased
MyObs@Mcv <- 0.15 # increase uncertainty in M
```

Here we have modified a few parameters in the *Stock*, *Fleet*, and *Obs* objects and assumed that the rest of the values in built-in objects adequately describe our fishery. We can now create an *Operating Model* object using the same method described previously:

```r
MyOM <- new("OM", MyStock, MyFleet, MyObs, myImp)
```

4.7 More on Customizing the Operating Model

In some circumstances there may be knowledge on the changes in fishing practices over the years, and it would be good to include this information in the *Operating Model*.

The *Operating Model* can be conditioned with respect to historical trends in the fishing mortality, and historical changes in the selectivity pattern.

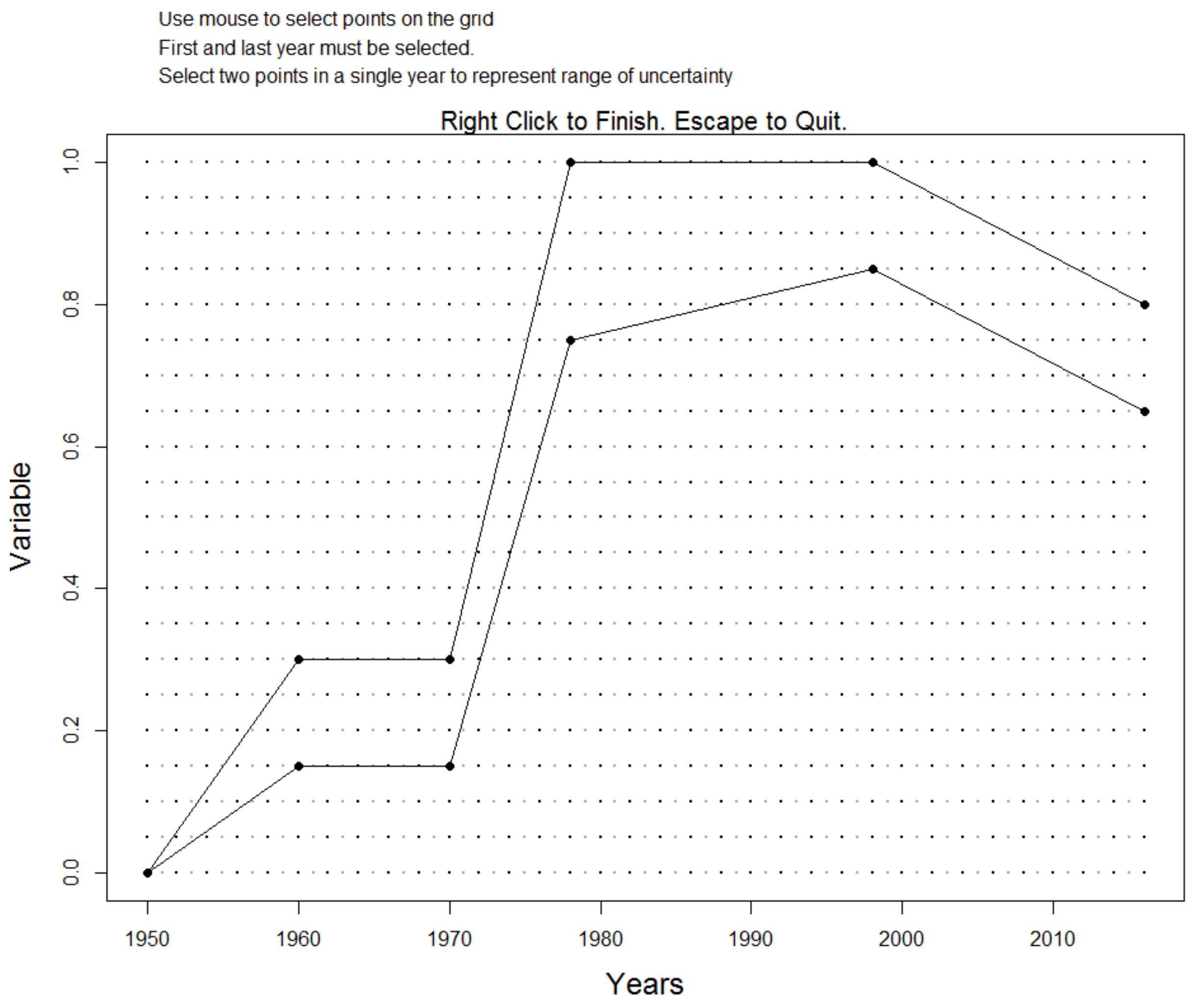
4.7.1 Historical Trends in Fishing Mortality

Suppose that we know the fishery began in 1950, and fishing effort increased slowly over the next decade, was relatively stable between 1960 and 1970, then increased dramatically over the next 10 years. We also know that, while fishing effort stayed relatively constant from 1980 to around 2000, there has been a general decline in fishing effort in recent years.

This information can be included in the *Operating Model* by using the `ChooseEffort` function. The `ChooseEffort` function takes an existing *Fleet* object as its first argument, and allows the user to manually map out the range for the historical trend in fishing effort. The `ChooseEffort` function then returns the updated *Fleet* object.

A second optional argument can be used to specify the historical years. If used, this will replace the `nyears` in the *Fleet* object with the length of the `Years` vector.

```r
MyFleet <- ChooseEffort(MyFleet, Years=1950:2016)
```



If we take a look at the `MyFleet` object, we will see that three slots `EffYears`, `EffLower` and `EffUpper` have been replaced with the new values.

Note that the trajectory that is mapped out here represents the bounds on the relative fishing mortality for each year. In this example, the fishing mortality rate was highest (on average) between 1980 and 2000, and is currently around 65 - 80% of this maximum level.

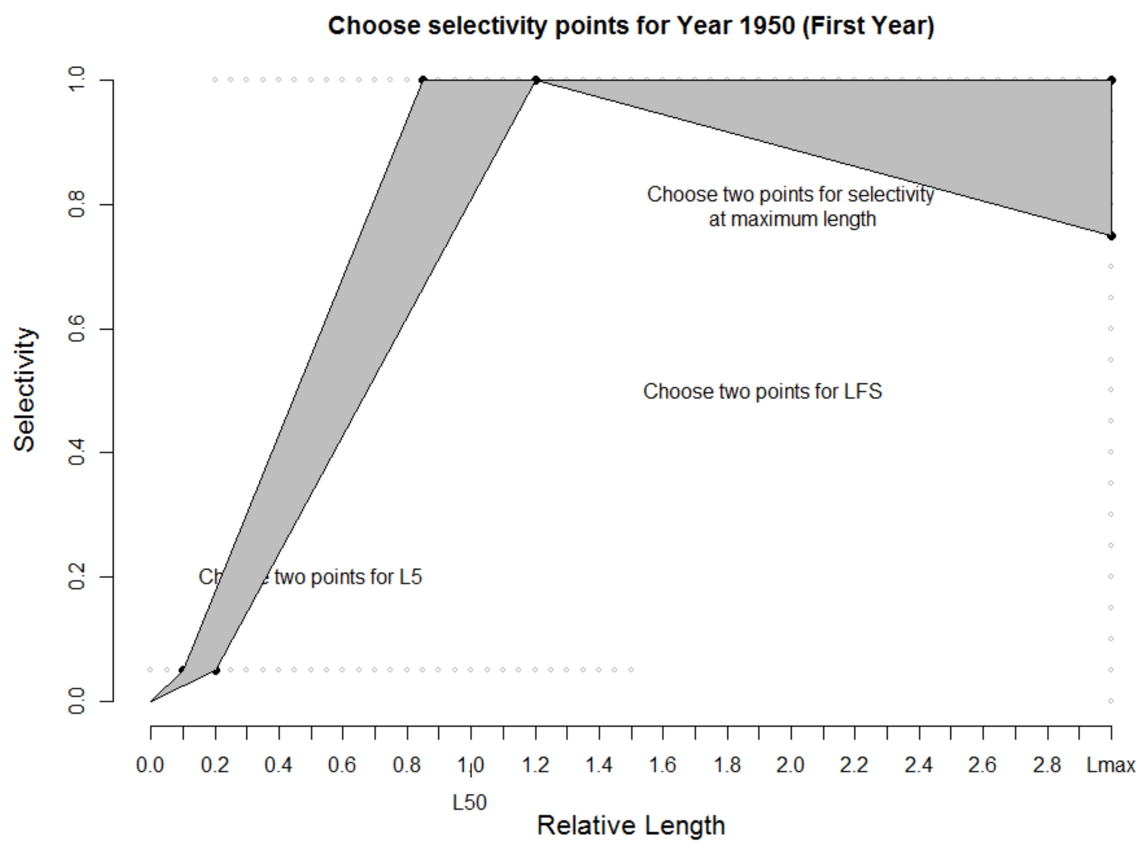
4.7.2 Historical Trends in the Selectivity Pattern

Suppose that we may knew there had been changes in the selectivity pattern of the fishery over time. This information can be included in the *Operating Model* by using the `ChooseSelect` function.

Like the `ChooseEffort` function described above, the `ChooseSelection` function takes a *Fleet* objects as it's first argument, and returns an updated *Fleet* object.

Suppose the selectivity pattern changed in 1970 and then again in 1990, perhaps because of changes in fishing regulations. These change points in the selectivity curve can be mapped by the following command:

```
MyFleet <- ChooseSelect(MyFleet, FstYr=1950, SelYears=c(1970, 1990))
```



Note that the first year (`FstYr`) must also be specified, and the selectivity pattern is mapped for this year as well.

When `ChooseSelect` is used, the `L5Lower`, `L5Upper`, `LFSLower`, `LFSUpper`, `VmaxLower`, `VmaxUpper`, and `SelYears` slots are updated in the *Fleet* object. If these slots are populated, the values in the `L5`, `LFS`, and `Vmaxlen` slots are ignored in the operating model.

Chapter 5

Management Procedures

The purpose of an MSE is to compare the performance of alternative management approaches, or ***Management Procedures*** to identify the method that is most likely to meet the management objectives for the fishery.

5.1 What is a Management Procedure?

In essence, a Management Procedure is simply a set of rules which define how a fishery will be managed. These rules can range from simple harvest policies to more complex arrangements.

For example, a simple Management Procedure may be a constant catch policy, where the annual total allowable catch (TAC) is set a some fixed value. Alternatively, a more complex Management Procedure may involve multiple data sources, with rules that increase or reduce the TAC in response to trends in one or several indicators.

Management Procedures can differ in data requirements and complexity. However, all Management Procedures have one thing in common. They take fishery information and return a management recommendation.

To be included in an MSE, a Management Procedure must be reproducible and able to be coded in a set of instructions. While fisheries are sometimes managed by expert judgment, it is difficult to reproduce the subjective decision-making process in a computer simulation and include such methods in an MSE.

5.2 Management Procedure Classes

Management Procedures in the DLMtool are divided into two classes: **Output controls** and **Input controls**.

5.2.1 Output Controls

The output control methods in the DLMtool provide a management recommendation in the form of a TAC. Some output controls are stochastic, allowing for uncertainty in the data or input parameters, and return a distribution of recommended TACs.

Output control methods are very common in fisheries management, especially in regions which have a tradition of managing fisheries by regulating the total amount of catch.

5.2.2 Input Controls

The input control methods in the DLMtool allow regulation of **fishing effort**, **size selecitivty**, or **spatial area**.

5.2.2.1 Effort Controls

Effort controls adjust the relative amount of fishing effort. In the DLMtool, all effort controls are designed to adjust effort relative to the current levels of effort. For example, an input control method may recommend a 15% increase or decrease in fishing effort in response to the signal in some fishery data.

5.2.2.2 Selectivity Controls

Selectivity controls provide recommendations on the size selection pattern for the fishery. Selectivity controls can be static, for example a single fixed size limit, or dynamic, where a selectivity pattern is modified in response to data.

The DLMtool can account for selectivity controls in the form of a minimum legal length or a harvest slot limit (upper size limit). A fixed size limit is a Management Procedure, but typically does not require any fishery data to implement the rule.

5.2.2.3 Spatial Controls

Spatial controls in the DLMtool allow for the opening and closing of spatial areas to fishing activity. Spatial controls can be static (permanent closed area) or dynamic with an area opening and closing to fishing in response to indicators in the fishery data.

5.3 Built-in Management Procedures

The DLMtool has a large number of Management Procedures that are built into the Toolkit. Many of these methods have been published in the scientific literature as recommended methods for managing data-limited fisheries. Other methods have been developed more recently and have been included in the DLMtool to evaluate if they have potential to be useful for management.

The DLMtool has been designed to be extensible, and it is straightforward to develop additional Management Procedures and include them in the MSE.

Details on how to develop new Management Procedures are provided in sections below.

5.3.1 Output Controls

Output control methods (methods that return a TAC) in the DLMtool are of the class `DLM_output`. You can use the `avail` function to list the available output controls in the DLMtool:

```r
avail("Output")
```

```
##  [1] "AvC"         "BK"          "BK_CC"       "BK_ML"       "CC1"
##  [6] "CC4"         "CompSRA"     "CompSRA4010" "DAAC"        "DBSRA"
## [11] "DBSRA_40"    "DBSRA_ML"    "DBSRA4010"   "DCAC"        "DCAC_40"
## [16] "DCAC_ML"     "DCAC4010"    "DD"          "DD4010"      "DepF"
## [21] "DynF"        "Fadapt"      "Fdem"        "Fdem_CC"     "Fdem_ML"
## [26] "FMSYref"     "FMSYref50"   "FMSYref75"   "Fratio"      "Fratio_CC"
## [31] "Fratio_ML"   "Fratio4010"  "GB_CC"       "GB_slope"    "GB_target"
## [36] "Gcontrol"    "HDAAC"       "Islope1"     "Islope4"     "IT10"
## [41] "IT5"         "Itarget1"    "Itarget4"    "ITM"         "L95target"
## [46] "LBSPR_ItTAC" "LstepCC1"    "LstepCC4"    "Ltarget1"    "Ltarget4"
## [51] "MCD"         "MCD4010"     "NFref"       "Rcontrol"    "Rcontrol2"
## [56] "SBT1"        "SBT2"        "SPmod"       "SPMSY"       "SPslope"
## [61] "SPSRA"       "SPSRA_ML"    "YPR"         "YPR_CC"      "YPR_ML"
```

As you can see, there are 65 output control methods built into the DLMtool.

5.3.2 Input Controls

The list of the 24 available input control methods (class `Input`) can be displayed in the same way:

```
avail("Input")
```

```
##  [1] "curE"        "curE75"      "DDe"         "DDe75"       "DDes"
##  [6] "DTe40"       "DTe50"       "EtargetLopt" "ItargetE1"   "ItargetE4"
## [11] "ITe10"       "ITe5"        "LBSPR_ItEff" "LBSPR_ItSel" "LstepCE1"
## [16] "LstepCE2"    "LtargetE1"   "LtargetE4"   "matlenlim"   "matlenlim2"
## [21] "minlenLopt1" "MRnoreal"    "MRreal"      "slotlim"
```

5.4 Details of the Methods

Information on each Management Procedure can be found be using the `?` function. For example, to access the documentation associated with the `CC1` method, you would type `?CC1` into the R console.

The help documentation provides a description of the method, the arguments for the function, and the output of the function, together with references to the scientific literature that are associated with the method.

5.5 Designing New Methods

DLMtool was designed to be extensible in order to promote the development of new Management Procedures. In this section we design a series of new Management Procedures that include spatial controls and input controls in the form of size limit restrictions. The central requirement of any MP is that it can be applied to a `Data` object using the function `sapply` (`sfSapply()` in parallel processing).

`Data` objects have a single position `x` for each data entry (e.g. one value for natural mortality rate, a single vector of historical catches, etc.). In the MSE analysis this is extended to `nsim` positions.

See the Managing Fishery Data section for more information about the `Data` object.

It follows that any MP arranged to function `sapply(x,MP,Data)` will work. For example we can get five stochastic samples of the TAC for the demographic FMSY MP paired to catch-curve analysis `FdemCC` applied to a real data-limited data object for red snapper using:

```
sapply(1,Fdem_CC,Red_snapper,reps=5)
```

```
##           [,1]
## [1,] 20.567294
## [2,]  2.925829
## [3,]  9.306221
## [4,]  5.164635
## [5,]  3.487370
```

The MSE populates a `Data` object with many simulations and uses `sfSapply()` (snowfall cluster computing equivalent) to calculate a management recommendation for each simulation. By making methods compatible with this standard the very same equations are used in both the MSE and the real management advice.

5.5.1 Average Historical Catch Management Procedure

The average historical catch has been suggested as a starting point for setting TACs in the most data-limited situations (following Restrepo et al., 1998).

Here we design such an Management Procedure:

```
AvC <- function(x, Data, reps) {
  rlnorm(reps, log(mean(Data@Cat[x,], na.rm=T)), 0.1)
}
```

Note that all Management Procedures have to be stochastic in this framework, which is why we sample from a log-normal distribution with a CV of roughly 10 per cent.

Before the Management Procedure can be 'seen' by the rest of the DLMtool package we have to do three more things.

The Management Procedure must be assigned a class based on what outputs it provides. Since this is an output control (TAC) based MP we assign it class `Output`. The Management Procedure must also be assigned to the DLMtool namespace:

```
class(AvC) <-"Output"
environment(AvC) <-asNamespace('DLMtool')
```

and - if we are using parallel computing - exported to the cluster:

```
sfExport("AvC")
```

5.5.2 Length-at-Selection Set Equal to Length-at-Maturity

To simulate input controls that aim to alter the length-vulnerability to fishing it is possible to design a Management Procedure of class `Input`.

In this example we set selectivity equal to the maturity curve, while the spatial and effort regulations remain constant:

```
matlenlim <- function (x, Data, ...) {
    Allocate <- 1
    Effort <- 1
    Spatial <- c(1, 1)
    newLFC <- Data@L50[x] * 0.95
    newLFS <- Data@L50[x]
    Vuln <- c(newLFC, newLFS)
    c(Allocate, Effort, Spatial, Vuln)
}
class(matlenlim) <- "Input"
environment(matlenlim) <- asNamespace("DLMtool")
```

and export to cluster:

```
sfExport("matlenlim")
```

5.5.3 Harvest Slot Limit

We can also choose to create a Management Procedure with a harvest slot limit. This example sets a minimum legal length at 1.1 times the size of maturity, and a maximum size limit at 75% of the distance between the minimum legal length and the estimate of asymptotic length recorded in the fishery data object:

```
slotlim <- function (x, Data, ...) {
    Allocate <- 1
    Effort <- 1
    Spatial <- c(1, 1)
    newLFS <- 1.1 * Data@L50[x]
    newLFC <- 0.95 * newLFS
    UppLim <- as.numeric(quantile(c(newLFS, Data@vbLinf[x]), 0.75))
    Vuln <- c(newLFC, newLFS, UppLim)
    c(Allocate, Effort, Spatial, Vuln)
}
class(slotlim) <- "Input"
environment(slotlim) <- asNamespace("DLMtool")
```

and export to cluster:

```
sfExport("slotlim")
```

Note that the arguments for the input methods must include either `reps` or `...`, even if these are not used.

5.5.4 Reducing Fishing Rate in area 1 by 50%

Spatial controls operate similarly to the age/size based controls: a vector of length 2 (the spatial simulator is a 2-box model) that indicates the fraction of current spatial catches. In this example we reduce effort in area 1 by 50%.

```
area1_50 <- function(x,Data, ...){
  Allocate <- 0 # Fraction of effort reallocated to open area
  Effort <- 1  # Fraction of effort in last historical year
  Spatial <- c(0.5,1) # Fraction of effort found in each area
  Vuln <- rep(NA,2) # Length vulnerability is not specified
  c(Allocate, Effort, Spatial, Vuln) # Input controls stitched togther
}
class(area1_50)<-"Input"
environment(area1_50) <- asNamespace('DLMtool')
```

```
sfExport("area1_50")
```

Chapter 6

Management Strategy Evaluation

The last two chapters described how to set up an Operating Model object and to select or develop Management Procedures.

This chapter will describe how to run the Management Strategy Evaluation using the DLMtool

6.1 Re-cap on Setting up DLMtool

Chapter 3 describes the necessary to set up the DLMtool. In brief, the following five lines should appear at the top of your script whenever you are using the DLMtool to run a MSE:

```r
library(DLMtool)
setup()
```

6.2 Defining the Operating Model

The operating model is the 'simulated reality': a series of known simulations for testing various data-limited Management Procedures. Operating models can either be specified in detail according to each variable (e.g. sample natural mortality rate between 0.2 and 0.3) or alternatively the user can rapidly construct an operating model based on a set of predefined `Stock`, `Fleet` and `Observation` models. See Chapter 4 for more details on the operating model.

In this case we take the latter approach and pick the `Blue_shark` stock type, a generic fleet type and an observation model that generates data that can be both imprecise and biased. We will ignore implementation error and use the default `Perfect_Imp`.

```r
OM <- new('OM', Blue_shark, Generic_fleet, Imprecise_Biased, Perfect_Imp)
```

6.3 Choose Management Procedures

The MSE can be run either with all of the Management Procedures contained within DLMtool or a subset of methods, for example, only the output control class of methods. As an example, if you wanted to choose four output controls, and one input control method, you would enter the following code. See Chapter 5 for more details on the Management Procedures, including how we created the `matlenlim` input control method.

```
MPs <- c("Fratio", "DCAC", "Fdem", "DD", "matlenlim")
```

6.4 The runMSE Function

The MSE is run using the `runMSE` function. In addition to the `OM` (Operating Model) and `MPs` (Management Procedures) arguments, the `RunMSE` function has a number of other arguments to control the MSE.

You can access the help file for the `runMSE` function by typing: `?runMSE` into the R console.

6.4.1 Description of the Arguments

Here we briefly describe the main arguments for the `runMSE` function:

- `OM` - An object of class `OM`. Must be specified, see Chapter 4 for details.
- `MPs` - A character vector specifying the Management Procedures to be included in the MSE. If not specified, the DLMtool will run the MSE for all available methods in the Toolkit.
- `nsim` - Number of simulations
- `proyears` - Number of projected years
- `interval` - The assessment interval - how often would you like to update the management system?
- `pstar` - The percentile of the sample of the management recommendation for each method
- `maxF` - Maximum instantaneous fishing mortality rate that may be simulated for any given age class
- `timelimit` - Maximum time taken for a method to carry out 10 reps (methods are ignored that take longer)
- `reps` - The number of samples of the management recommendation for each method. Note that when this is set to 1, the mean value of the data inputs is used
- `CheckMPs` - A logical value to specify if the DLMtool should check if the Management Procedures can be run before running the entire MSE. Mainly used for developing and testing new methods.

6.4.2 Using runMSE

Most of the arguments for the `runMSE` function have default values. The most important parameters to specify are the Operating Model object, the Management Procedures to include, the number of simulations, the number of projection years, and the interval for management.

Here we will demonstrate running the blue shark operating model that we created above, for the five Management Procedures selected above. Note that in this example we have chosen to include 10 simulations (`nsim`=10) and left the other parameters at the default values.

```
BSharkMSE <- runMSE(OM=OM, MPs=MPs, nsim=10)
```

```
## Loading operating model

## Optimizing for user-specified movement

## Optimizing for user-specified depletion

## Calculating historical stock and fishing dynamics

## Calculating MSY reference points

## Calculating reference yield - best fixed F strategy

## 1/5 Running MSE for Fratio

## ..................................................
## 2/5 Running MSE for DCAC

## ..................................................

## 3/5 Running MSE for Fdem

## ..................................................

## 4/5 Running MSE for DD

## ..................................................

## 5/5 Running MSE for matlenlim

## ..................................................
```

Note that this is just a demonstration, in a real MSE you should use many more simulations (`nsim` more than 200), `reps` (samples per method more than 100) and perhaps a more frequent assessment interval (`interval` of 2 or 3 years).

Note that when `reps` is set to 1, all stochastic MPs use the mean value of an input and do not sample from the distribution according to the specified CV (the `Output` MPs become deterministic and no longer produce samples of the TAC recommendation).

6.4.3 Robust Wrapper Function

The above example, with 10 simulations and 5 Management Procedures, should take about 47 seconds to complete. A real MSE would include many more simulations and take significantly longer.

The `runMSErobust` function has been designed as a more efficient way to run the MSE for a large number of simulations. The `runMSErobust` function splits the MSE into a number of individual packets.

The advantages are that it is more efficient (i.e., quicker), it automatically saves the final MSE object to the hard disk, and re-starts a packet if the MSE model crashes. Furthermore, he individual packets can be saved so if the model crashes, or you suffer a power outage the work is not lost.

It is recommended to use the `runMSErobust` function if you are running a large number of simulations. However, it is always a good idea to run a small number of simulations first, to test that the MSE is running correctly, and not crashing due to mis-specified parameters or bugs in the Management Procedure code.

The `runMSErobust` function is essentially a wrapper for the `runMSE` function and takes the same arguments as `runMSE`. The `runMSErobust` function has several additional arguments, including:

- `maxsims` - The maximum number of simulations to include in each packet.
- `name` - The file name of the MSE object that is saved to the hard disk.
- `maxCrash` - The maximum number of consecutive crashes before the MSE stops.
- `saveMSE` - A logical value (`TRUE`/`FALSE`) to indicate if final `MSE` object should be saved to current working directory (this is probably a good idea).
- `savePack` - A logical value (`TRUE`/`FALSE`) to indicate if the packets should be save to current working directory.

Here we will run the blue shark MSE with 300 simulations. Note that the output is not printed out in the User Manual.

```
BSharkMSE <- runMSErobust(OM=OM, MPs=MPs, nsim=300, name="blueshark")
```

This run should be completed in 10 minutes or less. The MSE object was saved (using the `saveRDS` function) to the current working directory as a `rdata` object named ‘bluesharkMSE.rdata’.

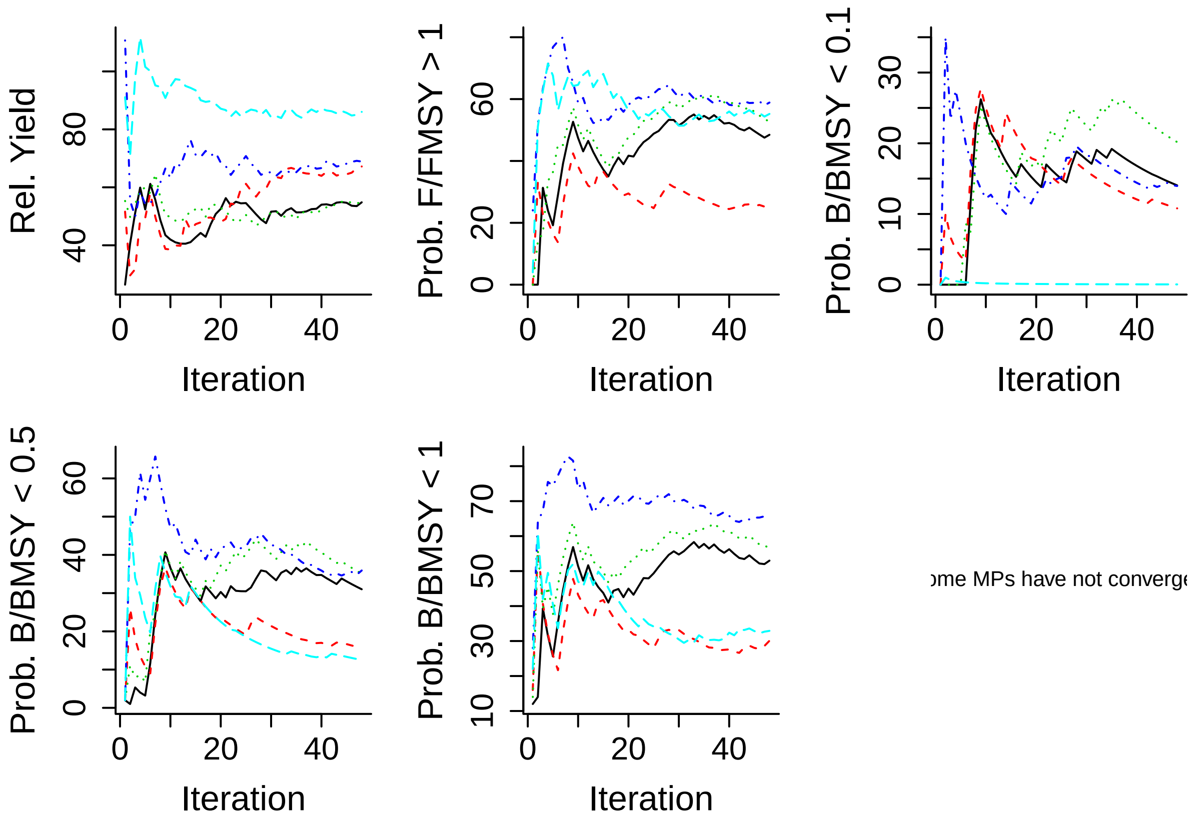
A saved MSE object can be loaded using the `readRDS` function:

```
BSharkMSE <- readRDS("bluesharkMSE.rdata")
```

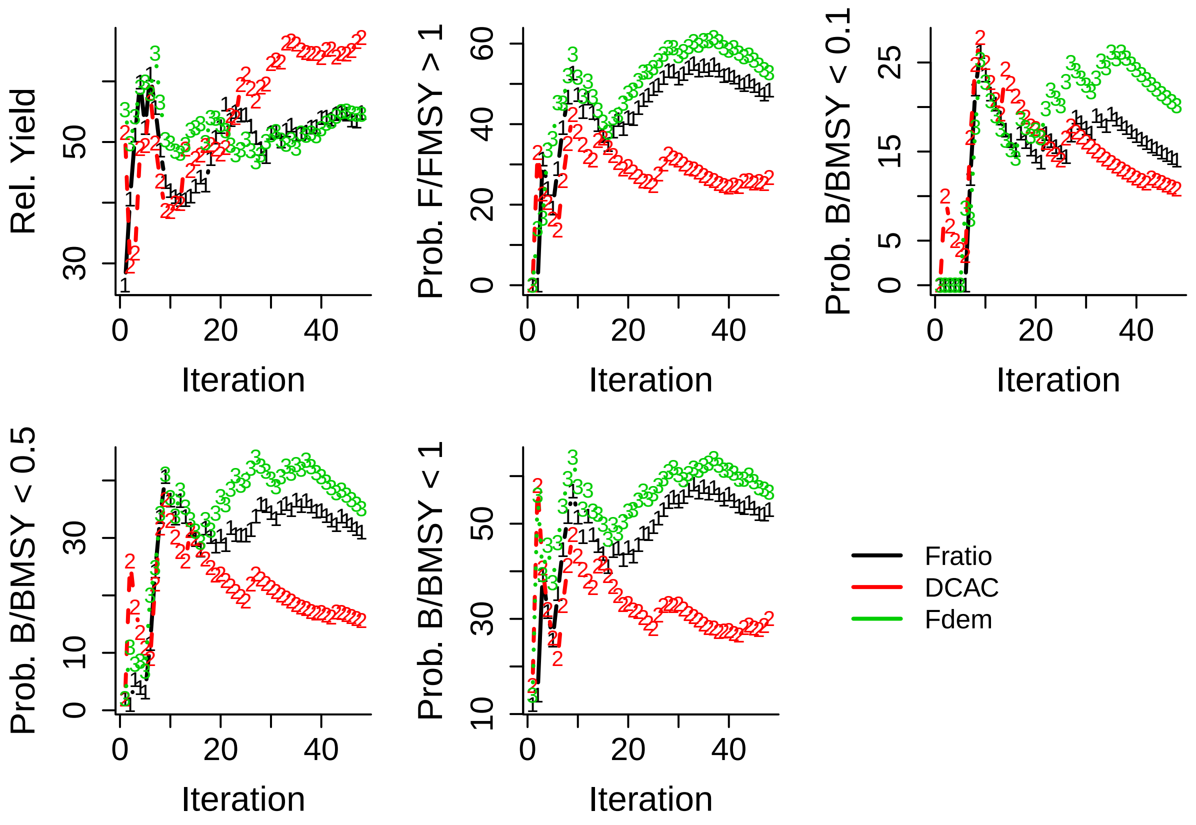
6.5 Checking Convergence

The MSE is now complete. We can use the `Converge` function to confirm that the number of simulations is sufficient and the MSE model has converged, by which we mean that the relative position of the Management Procedures are stable with respect to different performance metrics:

```
Converge(BSharkMSE)
```



```
## Some MPs may not have converged in 48 iterations (threshold = 2%)

## MPs are: Fratio  DCAC  Fdem

## MPs #: 1  2  3

##    Num     MP
## 1    1 Fratio
## 2    2   DCAC
## 3    3   Fdem
```



6.6 A Quick Recap

The next step is exploring the output of the MSE, and determining which Management Procedure best meets the management objectives.

The information above is spread out over a number of sections and paragraphs, and may appear a little complex. In fact, other than the initial set up of the DLMtool, it only took 4 lines of code to run the MSE and check the model for convergence:

```
OM <- new('OM', Blue_shark, Generic_fleet, Imprecise_Biased)
MPs <- c("Fratio", "DCAC", "Fdem", "DD", "matlenlim")
BSharkMSE <- runMSErobust(OM=OM, MPs=MPs, nsim=300, name="blueshark")
CheckConverg(BSharkMSE)
```

In the next Chapter we will look at how to examine the output of the MSE.

Chapter 7

Examining the MSE object

In the Chapter 6 we used the DLMtool to run a Management Strategy Evaluation for the blue shark stock.

The output of the MSE was saved to an object, of the class `MSE`, called `BSharkMSE`. You should be able to create the `BSharkMSE` object following the steps outlined in the previous chapters.

Alternatively you can download the 'BSharkMSE' `MSE` object and save to your working directory.

You may be able to import the MSE data file directly into R by executing the following command, although this may only working on some operating systems:

```
BSharkMSE <- readRDS(gzcon(url("https://github.com/DLMtool/DLMtool_Data/raw/master/bluesharkMSE.rdata"))
```

Note that objects of class `MSE` contain a lot of information and can be quite large in size. Downloading from the above link may take some time, depending on the speed of your internet connection.

7.1 The MSE Object

The names of the slots in an object of class `MSE` can be displayed using the `slotNames` function:

```
slotNames(BSharkMSE)
```

```
##  [1] "Name"     "nyears"   "proyears" "nMPs"     "MPs"      "nsim"
##  [7] "OM"       "Obs"      "B_BMSY"   "F_FMSY"   "B"        "SSB"
## [13] "VB"       "FM"       "C"        "TAC"      "SSB_hist" "CB_hist"
## [19] "FM_hist"  "Effort"   "PAA"      "CAA"      "CAL"      "CALbins"
```

As you can see, `MSE` objects contain all of the information from the MSE, stored in 24 slots.

7.1.1 The First Six Slots

The first six slots contain information on the structure of the MSE. For example the first slot (`Name`), is a combination of the names of the `Stock`, `Fleet`, and `Obs` objects that were used in the MSE:

```
BSharkMSE@Name
```

```
## [1] "Stock:Blue shark  Fleet:Generic_fleet  Obs model:Imprecise-Biased  Imp model:Perfect implementa
```

Other information in these first slots includes the number of historical years (`nyears`), the number of projection years (`proyears`), the number of name of the Management Procedures (`nMPs` and `MPs`), and the number of simulations (`nsim`).

7.1.2 The `OM` Slot

The `OM` slot in the `MSE` object is a data frame that the values of the parameters used in the Operating Model:

```
names(BSharkMSE@OM)
```

```
##  [1] "RefY"         "M"            "Depletion"    "A"
##  [5] "SSBMSY_SSB0"  "FMSY_M"       "Mgrad"        "Msd"
##  [9] "procsd"       "Esd"          "dFfinal"      "MSY"
## [13] "qinc"         "qcv"          "FMSY"         "Linf"
## [17] "K"            "t0"           "hs"           "Linfgrad"
## [21] "Kgrad"        "Linfsd"       "recgrad"      "Ksd"
## [25] "ageM"         "L5"           "LFS"          "Vmaxlen"
## [29] "LFC"          "OFLreal"      "Spat_targ"    "Frac_area_1"
## [33] "Prob_staying" "AC"           "L50"          "L95"
## [37] "B0"           "N0"           "SSB0"         "BMSY_B0"
## [41] "TACSD"        "TACFrac"      "ESD"          "EFrac"
## [45] "SizeLimSD"    "SizeLimFrac"  "DiscMort"     "Blow"
```

If you use the `dim` function to report the dimensions of the `OM` data frame, you'll notice that there are 48 columns, corresponding to the 48 parameters in the Operating Model, and 48 rows, each corresponding to a single simulation of the MSE.

7.1.3 The `Obs` Slot

The `Obs` slot contains another data frame, this one with 25 columns corresponding to the values drawn from the Observation model:

```
names(BSharkMSE@Obs)
```

```
##  [1] "Cbias"       "Csd"         "CAA_nsamp"   "CAA_ESS"     "CAL_nsamp"
##  [6] "CAL_ESS"     "Isd"         "Dbias"       "Derr"        "Mbias"
## [11] "FMSY_Mbias"  "BMSY_B0bias" "lenMbias"    "LFCbias"     "LFSbias"
## [16] "Abias"       "Aerr"        "Kbias"       "t0bias"      "Linfbias"
## [21] "hbias"       "Irefbias"    "Crefbias"    "Brefbias"    "betas"
```

The `Obs` data frame also has 48 rows, each corresponding to a single simulation.

The information contained in the `OM` and `Obs` slots can be used to examine the sensitivity of the performance of Management Procedures with respect to different operating model and observation parameters. This is discussed in more detail below.

7.1.4 The `B_BMSY` and `F_FMSY` Slots

The `B_BMSY` and `F_FMSY` are data frames containing the biomass relative to biomass at maximum sustainable yield $\left(\frac{B}{B_{MSY}}\right)$, and fishing mortality relative to the rate corresponding to maximum sustainable yield $\left(\frac{F}{F_{MSY}}\right)$ for each simulation, Management Procedure and projection year.

If we look at the class of the `B_BMSY` slot, we see that it is an `array`:

```
class(BSharkMSE@B_BMSY)
```

```
## [1] "array"
```

Using the `dim` function we can see that it is a 3-dimensional array, with the size corresponding to the number of simulations (`nsim`), the number of Management Procedures (`nMPs`), and the number of projection years (`proyears`):

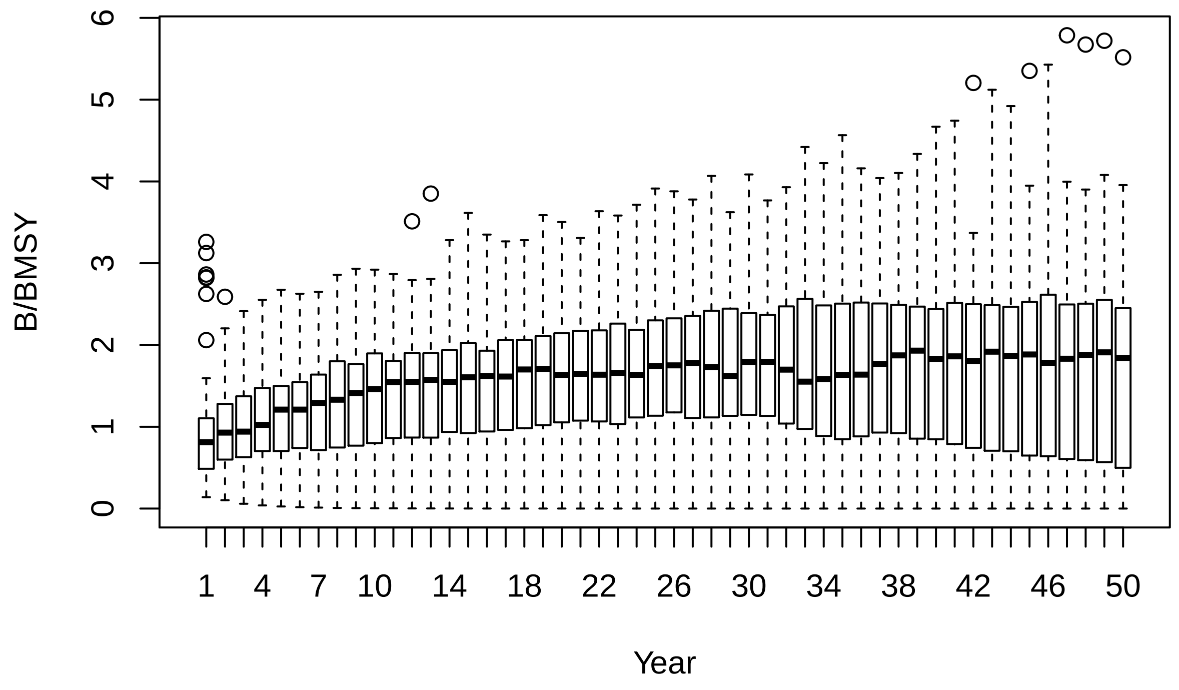
```
dim(BSharkMSE@B_BMSY)
```

```
## [1] 48  5 50
```

This information can be used to calculate statistics relating to the performance of each Management Procedure with respect to these metrics.

For example, if you wish to look at the distribution of $\frac{B}{B_{MSY}}$ for the second Management Procedure (DCAC), you could use the `boxplot` function:

```
boxplot(BSharkMSE@B_BMSY[,2,], xlab="Year", ylab="B/BMSY")
```



This plot shows that the relative biomass for the stock generally increases through the projection period when the DCAC method is used, with the median relative biomass increasing from about 0.81 in the first year to 1.84 in the final year.

However, the distribution appears to have quite high variability, which suggests that although the method works well on average, the final biomass was very low in some simulations.

We will look at more aspects of plotting the MSE results in the sections below.

7.1.5 The `B`, `FM`, and `C` Slots

The `B`, `FM`, and `C` slots contain the information relating to the stock biomass, the fishing mortality rate, and the catch for each simulation, Management Procedure, and projection year.

Typically, the MSE model in the DLMtool does not include information on the absolute scale of the stock biomass or recruitment, and all results usually must be interpreted in a relativistic context.

This is particularly true for the biomass (`B`) and catch (`C`) where the absolute values in the MSE results (other than 0!) have little meaning.

The biomass can by made relative to B_{MSY}, as shown above. Alternatively, biomass can be calculated with respect to the unfished biomass (B_0), from information stored in the `OM` slot.

The catch information is usually made relative to the highest long-term yield (mean over last five years of projection) for each simulation obtained from a fixed F strategy. This information (`RefY`) can be found in the `OM` slot.

Alternatively, the catch can be made relative to the catch in last historical year (`CB_hist`; see below), to see how future catches are expected to change relative to the current conditions.

Examples of this are shown in sections below.

7.1.6 The `TAC` Slot

Currently, the information in this slot is identical to that in the Catch (`C`) slot. This may change when an implementation error model is included in the DLMtool.

7.1.7 The `SSB_hist`, `CB_hist`, and `FM_hist` Slots

The `SSB_hist`, `CB_hist`, and `FM_hist` slots contain information on the spawning stock biomass, the catch biomass, and the fishing mortality from the historical period (the `nyears` in the operating model).

These data frames differ from the previously discussed slots as they are 4-dimensional arrays, with dimensions corresponding to the simulation, the age classes, the historical year, and the spatial areas.

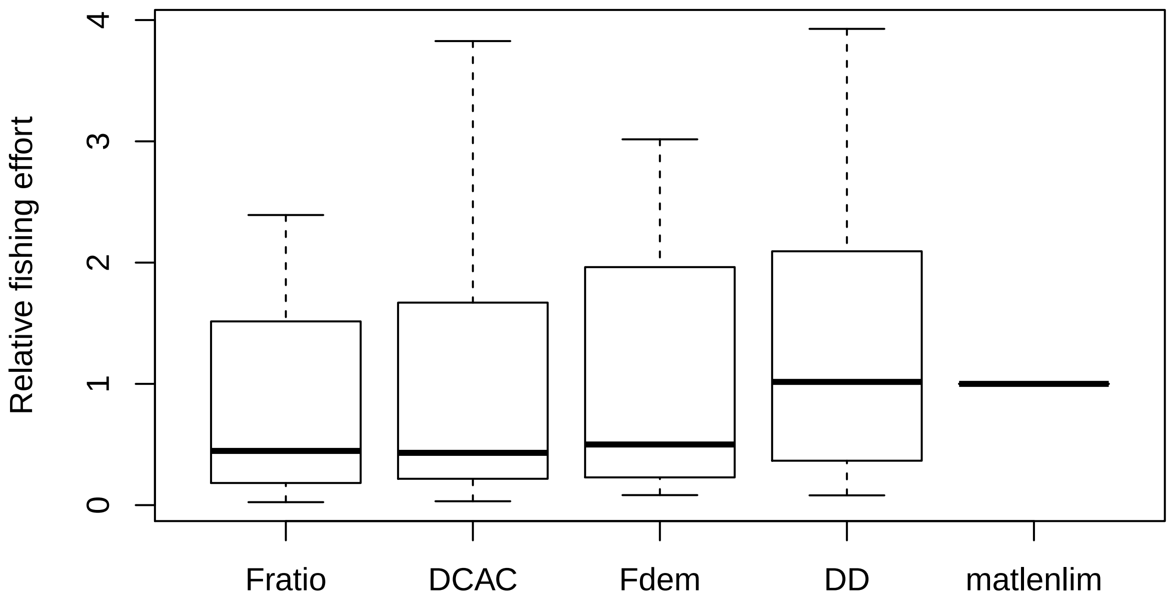
The `apply` function can be used to aggregate these data over the age-classes or spatial areas.

7.1.8 The `Effort` Slot

The `Effort` slot is a 3-dimensional array containing information on the relative fishing effort (relative to last historical year, or current conditions) for each simulation, Management Procedure and projection year.

We can look at the distribution of fishing effort for each Management Procedure in the final year of the projection period:

```r
pyear <- BSharkMSE@proyears
boxplot(BSharkMSE@Effort[,, pyear], outline=FALSE, names=BSharkMSE@MPs, ylab="Relative fishing effort")
```



This plot shows that the median fishing effort in the final year ranges from 0.43 to 1.02 for the first four output control methods, and is constant for the input control method (`matlenlim`).

This is because the output control method adjusts the total allowable catch, which depending on the amount of available stock, also impacts the amount of fishing activity.

The input control methods assume that fishing effort is held at constant levels in the future, although the catchability is able to randomly or systematically vary between years. Furthermore, input control methods can also adjust the amount of fishing effort in each year.

7.2 Performance Metrics

A key use of the DLMtool is to evaluate the trade-offs in the performance of different potential Management Procedures and to assist in the decision-making process as to which Management Procedure is most likely to satisfy the various management objectives under realistic range of uncertainty and variability in the system.

7.2.1 The Need for Performance Metrics

In order to evaluate the relative effectiveness of different Management Procedures, it is important that decision-makers have clearly-defined management objectives. These management objectives can be incorporated into the MSE process in the form of performance metrics, which provide the yardstick with which to compare the relative performance of different management strategies.

Fisheries managers are confronted with the difficult task of maximizing yield and ensuring the sustainability of the resource and the overall health of the marine environment. The principal objectives of fisheries management could be described as ensuring sustainable harvests and viable fishing communities, while maintaining healthy ecosystems. However, this simplistic view overlooks the fact that there are often conflicts in different management objectives and that there is rarely an optimal management approach that fully satisfies all management objectives (Punt, 2015). Walters and Martell (2004) explain that the task of modern fisheries management is to identify the various trade-offs among conflicting objectives and decide how to balance them in a satisfactory way.

7.2.2 Inevitable Trade-Offs

A typical trade-off is the abundance of the target species versus the catch. Assuming no significant system-wide natural perturbations, a fish stock may be exploited sustainability if catches are set at low levels. However, such economic under-utilization of the resource is often seen as undesirable. Alternatively, high catches may produce immediate short-term benefits, but may result in long-term degradation, or perhaps collapse, of the stock.

Additionally, there is often a trade-off between stock size and fishing effort, which results in lower catch rates (and lower profit) for individual fishers when a large number of fishers are active in the fishery (Walters and Martell, 2004). Other common trade-offs include the age and size at first capture, either delaying harvest until individuals are fewer in number (due to natural mortality) but larger in size, or capturing a large number of small sized fish (Punt, 2015).

When multiple objectives are considered, there is usually not a single optimum solution, and fisheries managers are faced with the difficult task of determining the most appropriate management action that satisfies the numerous management objectives and stakeholder interests (Punt, 2015).

7.2.3 Operational Management Objectives

A key strength of the MSE approach is that decision-makers are required to specify clear objectives, which can be classified as either “conceptual” or “operational” (Punt et al., 2014). Conceptual objectives are typically high-level policy goals that may be broadly defined.

However, in order to be included in an MSE, conceptual objectives must be translated into operational objectives (i.e., expressed as values for performance metrics). Such operational objectives, or performance metrics, may consist of both a reference point (e.g., biomass some fraction of equilibrium unfished level) as well as a measure of the acceptable associated risk (e.g., less than 10% chance that biomass declines below this reference level).

It is not unusual that some of the management objectives are in conflict. A key benefit of the MSE approach is to highlight these trade-offs among the different management objectives to guide the decision-making process. However, in order for these trade-offs to be quantified, it is critically important that the performance metrics are quantifiable and thus able to be incorporated into the MSE framework (Punt, 2015).

7.2.4 Performance Metrics in the DLMtool

Management strategy evaluation is a simulation exercise where the model can track the specific performance with perfect information, so it is possible to state performance objectives in specific terms that are consistent with the typical objectives of fisheries policies, such as:

- Biomass relative to unfished biomass (B_0) or biomass at maximum sustainable yield (B_{MSY}).
- Fishing mortality rate relative to fishing at maximum sustainable yield (F_{MSY}).
- Yield (short-term or long-term) of a particular management strategy relative to the yield if the fishery were being exploited at F_{MSY}.
- Inter-annual variability in yield or effort (e.g., fluctuations in yield from year to year).

Because the management strategy evaluation runs many simulations of the fisheries performance under each management strategy being tested, the performance can be stated probabilistically, such as the specific probability of biomass being above or below a specific biomass threshold or target.

7.2.4.1 Fishing Mortality

For example, the management strategies can be ranked by the likelihood of overfishing to occur, where the probability of overfishing is measured by the proportion of simulation runs where the fishing mortality rate (F) under a specific management strategy is higher than the F that is expected to produce the maximum sustainable yield.

Management strategies that have a lower probability of overfishing occurring are typically preferable to those that frequently cause excessive fishing mortality rates. If there are 1,000 simulation runs for each management strategy over a 50-year projection period, then the probability of overfishing could be based on the proportion where F is greater than (or less than) F_{MSY} over all years or any subset of years (e.g., probability of overfishing in years 41-50 of the 50-year projection period).

7.2.4.2 Stock Biomass

Another performance metric included in DLMtool is the probability that the stock biomass is above or below some biological reference point. For example, a minimum performance limit may be half the biomass at maximum sustainable yield (0.5 BMSY), and the performance of the management strategies can be ranked by the probability of the stock remaining above this level.

Management strategies that fail to maintain biomass above this limit with a high priority may be considered too risky and therefore excluded from further examination.

7.2.5 Developing Additional Performance Metrics

There may be other performance metrics that are of interest to fishery managers and stakeholders. Stakeholder participation is critical when developing performance metrics to evaluate different biological scenarios or management strategies in a MSE. Furthermore, it is important that the performance metrics, together with any acceptable risk thresholds are identified and agreed upon before the MSE is conducted.

The DLMtool can be customized to track and display additional performance metrics as identified by stakeholders.

7.2.6 Summarizing Management Procedure Performance

The information in the `MSE` object can be summarized in a number of ways.

The `summary` function provides information on the performance of the Management Procedures with respect to various metrics, including the probability of overfishing, and the probability that the biomass is below various reference levels:

```
summary(BSharkMSE)
```

```
##          MP Yield stdev   POF stdev    P10 stdev   P50 stdev  P100 stdev
## 1    Fratio 54.83 41.00 48.46  43.67 14.04 29.44 31.00 39.29 53.00 43.06
## 2      DCAC 67.19 48.33 26.79  37.56 10.79 26.00 15.62 30.84 30.08 36.47
## 3      Fdem 54.55 34.41 52.75  40.94 20.17 34.81 35.08 42.65 56.62 39.67
## 4        DD 68.88 42.15 58.88  36.19 13.96 22.70 35.92 35.54 66.38 33.26
## 5 matlenlim 86.04 37.96 55.21  40.94  0.04  0.29 12.50 28.50 32.92 36.74
##    LTY  STY   VY
## 1 57.9 59.4 33.3
## 2 65.2 53.5 91.7
## 3 58.3 64.0 41.7
## 4 69.4 75.8 70.8
## 5 84.4 72.3  2.1
```

This information can be used to identify poorly performing methods, and exclude them from further, perhaps more comprehensive, runs of the MSE.

7.3 Plotting the MSE Results

The DLMtool has 18 plotting functions which can be used to examine the performance of the Management Procedures.

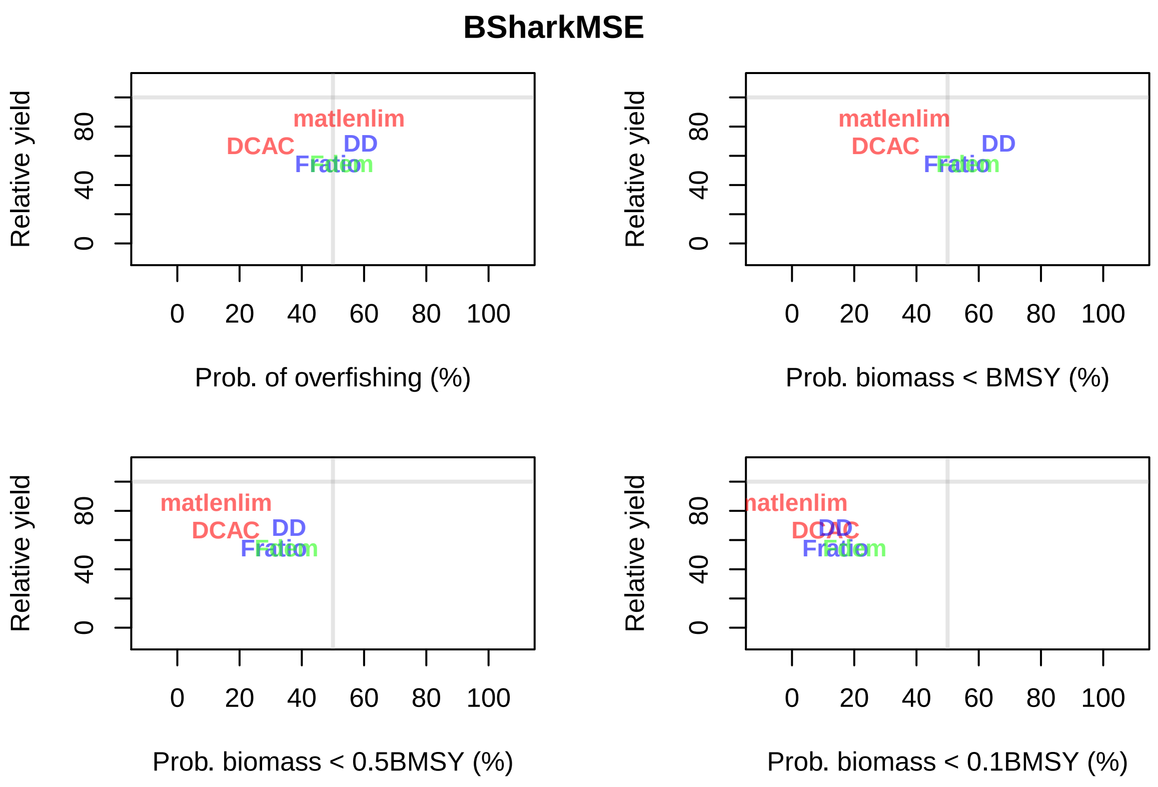
7.3.1 Trade-off Plots

One of the most common use of a MSE is to examine the trade-offs in the performance of alternative management procedures.

The DLMtool has a number of functions to examine these trade-offs, and users of the Toolkit can easily develop their own.

The `Tplot` function creates four plots that show the trade-off between the expected relative yield and the probability of overfishing and the probability of the biomass being below three different reference points:

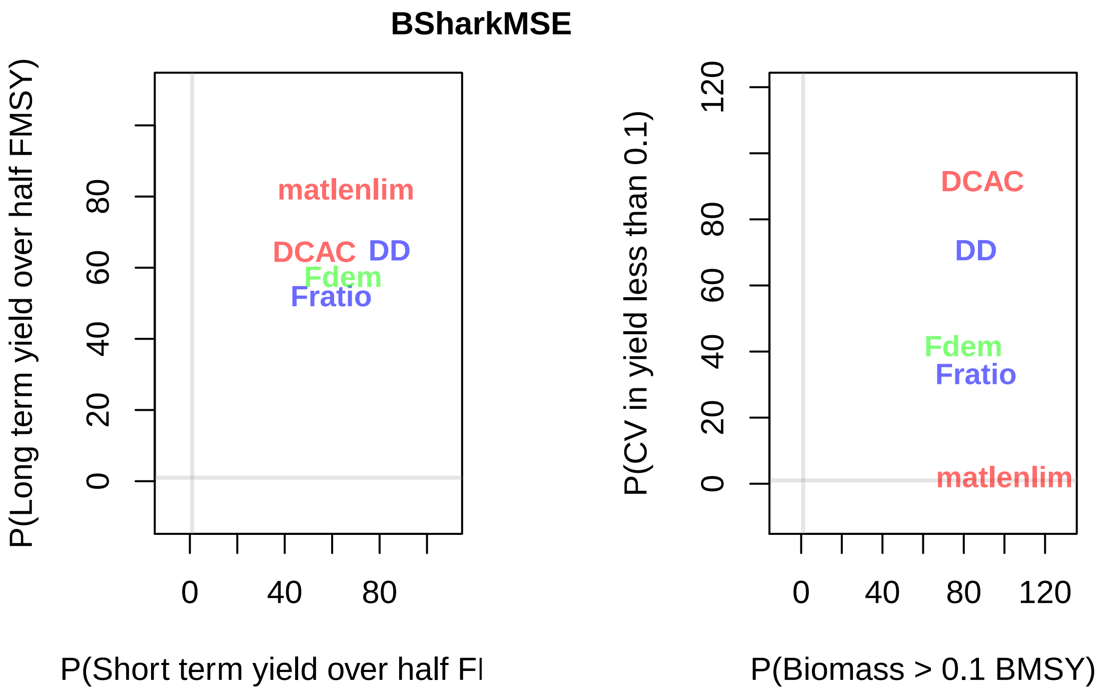
```
Tplot(BSharkMSE)
```



In this case, the plot shows that the `matlenlim` method results in the highest long-term yield, and also one of the lower probabilities that the biomass will fall below $0.5B_{MSY}$.

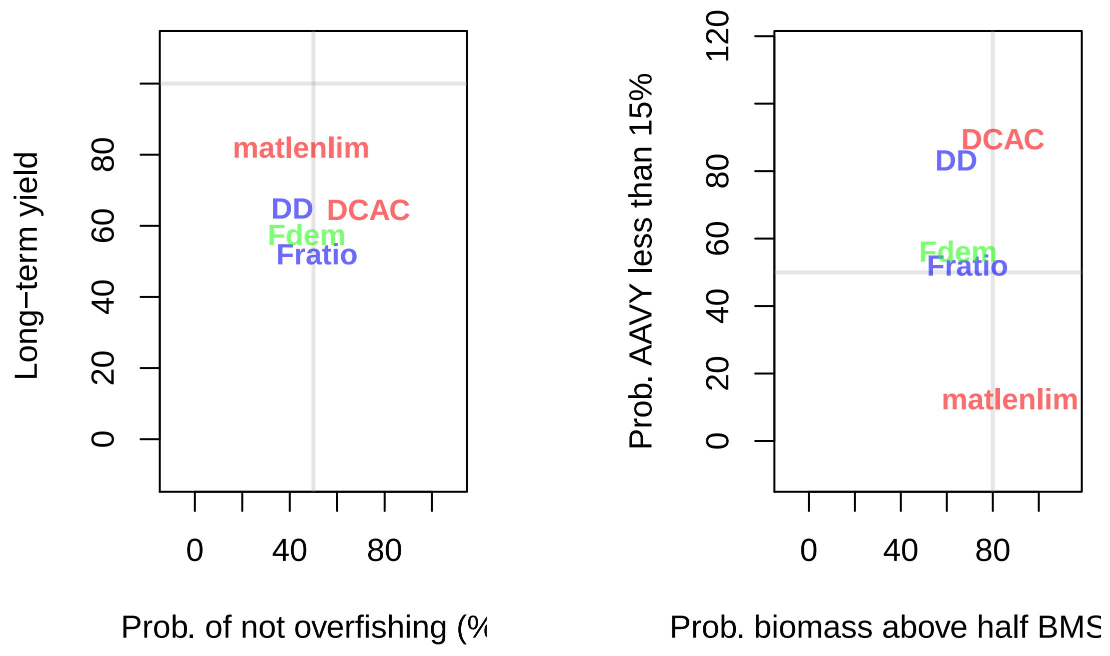
The `Tplot2` function shows the trade-off between long-term and short-term yield, and the trade-off between biomass being above $0.1B_{MSY}$ and the expected variability in the yield:

```r
Tplot2(BSharkMSE)
```



The `NOAA_plot` function was developed from applications of the DLMtool to fisheries in the Caribbean. This plot shows the trade-offs between the probability of not overfishing and long-term yield, and the probability of not being in an overfished state versus the probability of the annual variation in yield being less than 15%:

```r
NOAA_plot(BSharkMSE)
```



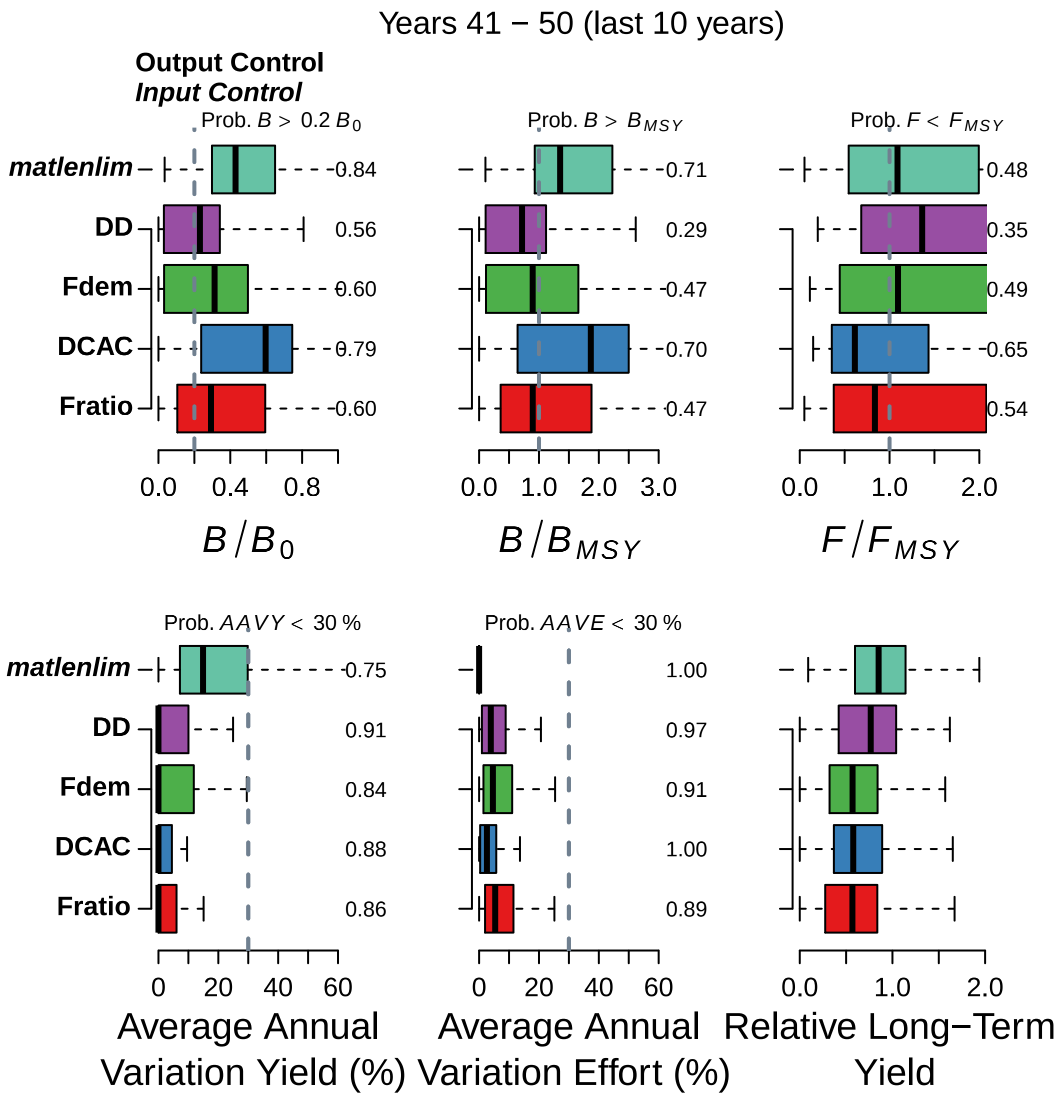
```
##           PNOF  B50  LTY   VY
## Fratio    51.5 69.0 52.1 52.1
## DCAC      73.2 84.4 64.6 89.6
## Fdem      47.2 64.9 57.5 56.2
## DD        41.1 64.1 65.0 83.3
## matlenlim 44.8 87.5 82.1 12.5
```

7.3.2 Boxplot

The distribution of various statistics can be examined for the Management Procedures using the `boxplot` function:

```
boxplot(BSharkMSE)
```

```
## Calculating MP Performance for last 10 years
```



`boxplot` is a generic function, which means that its behavior depends on the class of object that is supplied to it. In this case, R recognizes that we pass an object of class `MSE` to the `boxplot` function, and calls the appropriate plotting function `boxplot.MSE`.

The `boxplot.MSE` function has a number of arguments which allow users to control various aspects of the plot. You can display the arguments by using the `args` function (`args(boxplot.MSE)`) or looking at the help documentation (`?boxplot.MSE`).

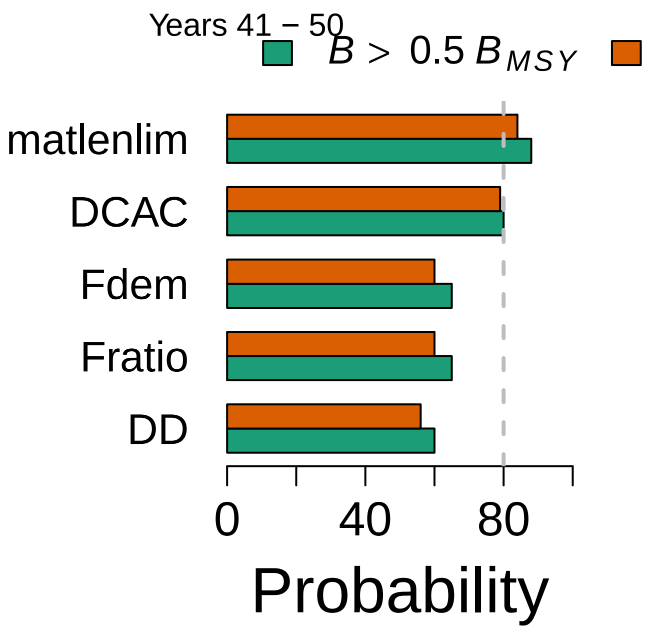
7.3.3 Barplot

The `barplot` function is another generic function, which calls `barplot.MSE` when it is supplied with an object of class `MSE`.

The `barplot` function shows the probability of each Management Procedure meeting the specified performance criteria:

```
out <- barplot(BSharkMSE)
```

```
## Calculating MP Performance for last 10 years
```



Many aspects of the plot can be controlled using the arguments to the `barplot.MSE` function, including the number of years over which to calculate the probabilities, as well control the performance metrics that shown in the plot.

The `barplot` function returns invisible output, which we have captured here by assigning the output of the function to the variable `out`. We can display this output by typing `out` into the R console:

```
out
```

```
##          MP B_BMSYp SSB_SSB0p  Pass MPClass
## 1        DD      60        56 FALSE  Output
## 2    Fratio      65        60 FALSE  Output
## 3      Fdem      65        60 FALSE  Output
## 4      DCAC      80        79 FALSE  Output
## 5 matlenlim      88        84  TRUE   Input
```

This data frame displays the probability with respect to the different performance metrics, as well as whether the Management Procedure passed or failed the specified level of acceptable risk.

In this can, we can see that the `matlenlim` method is the only method that meets the requirement of at least 80% probability that the biomass in the last 10 years of the projection period is above $0.5B_{MSY}$ and above $0.2B_0$.

7.3.4 Joint Probability Plot

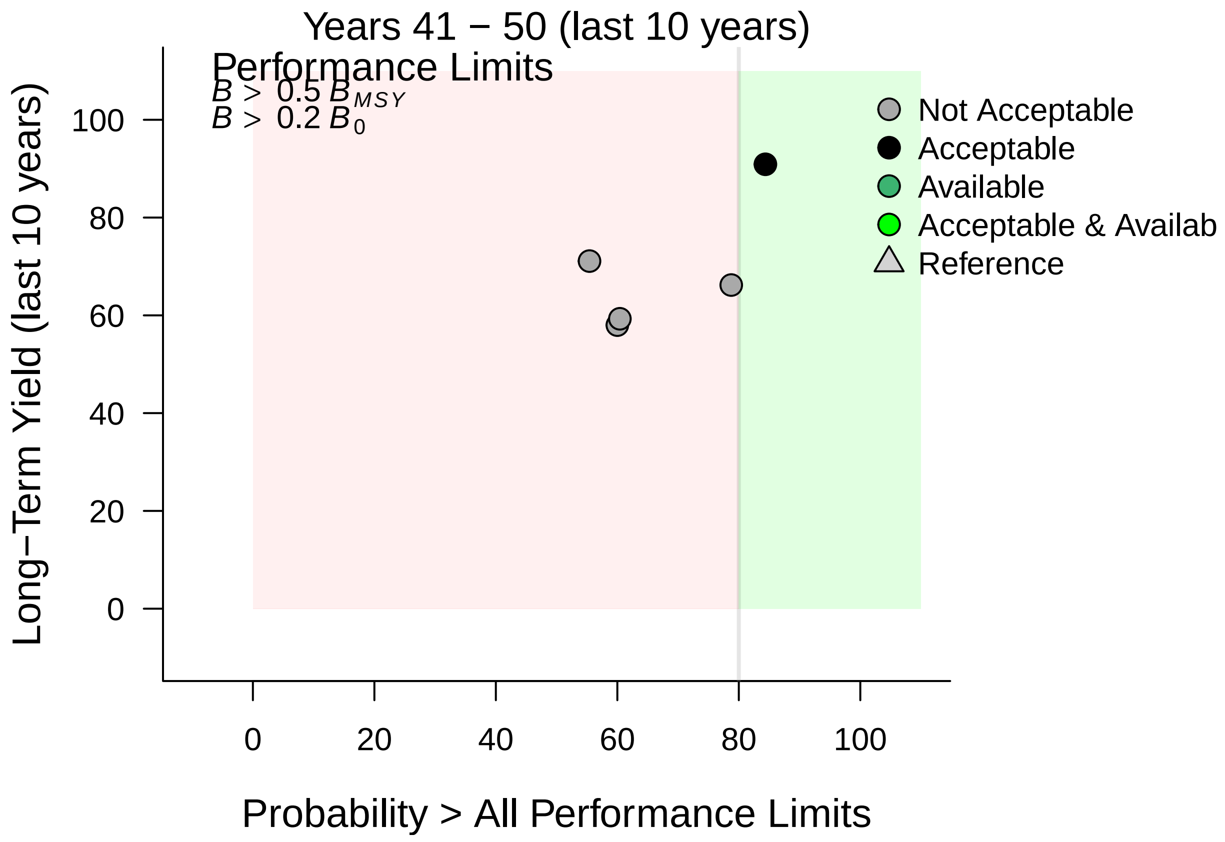
The previous plots calculate the probability of that Management Procedure will meet individual performance criteria.

An alternative is to calculate the probability that a Management Procedure will simultaneously meet all of the performance criteria. The `Jplot` function has been designed to calculate and display the joint probability of meeting multiple criteria.

For example, the plot below calculates the probability that the biomass in the last 10 years of the projection period is above $0.5B_{MSY}$ and $0.2B_0$ for each of the 5 Management Procedures included in the MSE:

```
Jplot(BSharkMSE)
```

```
## Calculating MP Performance for last 10 years
```

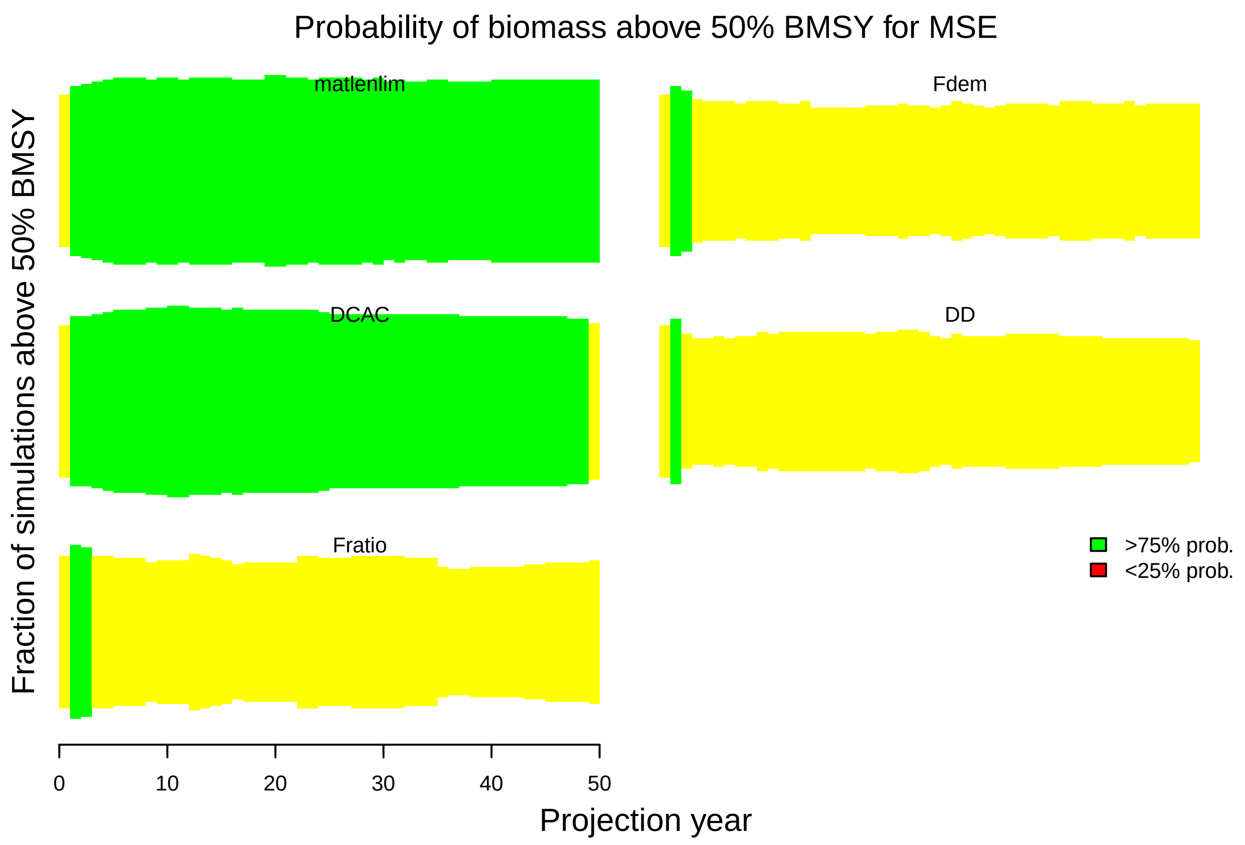


The risk threshold and the performance criteria can be adjusted in the arguments to the `Jplot` function.

7.3.5 Wormplot

The `wormplot` function plots the likelihood of meeting biomass targets in future years:

```
wormplot(BSharkMSE)
```



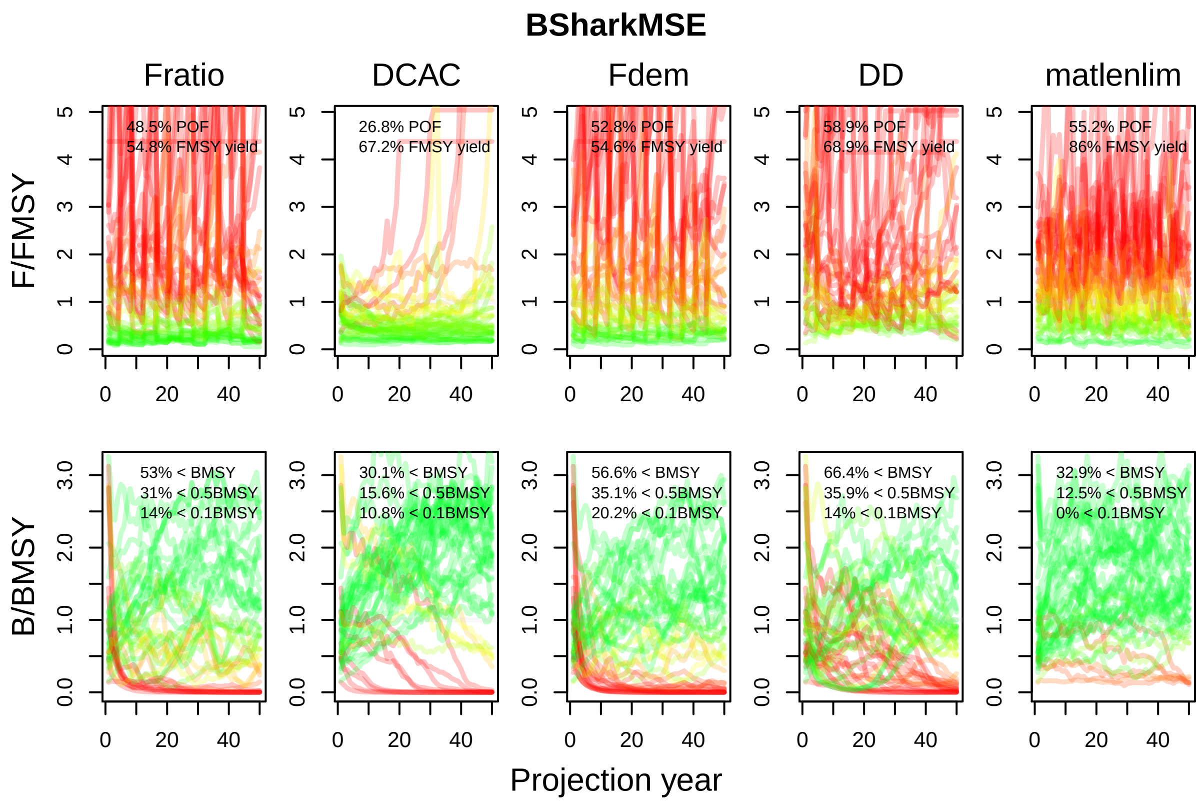
The arguments to the `wormplot` function allow you to choose the reference level for the biomass relative to B_{MSY}, as well as the upper and lower bounds of the colored bands.

7.3.6 Projection Plots

The `Pplot` function plots the trajectories of biomass, fishing mortality, and relative yield for the Management Procedures.

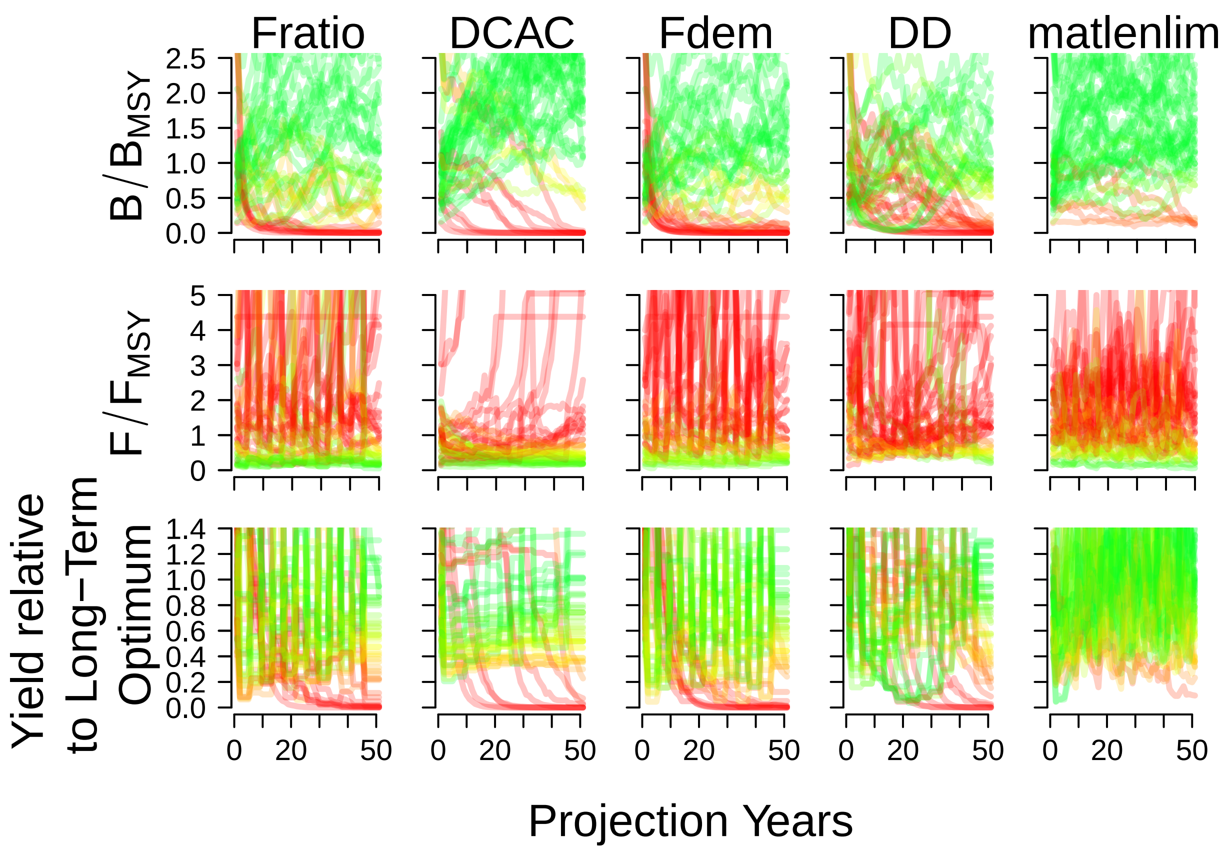
By default, the `Pplot` function shows the individual trajectories of B/B_{MSY} and F/F_{MSY} for each simulation:

```
Pplot(BSharkMSE)
```



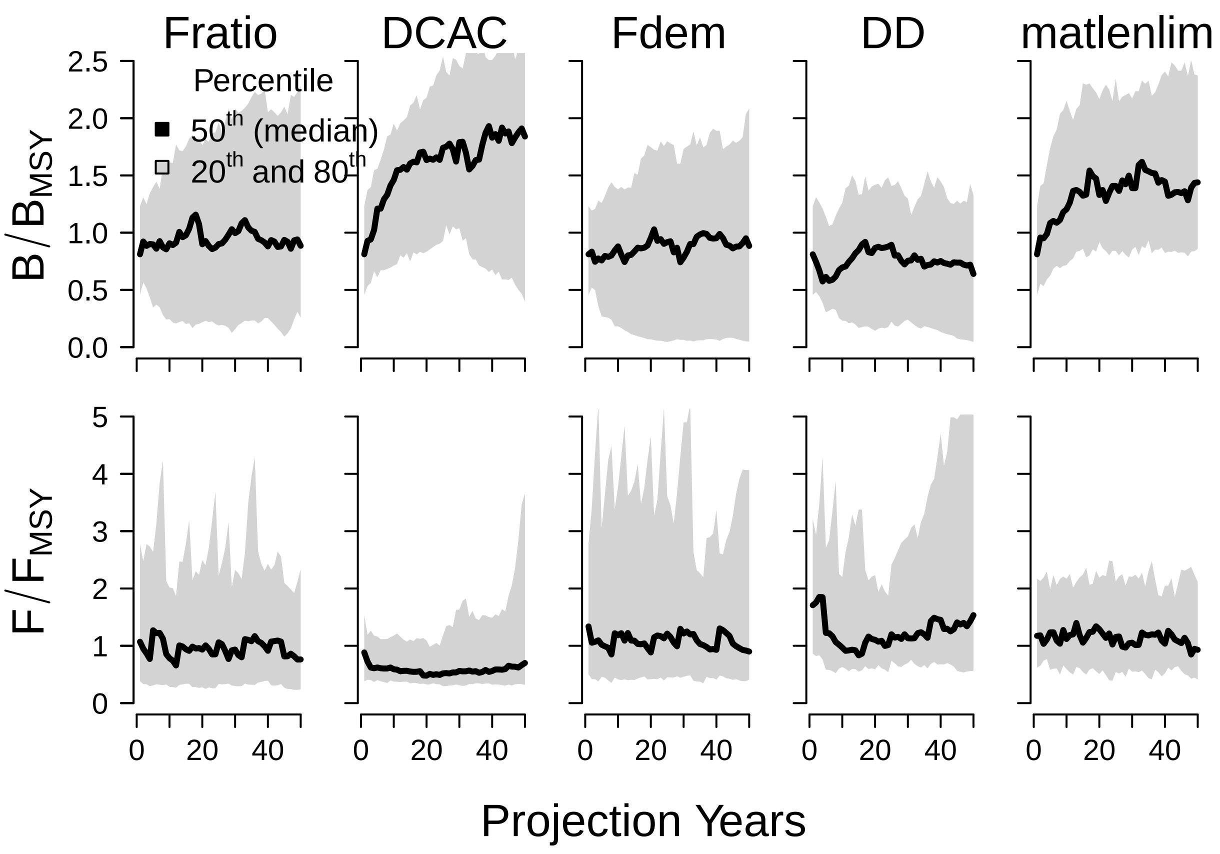
The Pplot2 function has several additional arguments. The YVar argument can be used to specify additional variables of interest. For example, here we have included the projections of yield relative to the long-term optimum yield:

```
Pplot2(BSharkMSE, YVar=c("B_BMSY", "F_FMSY", "Yield"))
```



The `traj` argument can be used to summarize the projections into quarantines. Here we show the 20th and 80th percentiles of the distributions (the median (50th percentile) is included by default):

```
Pplot2(BSharkMSE, traj="quant", quants=c(0.2, 0.8))
```

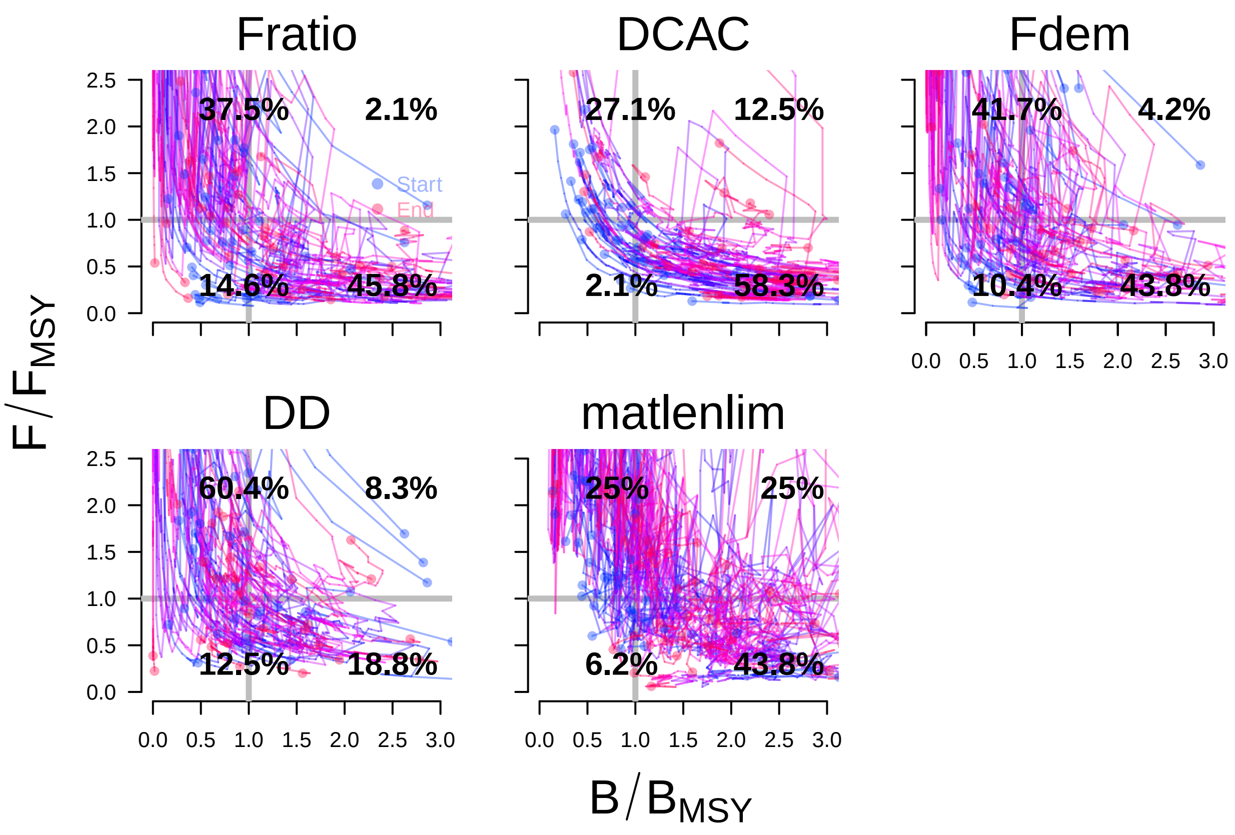


Details on additional controls for the `Pplot` and `Pplot2` functions can be found in the help documentation associated with this function.

7.3.7 Kobe Plots

Kobe plots are often used in stock assessment and MSE to examine the proportion of time the stock spends in different states. A Kobe plot of the MSE results can be produced with the `Kplot` function:

```
Kplot(BSharkMSE)
```



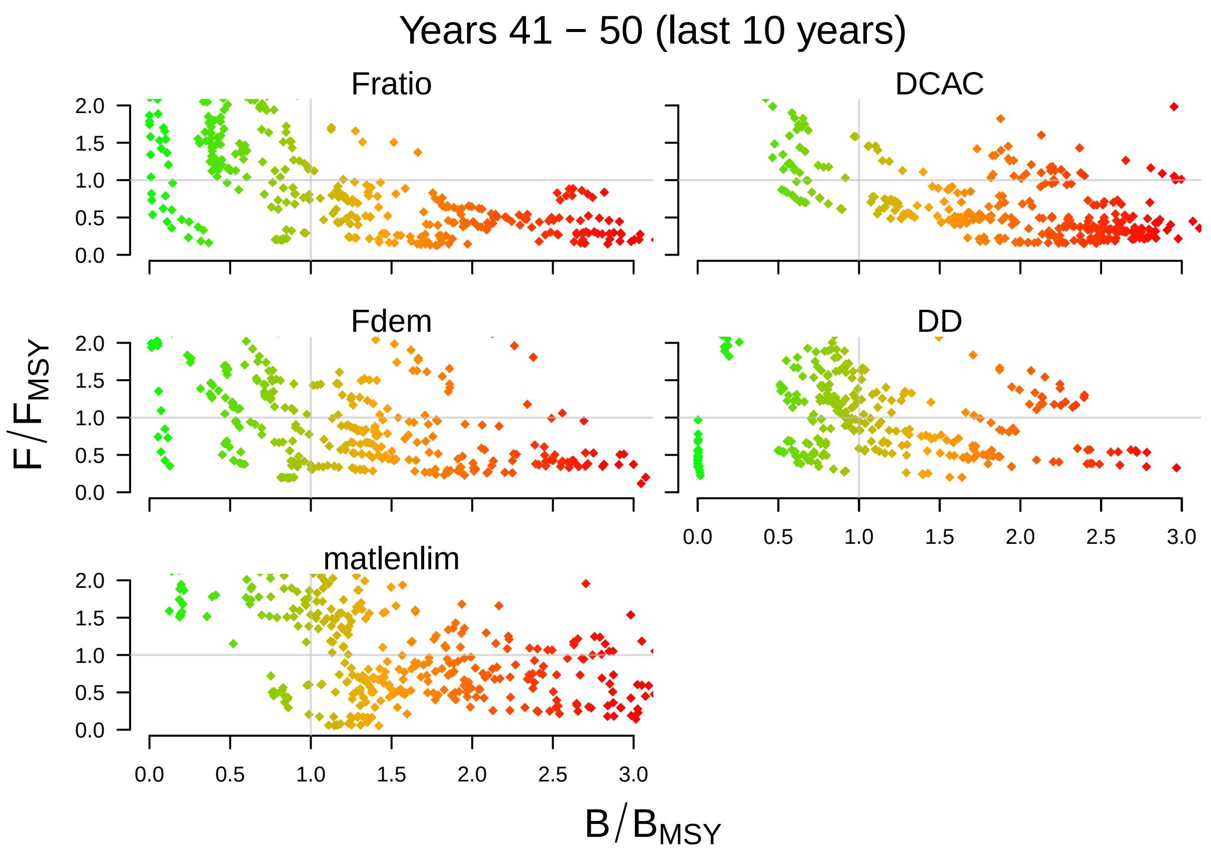
Because of the way it uses transparent colors to show the overlapping lines, the `Kplot` function can take a long time to render, especially on machines with a Windows OS.

7.3.8 Scatter Plots

The `Splot` function can be used to create a simulation-by-simulation scatter plot of the relative fishing mortality and stock biomass for each Management Procedure:

```
Splot(BSharkMSE)
```

```
## Calculating MP Performance for last 10 years
```

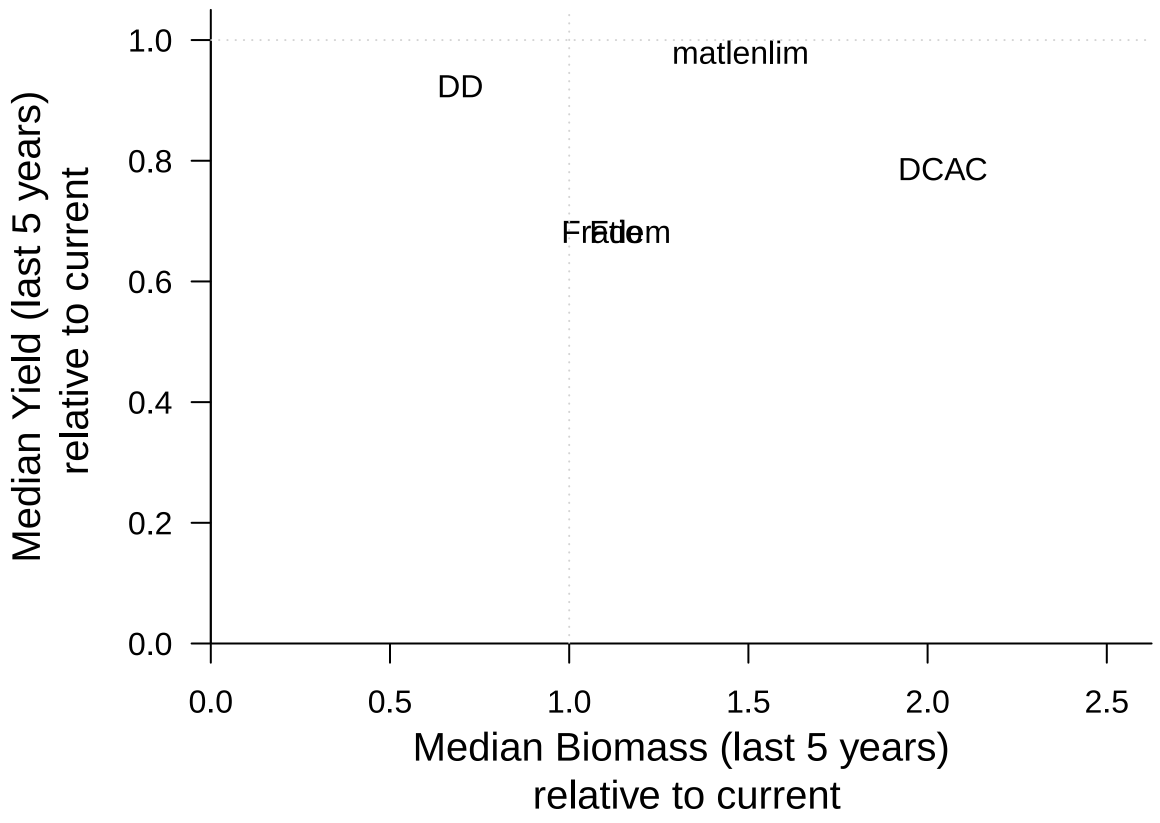


7.3.9 Compare to Current Conditions

The `Cplot` shows a scatter plot of the median biomass and median yield over the last five years of the projection relative to the current conditions (the last year in the historical period):

```
Cplot(BSharkMSE, ShowLabs=TRUE)
```

```
## Calculating MP Performance for last 5 years
```



In this example, the results show that catches in the future are likely to be lower than the current levels of catch regardless of which Management Procedure is used.

However, the future median biomass for the `DD` and `Fdem` methods is also lower than the current levels, which suggests that, in this case, these two methods result in both lower catches and lower biomass and are probably not the most suitable for managing this fishery.

7.3.10 List the MSE Plotting Functions

You can see a list of all the plotting functions in the DLMtool for `MSE` objects using the `plotFun` function:

```
plotFun()
```

```
## DLMtool functions for plotting objects of class MSE are:
```

```
## barplot boxplot COSEWIC_plot Cplot DFO_plot2
## Jplot Kplot NOAA_plot Pplot Pplot2
## Splot Tplot Tplot2 TradePlot VOI
## VOI2 VOIplot wormplot
```

7.4 Subsetting the MSE Object

The plotting functions demonstrated above calculate the probabilities and show the trade-offs for all the simulations in the MSE. However, sometimes it is interesting to examine the results of individual Management Procedures or simulations.

Many of the plotting functions have the optional arguments `MPs` and `sims` which allow you to specify which particular Management Procedures or simulations to include in the plots.

You can also manually subset the `MSE` object using the `Sub` function.

7.4.1 Subsetting by Performance

For example, we may wish to exclude any Management Procedures that have less than 30% probability that the biomass is below $0.5B_{MSY}$, and focus our analysis on the remaining Management Procedures.

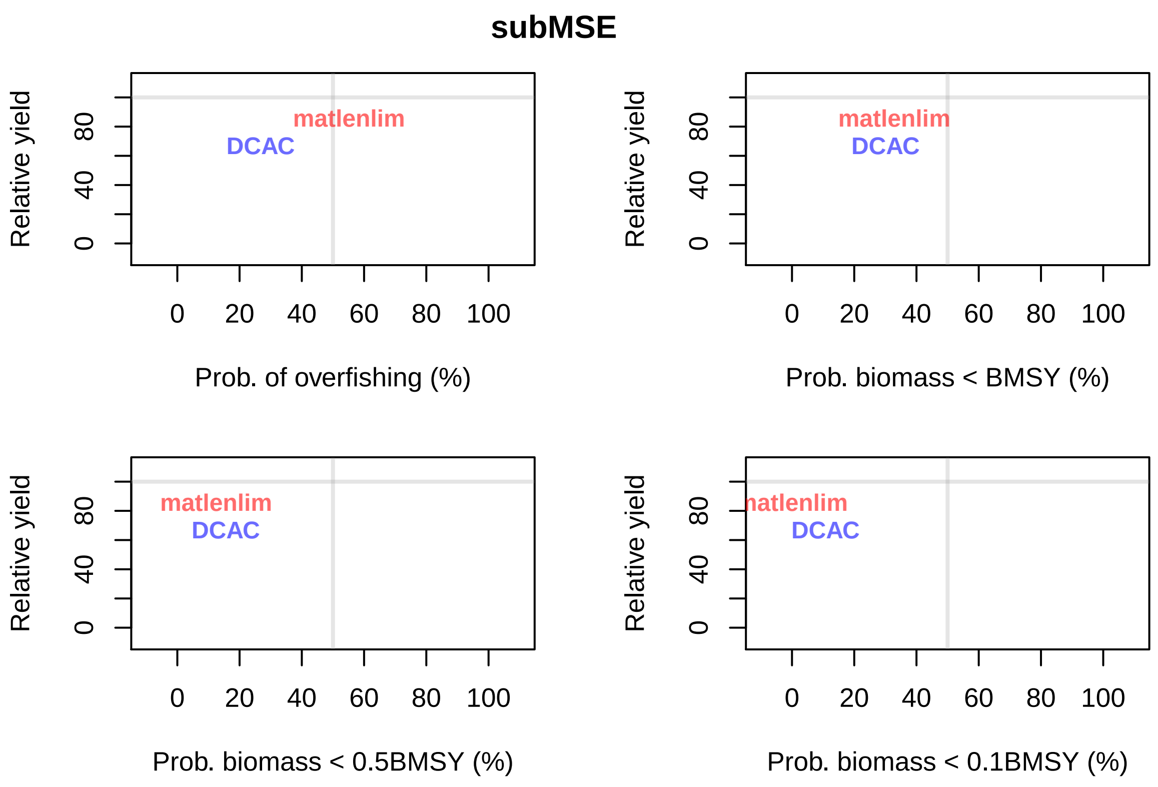
We can do this using a combination of the `summary` function and the `Sub` function:

```
stats <- summary(BSharkMSE) # save summary object to `stats`
accept <- which(stats$P50 < 30) # index of methods that pass the criteria
MPs <- stats[accept,"MP"] # the acceptable MPs

subMSE <- Sub(BSharkMSE, MPs=MPs)
```

Here we can see that the DCAC, matlenlim methods (2 of the 5) met our specified criteria. We used the `Sub` function to create a new `MSE` object that only includes these Management Procedures.

We can than proceed to continue our analysis on the `subMSE` object, e.g.:

```
Tplot(subMSE)
```



7.4.2 Subsetting by Operating Model Parameters

We can also subset the `MSE` object by simulation. For example, we may be interested to look at how the methods perform under different assumptions about the natural mortality rate (M).

In this MSE M ranged from 0.15 to 0.25. Here we identify the simulations where M was below and above the median rate:

```
below <- BSharkMSE@OM$M < median(BSharkMSE@OM$M)
above <- BSharkMSE@OM$M > median(BSharkMSE@OM$M)
```

We can then use the `Sub` function to create two MSE objects, one only including simulations with lower values of M, and the other with simulations where M was above the median value:

```
belowMSE <- Sub(BSharkMSE, sims=below)
aboveMSE <- Sub(BSharkMSE, sims=above)
```

You can see that the original MSE object has been split into two objects, each with half of the simulations:

```
belowMSE@nsim
```

```
## [1] 24
```

```
aboveMSE@nsim
```

```
## [1] 24
```

We could then continue our analysis on each subset MSE and determine if the natural mortality rate is critical in determining which Management Procedure we would choose as the best option for managing the fishery.

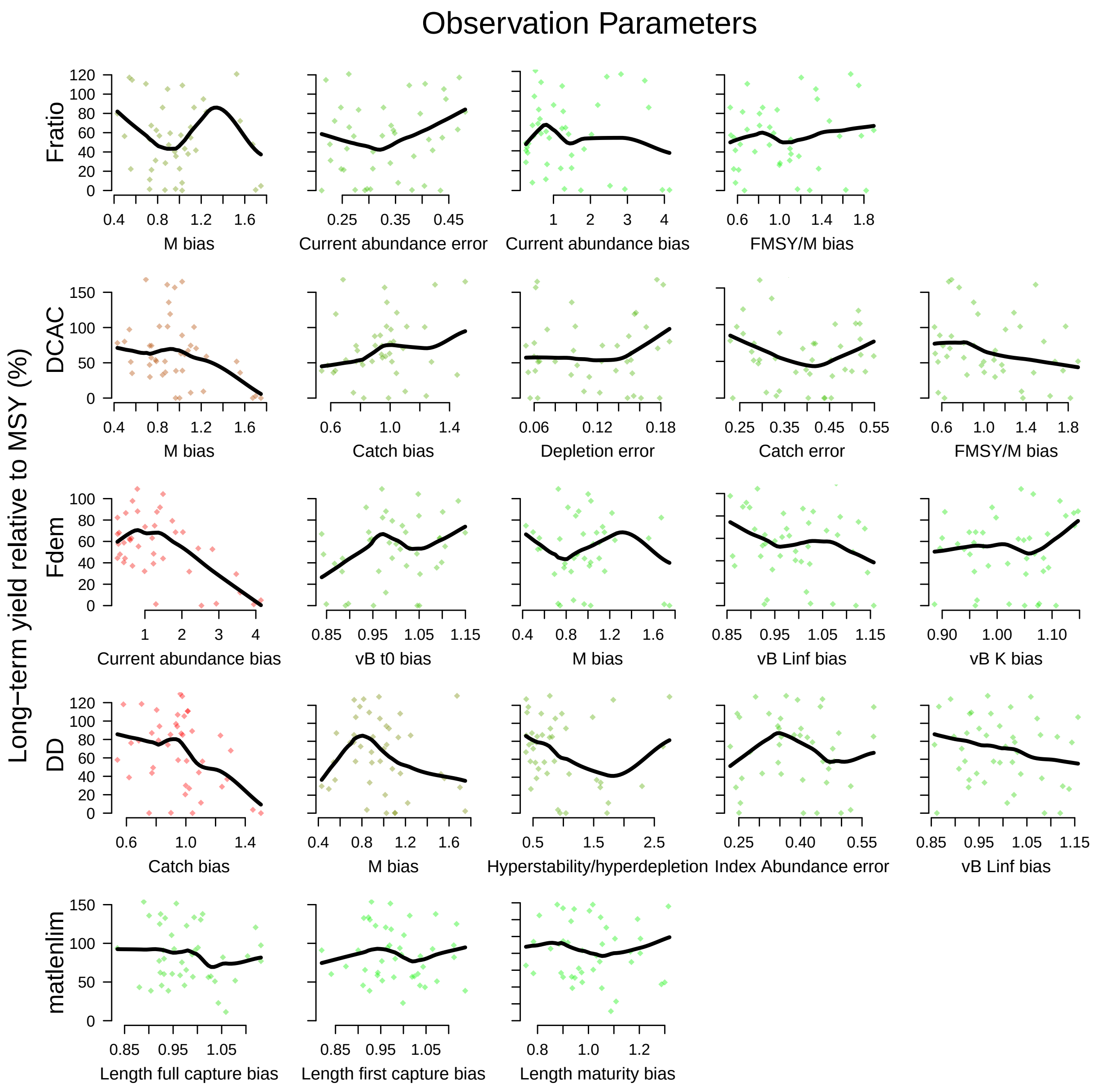
7.5 Value of Information

In the last section we looked at how the `MSE` object can subset by simulations which allows us to explore the sensitivity to various operating model parameters. However, this method only allows us the examine the sensitivity to a single parameter at a time. The Value of Information (VOI) functions have been designed to explore the sensitivity of the performance of the Management Procedures in more detail.

7.5.1 Observation Parameters

The `VOIplot` function shows how the relative long-term yield changes with respect to the Observation parameters:

```
VOIplot(BSharkMSE, nMP=5)
```



By default, the `VOIplot` function only shows the four Management Procedures with the greatest sensitivity. Here we've made it show all five methods using the `nMP` argument.

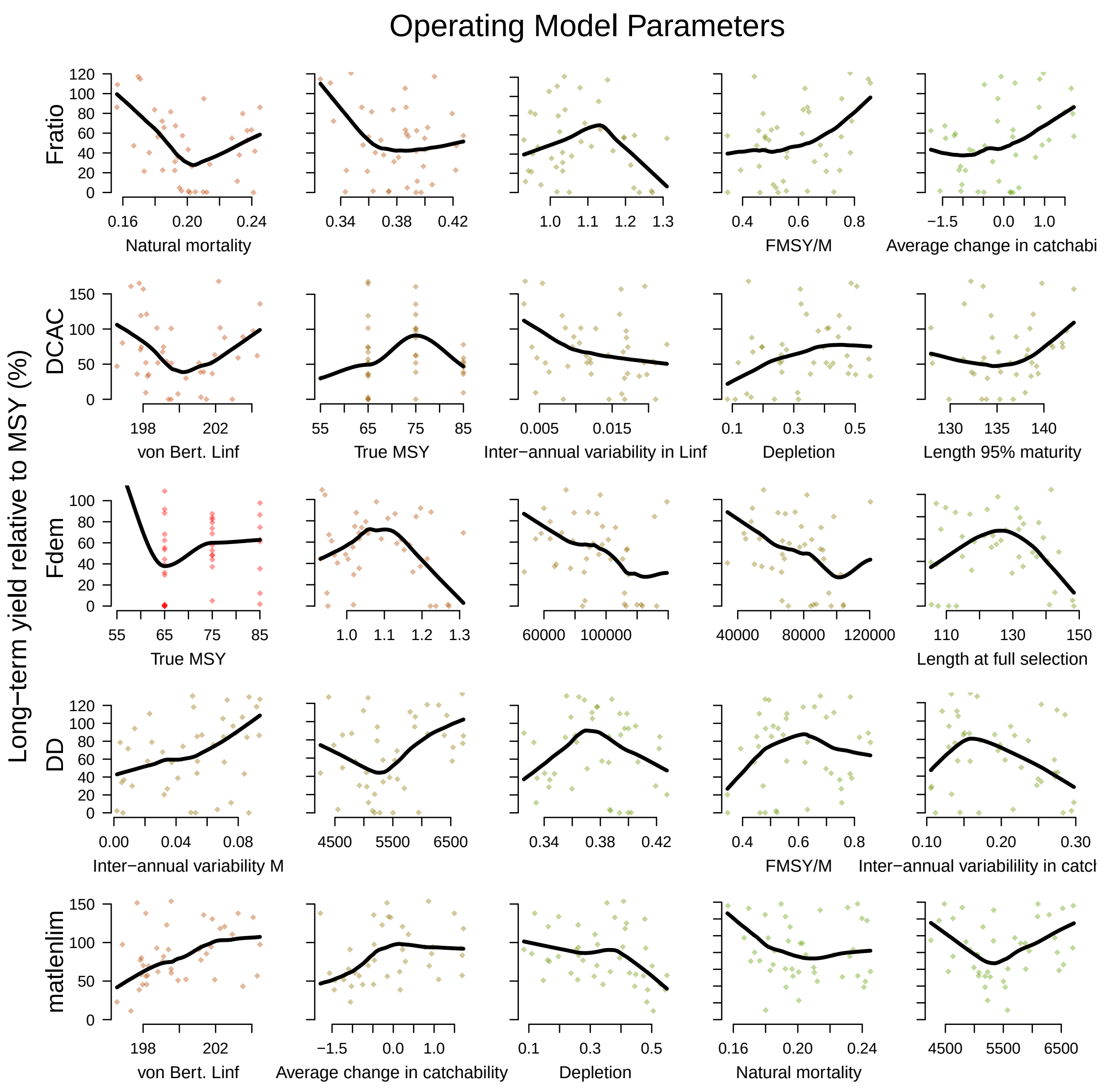
In this example, we can see that the `Fratio` method is particularly sensitive to bias in the current estimate of abundance, and over-estimates of the current abundance result in very low long-term yield (probably do to collapse of the stock). The DCAC method appears most sensitive to bias in the estimated catch.

7.5.2 Operating Model Parameters

We can also look at the sensitivity with respect to the Operating Model parameters:

```
VOIplot(BSharkMSE, Par="OM", nMP=5)
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100
```

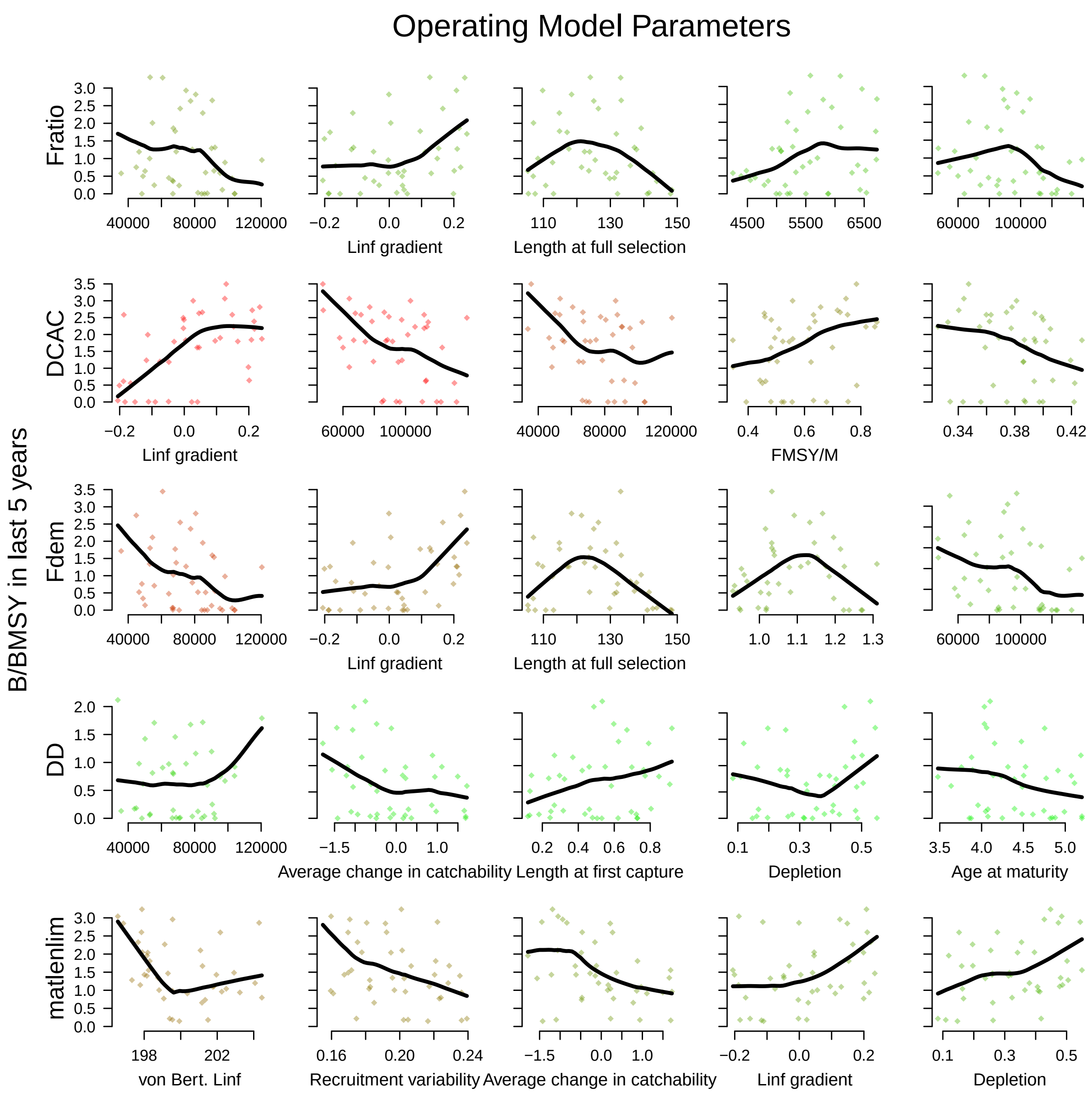


Here we can see that the `Fratio` method is most sensitive to F_{MSY}/M, with yield generally increasing for higher values of the ratio.

We can also use the `VOIplot` function to look at the sensitivity with respect to the final biomass by specifying the `YVar` argument:

```r
VOIplot(BSharkMSE, Par="OM", nMP=5, YVar="B")
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : pseudoinverse used at 65

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : neighborhood radius 10

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : reciprocal condition number 0

## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## FALSE, : There are other near singularities as well. 100
```



This result shows, perhaps unsurprisingly, that the final biomass is often strongly sensitive to the initial depletion, particularly for the DCAC and matlenlim methods.

The DLMtool also includes other value of information functions (`VOI` and `VOI2`) which present this information in alternative ways.

Chapter 8

Managing Fishery Data

8.1 The Fishery Data Object (`DLM_data`)

The second argument for all Management Procedures in the DLMtool is something called `DLM_data`.

`Data` is an object class in the DLMtool that contains all of the fishery information that can be used by the Management Procedure. You find the documentation for the `DLM_data` class by typing:

```
class?Data
```

You can see from the documentation that the `Data` object, or Fishery Data object, contains many slots, and a lot of information can be stored in this object, including biological parameters, fishery statistics such as time-series of catch, and past management recommendations.

8.1.1 In the MSE

In the MSE the Fishery Data object is populated with data that is generated by the simulation model. Here the 'true' data generated by the model is filtered through the Observation Model (using the Observation parameters) and entered into the Fishery Data object to represent typical fisheries data.

The MSE consists of many hundreds of simulations, and because the DLMtool has been designed for parallel processing, the Fishery Data object in the MSE actually consists of hundreds of 'versions' of the simulated fishery data.

The first argument for all Management Procedure functions is `x`, which is the position in the `Data` object that refers to the data corresponding that particular iteration. In the MSE, the value of `x` goes from 1 to the total number of simulations (`nsim`).

8.1.2 Application of Management Procedures Using Real Fisheries Data

In contrast to the MSE, in the real world application of a Management Procedure, we only have one version of the fishery data: the data that has been collected from the fishery.

The Fishery Data object contains all of the fishery information that can be used by a Management Procedure. By definition, many sources of data are not available in data-limited fisheries, and the Fishery Data object may not be completely populated. The DLMtool can be used to determine which of the Management Procedures in the Toolkit are available to be used given the data in the Fishery Data object, which methods cannot be used, and what data are required to make these methods available. More information on applying Management Procedures to fishery data can be found in the [Application of Management Procedure] section.

8.2 Example Data Objects

The DLMtool package has a number of example Fishery Data objects. This can be listed using the `avail` function:

```
avail("Data")
```

```
## [1] "Atlantic_mackerel"  "China_rockfish"     "Cobia"
## [4] "Example_datafile"   "Gulf_blue_tilefish" "ourReefFish"
## [7] "Red_snapper"        "Simulation_1"
```

This shows us that there is 8 example Fishery Data objects in the DLMtool.

8.3 Creating Your Own DLM_data Object

DLMtool has a series of functions to make importing data and applying data-limited Management Procedures relatively straightforward.

There are two approaches:

1. Fill out a .csv data file in excel or a text editor and use a DLMtool function to create a properly formatted `Data` object (class `Data`), or
2. Create a blank `Data` object and populate it in R.

8.3.1 Creating a CSV Data File

Probably the easiest way to get your data into the DLMtool is to populate a .csv data file. These files have a line for each slot of the `Data` object:

```
slotNames('Data')
```

```
##  [1] "Name"       "Year"       "Cat"        "Ind"        "Rec"
##  [6] "t"          "AvC"        "Dt"         "Mort"       "FMSY_M"
## [11] "BMSY_B0"    "Cref"       "Bref"       "Iref"       "L50"
## [16] "L95"        "LFC"        "LFS"        "CAA"        "Dep"
## [21] "Abun"       "SpAbun"     "vbK"        "vbLinf"     "vbt0"
## [26] "wla"        "wlb"        "steep"      "CV_Cat"     "CV_Dt"
## [31] "CV_AvC"     "CV_Ind"     "CV_Mort"    "CV_FMSY_M"  "CV_BMSY_B0"
## [36] "CV_Cref"    "CV_Bref"    "CV_Iref"    "CV_Rec"     "CV_Dep"
## [41] "CV_Abun"    "CV_vbK"     "CV_vbLinf"  "CV_vbt0"    "CV_L50"
## [46] "CV_LFC"     "CV_LFS"     "CV_wla"     "CV_wlb"     "CV_steep"
## [51] "sigmaL"     "MaxAge"     "Units"      "Ref"        "Ref_type"
## [56] "Log"        "params"     "PosMPs"     "MPs"        "OM"
## [61] "Obs"        "TAC"        "TACbias"    "Sense"      "CAL_bins"
## [66] "CAL"        "MPrec"      "MPeff"      "ML"         "Lbar"
## [71] "Lc"         "LHYear"     "Misc"
```

You do not have to enter data for every line of the data file, if data are not available simply put an 'NA' next to any given field.

8.3.1.1 Example Fishery Data CSV Files

There are also CSV files for these example Fishery Data objects that are included in the DLMtool package. To find the location where these files are located on your machine, use the `DLMDataDir` function:

```
DLMDataDir()
```

```
## [1] "C:/Program Files/R/R-3.3.3/library/DLMtool/"
```

We can then load one of the example CSV files using the `new` function:

```
China_rockfish2 <- new("Data", paste0(DLMDataDir(),"China_rockfish.csv"))
```

Alternatively, you can navigate to the data directory on your machine and examine the contents and structure of the CSV data files in MS Excel or other software.

8.3.2 Populating a `Data` Object in R

Alternatively you can create a blank `Data` object and fill the slots directly in R. For example:

```
Madeup <- new('Data')                             #  Create a blank DLM object
```

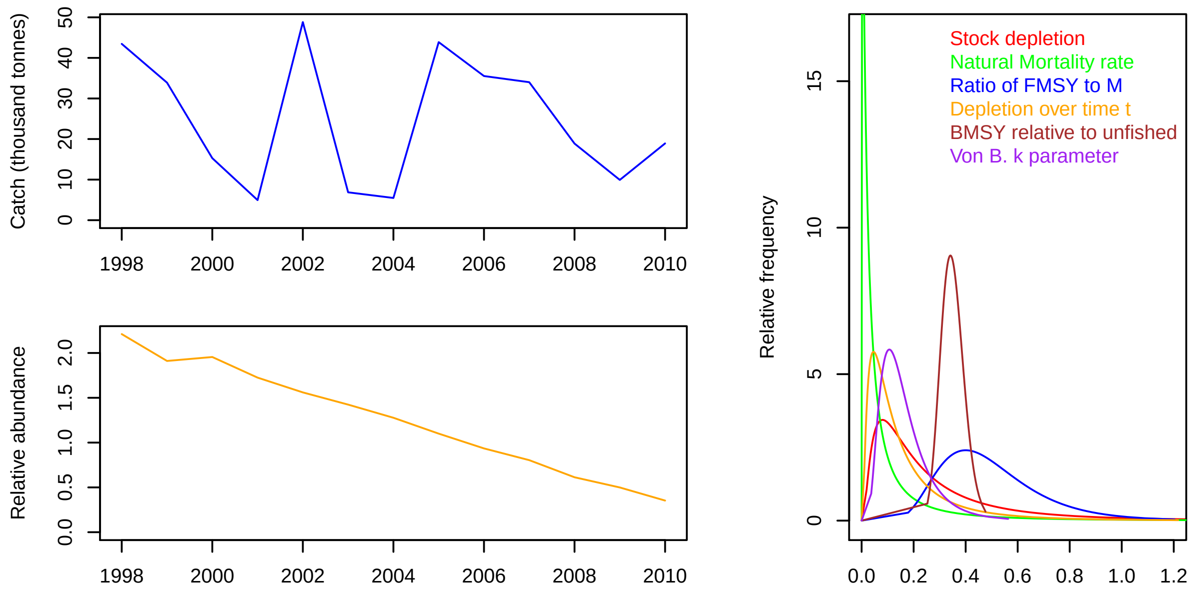
```
## [1] "Couldn't find specified csv file, blank DLM object created"
```

```
Madeup@Name <- 'Test'                               #  Name it
Madeup@Cat <- matrix(20:11*rlnorm(10,0,0.2),nrow=1)  #  Generate fake catch data
Madeup@Units <- "Million metric tonnes"             #  State units of catch
Madeup@AvC <- mean(Madeup@Cat)                      #  Average catches for time t (DCAC)
Madeup@t <- ncol(Madeup@Cat)                        #  No. yrs for Av. catch (DCAC)
Madeup@Dt <- 0.5                                    #  Depletion over time t (DCAC)
Madeup@Dep <- 0.5                                   #  Depletion relative to unfished
Madeup@vbK <- 0.2                                   #  VB maximum growth rate
Madeup@vbt0 <- (-0.5)                               #  VB theoretical age at zero length
Madeup@vbLinf <- 200                                #  VB maximum length
Madeup@Mort <- 0.1                                  #  Natural mortality rate
Madeup@Abun <- 200                                  #  Current abundance
Madeup@FMSY_M <- 0.75                               #  Ratio of FMSY/M
Madeup@L50 <- 100                                   #  Length at 50% maturity
Madeup@L95 <- 120                                   #  Length at 95% maturity
Madeup@BMSY_B0 <- 0.35                              #  BMSY relative to unfished
```

8.4 Working With `Data` Objects

A generic summary function is available to visualize the data in a `Data` object:

```
summary(Atlantic_mackerel)
```



You can see what Management Procedures can and can't be applied given your data and also what data are needed to get them working:

```
Can(Atlantic_mackerel)
```

```
##  [1] "AvC"         "BK"          "CC1"         "CC4"         "DAAC"
##  [6] "DBSRA"       "DBSRA_40"    "DBSRA4010"   "DCAC"        "DCAC_40"
## [11] "DCAC4010"    "DD"          "DD4010"      "DepF"        "DynF"
## [16] "Fadapt"      "Fdem"        "Fratio"      "Fratio4010"  "GB_slope"
## [21] "Gcontrol"    "HDAAC"       "Islope1"     "Islope4"     "Itarget1"
## [26] "Itarget4"    "MCD"         "MCD4010"     "NFref"       "Rcontrol"
## [31] "Rcontrol2"   "SBT1"        "SPmod"       "SPMSY"       "SPslope"
## [36] "SPSRA"       "YPR"         "curE"        "curE75"      "DDe"
## [41] "DDe75"       "DTe40"       "DTe50"       "EtargetLopt" "ItargetE1"
## [46] "ItargetE4"   "matlenlim"   "matlenlim2"  "minlenLopt1" "MRnoreal"
## [51] "MRreal"      "slotlim"     "area1_50"
```

```
Cant(Atlantic_mackerel)
```

```
##       [,1]          [,2]
##  [1,] "BK_CC"       "Insufficient data"
##  [2,] "BK_ML"       "Insufficient data"
##  [3,] "CompSRA"     "Insufficient data"
##  [4,] "CompSRA4010" "Insufficient data"
##  [5,] "DBSRA_ML"    "Insufficient data"
##  [6,] "DCAC_ML"     "Insufficient data"
##  [7,] "Fdem_CC"     "Insufficient data"
##  [8,] "Fdem_ML"     "Insufficient data"
##  [9,] "FMSYref"     "Insufficient data"
## [10,] "FMSYref50"   "Insufficient data"
## [11,] "FMSYref75"   "Insufficient data"
## [12,] "Fratio_CC"   "Insufficient data"
## [13,] "Fratio_ML"   "Insufficient data"
## [14,] "GB_CC"       "Produced all NA scores"
## [15,] "GB_target"   "Produced all NA scores"
## [16,] "IT10"        "Insufficient data"
## [17,] "IT5"         "Insufficient data"
## [18,] "ITM"         "Insufficient data"
## [19,] "L95target"   "Insufficient data"
## [20,] "LBSPR_ItTAC" "Produced all NA scores"
## [21,] "LstepCC1"    "Insufficient data"
## [22,] "LstepCC4"    "Insufficient data"
## [23,] "Ltarget1"    "Insufficient data"
## [24,] "Ltarget4"    "Insufficient data"
## [25,] "SBT2"        "Produced all NA scores"
## [26,] "SPSRA_ML"    "Insufficient data"
## [27,] "YPR_CC"      "Insufficient data"
## [28,] "YPR_ML"      "Insufficient data"
## [29,] "DDes"        "Insufficient data"
## [30,] "ITe10"       "Insufficient data"
## [31,] "ITe5"        "Insufficient data"
## [32,] "LBSPR_ItEff" "Insufficient data"
## [33,] "LBSPR_ItSel" "Insufficient data"
## [34,] "LstepCE1"    "Insufficient data"
## [35,] "LstepCE2"    "Insufficient data"
## [36,] "LtargetE1"   "Insufficient data"
## [37,] "LtargetE4"   "Insufficient data"
```

```
Needed(Atlantic_mackerel)
```

```
##  [1] "BK_CC: CAA"              "BK_ML: CAL"
##  [3] "CompSRA: CAA"            "CompSRA4010: CAA"
##  [5] "DBSRA_ML: CAL"           "DCAC_ML: CAL"
##  [7] "Fdem_CC: CAA"            "Fdem_ML: CAL"
##  [9] "FMSYref: OM"             "FMSYref50: OM"
## [11] "FMSYref75: OM"           "Fratio_CC: CAA"
## [13] "Fratio_ML: CAL"          "GB_CC: Cref"
## [15] "GB_target: Cref, Iref"   "IT10: Iref, MPrec"
## [17] "IT5: Iref, MPrec"        "ITM: Iref, MPrec"
## [19] "L95target: ML"           "LBSPR_ItTAC: CAL, MPrec"
## [21] "LstepCC1: MPrec, ML"     "LstepCC4: MPrec, ML"
## [23] "Ltarget1: ML"            "Ltarget4: ML"
## [25] "SBT2: Rec, Cref"         "SPSRA_ML: CAL"
## [27] "YPR_CC: CAA"             "YPR_ML: CAL"
## [29] "DDes: MPeff"             "ITe10: Iref, MPeff"
## [31] "ITe5: Iref, MPeff"       "LBSPR_ItEff: CAL, MPeff"
## [33] "LBSPR_ItSel: CAL"        "LstepCE1: MPeff, ML"
## [35] "LstepCE2: MPeff, ML"     "LtargetE1: MPeff, ML"
## [37] "LtargetE4: MPeff, ML"
```

8.5 Applying Management Procedures

8.5.1 `Input` Methods

Spatial and length-vulnerability Management Procedures (class `Input`) can be MSE tested but are often a management recommendation in themselves (e.g., setting a static size limit or closing a spatial area). Other input control methods are dynamic and respond to trends in different indicators in the data.

The `Input` function can be used to apply an input control method. For example, here we apply the `matlenlim` method to the `Atlantic_mackerel` data object:

```
Input(Atlantic_mackerel, "matlenlim")
```

```
## [1] "Checking which MPs can be run"
## [1] "Running 1 of 1 - matlenlim"
```

```
##           Effort Area 1 Area 2  SL50 SL95 UpperLimit
## matlenlim      1      1      1 90.25   95         NA
```

The resulting recommendation is a size limit around 90 cm. There is no upper slot limit specified, and the fishing effort and spatial areas open to fishing remained unchanged.

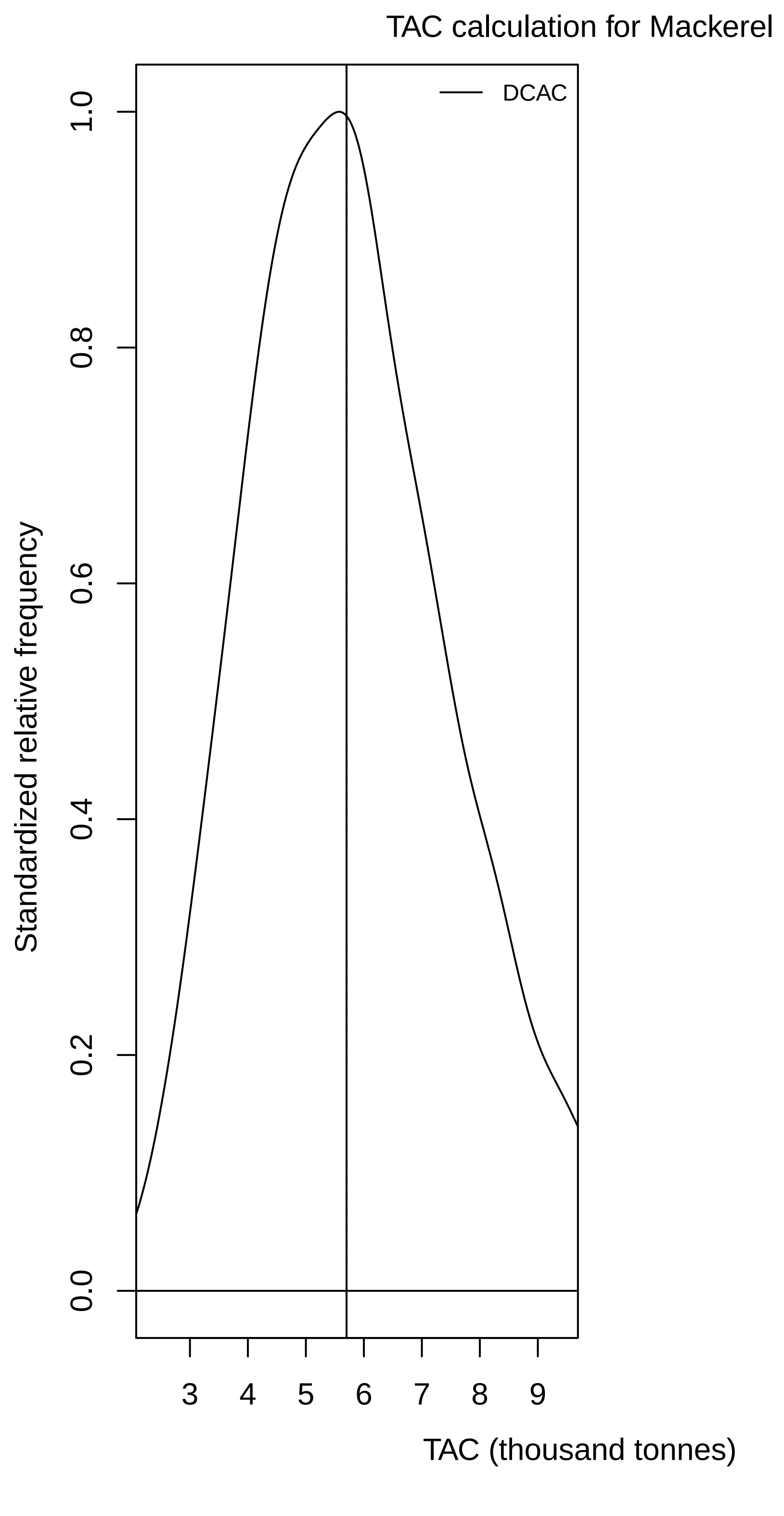
8.5.2 `Output` Methods

The `TAC` function can be used to calculate the recommended TAC for output controls Management Procedures. Here we apply the `DCAC` method, with 1,000 repetitions, to the `Atlantic_mackerel` data object:

```
Atlantic_mackerel <- TAC(Atlantic_mackerel, MPs="DCAC", reps=1000)
```

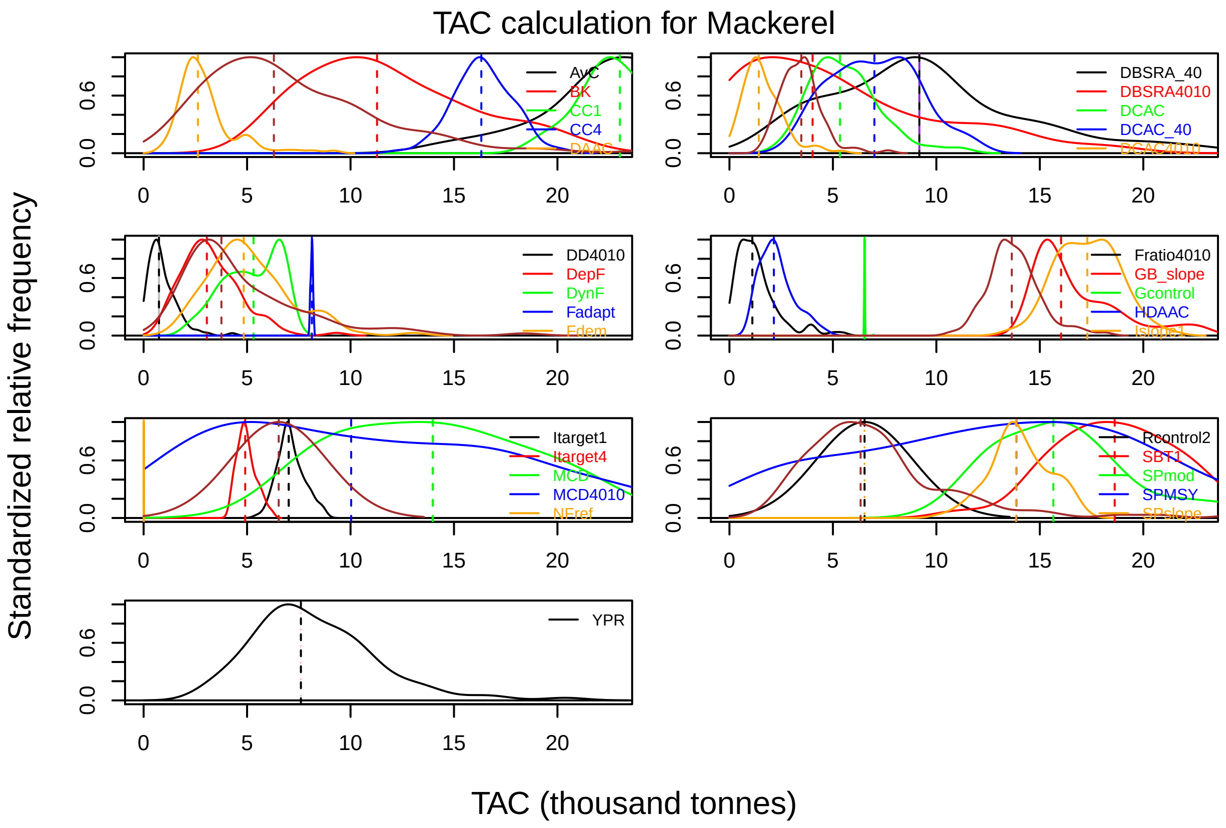
The resulting distribution of the recommended TAC can be plotted using the `plot` function:

```
plot(Atlantic_mackerel)
```



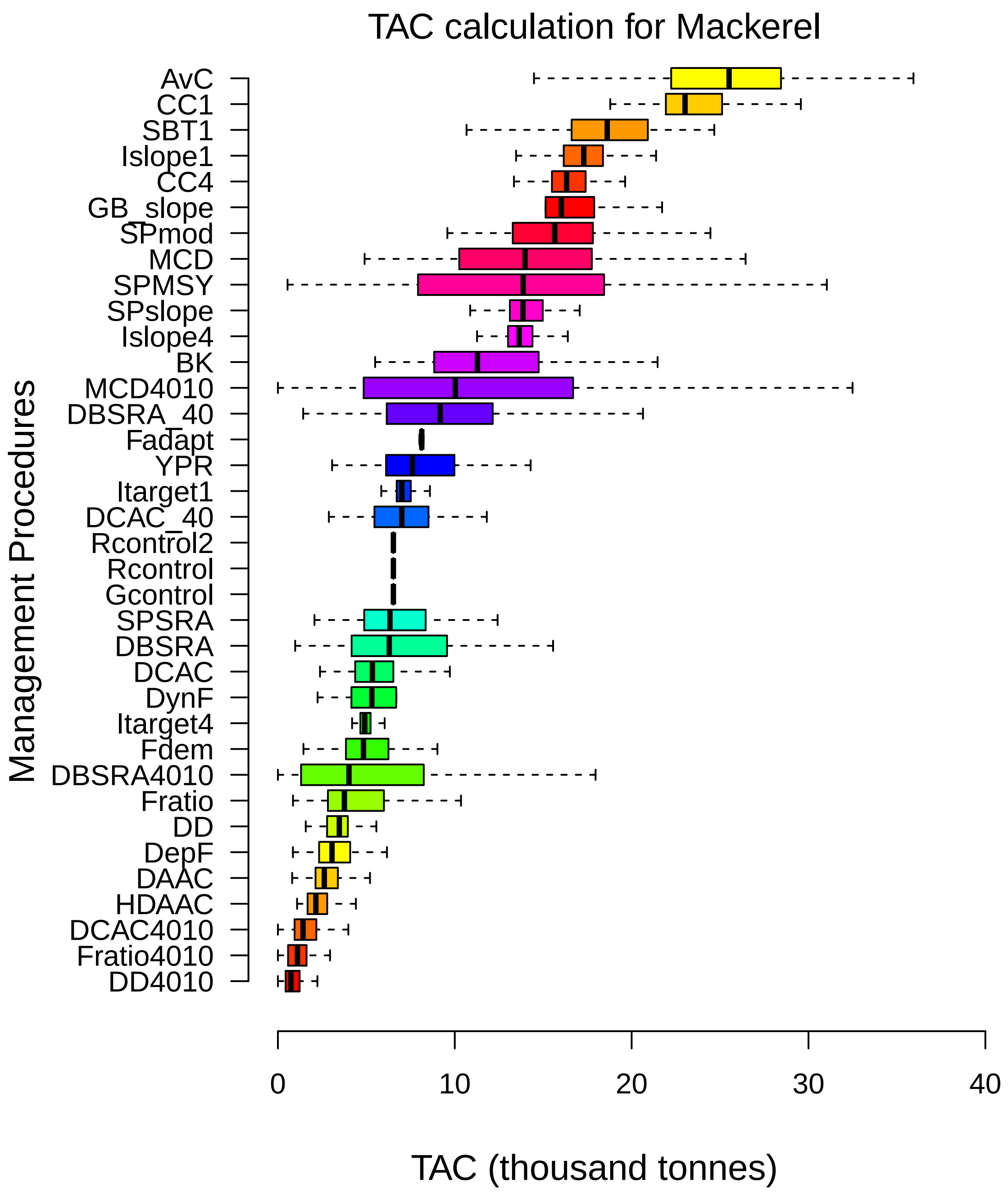
We can also apply all the out control methods that can be run with the Fishery Data object to compare the resulting recommendations:

```r
Atlantic_mackerel <- TAC(Atlantic_mackerel)
plot(Atlantic_mackerel)
```



Alternatively, we can use `boxplot`, a generic function that calls the `boxplot.DLM_data` function, to display the distribution of recommended TACs and report the statistics (median and standard deviation):

```r
boxplot(Atlantic_mackerel)
```



```
##             MP Median   SD           Units
## 1       DD4010   0.74 0.70 thousand tonnes
## 2   Fratio4010   1.11 1.03 thousand tonnes
## 3     DCAC4010   1.42 0.97 thousand tonnes
## 4        HDAAC   2.15 0.83 thousand tonnes
## 5         DAAC   2.62 1.44 thousand tonnes
## 6         DepF   3.06 1.45 thousand tonnes
## 7           DD   3.47 0.99 thousand tonnes
## 8       Fratio   3.76 3.04 thousand tonnes
## 9    DBSRA4010   4.02 4.73 thousand tonnes
## 10        Fdem   4.84 2.02 thousand tonnes
## 11   Itarget4   4.91 0.46 thousand tonnes
## 12       DynF   5.31 1.35 thousand tonnes
## 13       DCAC   5.34 1.72 thousand tonnes
## 14      DBSRA   6.30 4.69 thousand tonnes
## 15      SPSRA   6.34 4.38 thousand tonnes
## 16   Gcontrol   6.53 0.04 thousand tonnes
## 17   Rcontrol   6.53 0.00 thousand tonnes
## 18  Rcontrol2   6.53 0.00 thousand tonnes
## 19    DCAC_40   7.00 2.10 thousand tonnes
## 20   Itarget1   7.01 0.66 thousand tonnes
## 21        YPR   7.60 3.03 thousand tonnes
## 22     Fadapt   8.13 0.05 thousand tonnes
## 23   DBSRA_40   9.18 6.18 thousand tonnes
## 24    MCD4010  10.03 7.50 thousand tonnes
## 25         BK  11.28 5.16 thousand tonnes
## 26    Islope4  13.64 1.27 thousand tonnes
## 27    SPslope  13.86 1.44 thousand tonnes
## 28      SPMSY  13.87 8.40 thousand tonnes
## 29        MCD  13.98 5.87 thousand tonnes
## 30      SPmod  15.65 3.44 thousand tonnes
## 31   GB_slope  16.03 2.15 thousand tonnes
## 32        CC4  16.32 1.43 thousand tonnes
## 33    Islope1  17.29 1.52 thousand tonnes
## 34       SBT1  18.62 3.25 thousand tonnes
## 35        CC1  23.02 2.29 thousand tonnes
## 36        AvC  25.51 5.05 thousand tonnes
```

Chapter 9

Assumptions of DLMtool

Like all models, DLMtool is a simplication of reality. In order to approximate real fishery dynamics, DLMtool relies on a number of simplifying assumptions.

Some of these assumptions are common to many fishery science models (e.g., age-structured population dynamics) and are a central to the structure of DLMtool. Other assumptions are a result of the way DLMtool was designed and developed, and may represent limitations of DLMtool for applications to particular situations. It may be possible to deal with some of these assumptions by further development of DLMtool.

9.1 Biology

9.1.1 Short-Lived Species

Due to the problems with approximating fine-scale temporal dynamics with an annual model it is not advised to use the DLMtool for very short lived stocks (i.e., species with a longevity of 5 years or less).

Technically, you could just divide all temporal parameters by a subyear resolution, but the TAC would be set by sub year and the data would also be available at this fine-scale which is highly unlikely in a data-limited setting.

A MSE model with monthly or weekly time-steps for the population dynamics is required for short-lived species, and may be developed in the future.

9.1.2 Density-Dependent Compensation

DLMtool assumes there is no density-dependent compensation in the population dynamics, and fish growth, maturity, and mortality does not change directly in response to changes in stock size.

9.1.3 von Bertalanffy Growth

Growth model in DLMtool is modelled using the von Bertalanffy growth curve. While this is the most commonly applied model to describe fish growth, it may not be the preferred growth model for some species. The consequences of assuming the von Bertalanffy growth model should be considered when using the DLMtool for species with alternative growth patterns.

9.1.4 Natural Mortality Rate at Age

DLMtool currently assumes that natural mortality (M) is constant with age. Age-specific M may be added in a future developments.

9.2 MSE Model Assumptions

9.2.1 Idealised Observation Models for Catch Composition Data

Currently, DLMtool simulates catch-composition data from the true simulated catch composition data via a multinomial distribution and some effective sample size. This observation model may be unrealistically well-behaved and favour those approaches that use these data. We are considering adding a growth-type-group model to improve the realism of simulated length composition data.

9.2.2 Implementation Error

In this edition of DLMtool there is no implementation error. The only imperfection between a management recommendation and the simulated TAC comes in the form of the MaxF argument that limits the maximum fishing mortality rate on any given age-class in the operating model. The default is 0.8 which is high for all but the shortest living fish species.

9.2.3 Discard Mortality

Related to the previous point, DLMtool assumes that there is no discard mortality or fishing mortality for fish that are below the length of selectivity. This can be an important assumption, particularly when evaluating the impact of size-selectivity management methods such as a minimum legal length.

9.2.4 Two-Box Model

DLMtool uses a two-box spatial model and assumes homogeneous fishing, and distribution of the fish stock. That is, growth and other life-history characteristics do not vary across the two spatial areas. Spatial targeting of the fishing fleet is currently being developed in the model.

9.2.5 Ontogenetic Habitat Shifts

Since the operating model simulates two areas, it is possible to prescribe a log-linear model that moves fish from one area to the other as they grow older. This could be used to simulate the ontogenetic shift of groupers from near shore waters to offshore reefs. Currently this feature is in development.

9.2.6 Closed System

DLMtool assumes that the population being modelled is in a closed system. There is no immigration or emigration, and a unit stock is assumed to be represented in the model and impacted by the management decisions. This assumption may be violated where the stock extends across management jurisdictions. Violations of this assumption may impact the interpretation of the MSE results, and these implications should be considered when applying DLMtool.

Although a unit stock is a central assumption of many modeling and assessment approaches, it may be possible to further develop DLMtool to account for stocks that cross management boundaries.

9.2.7 Marine Protected Areas (MPA)

MPAs can be evaluated as management action in DLMtool by closing one area to fishing in the future projections. Currently, it is not possible to include MPAs in future projections alongside other management methods such as TAC or effort controls. Similarly, further development of DLMtool is required to fully incorporate existing MPA networks into the MSE simulations

9.3 Management Procedures

9.3.1 Harvest Control Rules Must be Integrated into Data-Limited MPs

In this version of DLMtool, harvest control rules (e.g. the 40-10 rule) must be written into a data-limited MP. There is currently no ability to do a factorial comparison of say 4 harvest controls rules against 3 MPs (the user must describe all 12 combinations). The reason for this is that it would require further subclasses.

For example the 40-10 rule may be appropriate for the output of DBSRA but it would not be appropriate for some of the simple management procedures such as DynF that already incorporate throttling of TAC recommendations according to stock depletion.

9.4 Data and Method Application

9.4.1 Data Assumed to be Representative

The MSE model accounts for observation error in the simulated fishery data. However, the application of management procedures for management advice assumes that the provided fishery data is representative of the fishery and is the best available information on the stock. Processing of fishery data should take place before entering the data into the fishery data tables, and assumptions of the management procedures should be carefully evaluated when applying methods using DLMtool.

Chapter 10

References

Beverton, R. J. H., & Holt, S. J. (1957). *On the dynamics of exploited fish populations.* Fishery Investigation Series 2, United Kingdom Ministry of Agriculture and Fisheries, (Vol. 19). Book, London, United Kingdom.

Butterworth, D. S. (2007). Why a management procedure approach? Some positives and negatives. *ICES Journal of Marine Science: Journal Du Conseil*, 64(1995), 613–617.

Costello, C., Ovando, D., Hilborn, R., Gaines, S. D., Deschenes, O., & Lester, S. E. (2012). Status and solutions for the world's unassessed fisheries. *Science*, 338, 517–520.

Newman, D., Berkson, J., & Suatoni, L. (2015). Current methods for setting catch limits for data-limited fish stocks in the United States. *Fisheries Research*, 164, 86–93.

Punt, A. E. (2015). Strategic management decision-making in a complex world: quantifying, understanding, and using trade-offs. *ICES Journal of Marine Science*, (fsv193), 12.

Punt, A. E., Butterworth, D. S., de Moor, C. L., De Oliveira, J. A. A., & Haddon, M. (2014). Management strategy evaluation: best practices. *Fish and Fisheries.*

Restrepo, V., Thompson, G. G., Mace, P., Gabriel, W., Low, L., MacCall, A., Methot, R.D., Powers, J.E., Taylor, B., Wade, P.R., & Witzig, J. (1998). Guidance on the use of precautionary approaches to implementing National Standard 1 of the Magnuson-Stevens Fishery Conservation and Management. NOAA Technical Memorandum.

Walters, C. J., & Martell, S. J. D. (2004). *Fisheries ecology and management.* Book, Princeton, USA: Princeton University Press.