Data-Limited Methods Toolkit (DLMtool 5.1)

User Guide

Tom Carruthers & Adrian Hordyk

2018-03-17

Contents

Introduction

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 90 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 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          4.3
## year           2017
## month          11
## day            30
## svn rev        73796
## language       R
## version.string R version 3.4.3 (2017-11-30)
## nickname       Kite-Eating Tree
```

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

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

```
## [1] '5.1'
```

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.

1.5 The User Manual

This user manual has been designed to introduce users to DLMtool and does not assume prior knowledge of DLMtool or extensive knowledge of R. Some familiarity with the concept of Management Strategy Evaluation and the commonly used parameters and data types is assumed.

The user manual is continually being developed and we could use your help!

We've tried to design it from the perspective of someone who is brand new to DLMtool. But there are undoubtedly many ways in which it can be improve. Please contact us through our website or email us directly if you have any questions or suggestions for improvement.

Bug or typos can be reported on the userguide GitHub issues page.

Pull requests with edits are most welcome.

1.6 DLMtool Bug Reports

The package is subject to ongoing development and testing. If you find a bug or a problem please contact us or report an issue on GitHub so that it can be fixed. If possible, please provide a minimal reproducible example so that we can recreate the problem and fit it.

1.7 Version Notes

The current version of the DLMtool package, available for download from CRAN, is 5.1.

Version notes for previous versions of DLMtool can be found at DLMtool News

Getting Started with DLMtool

Chapter 2

Getting Started

2.1 Required Software

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.

2.1.1 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.

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          4.3
## year           2017
## month          11
## day            30
## svn rev        73796
## language       R
## version.string R version 3.4.3 (2017-11-30)
## nickname       Kite-Eating Tree
```

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 DLMtool

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:

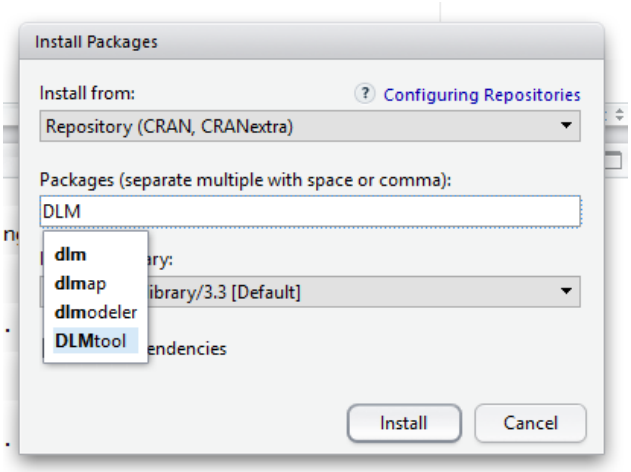
```
install.packages("DLMtool")
```

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 in 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 Checking DLMtool version

You can confirm the version of DLMtool by typing:

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

```
## [1] '5.1'
```

2.3 Loading DLMtool

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.

You need to load the DLMtool package each time you start a new instance of R.

Chapter 3

A Very Quick Demo

Running an MSE with DLMtool is quite straightforward and only requires a single line of code:

```
myMSE <- runMSE()
```

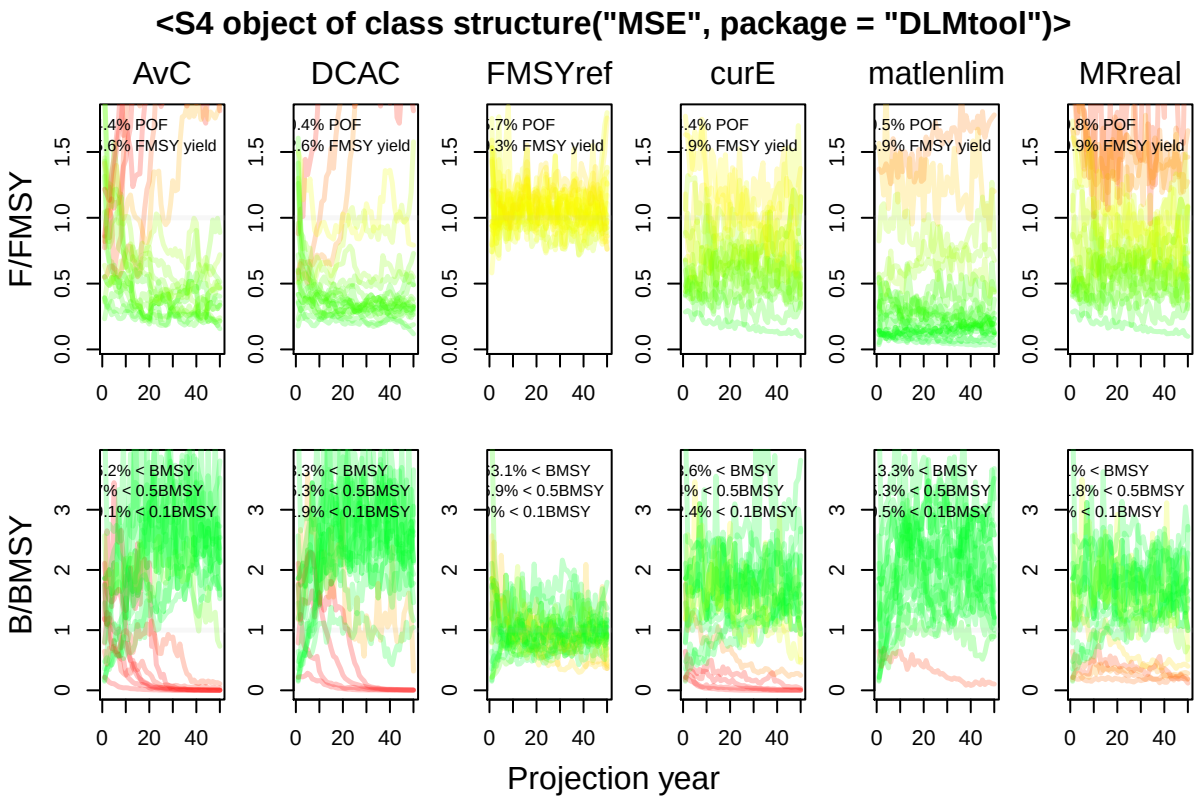
```
## Loading operating model
## Optimizing for user-specified movement
## Optimizing for user-specified depletion
## Calculating historical stock and fishing dynamics
## Calculating MSY reference points
## Calculating Blow reference points
## Calculating reference yield - best fixed F strategy
## Calculating MSY reference points for each projection year
## ..............................................
## 1/6 Running MSE for AvC
## ..............................................
## 2/6 Running MSE for DCAC
## ..............................................
## 3/6 Running MSE for FMSYref
## ..............................................
## 4/6 Running MSE for curE
## ..............................................
## 5/6 Running MSE for matlenlim
## ..............................................
## 6/6 Running MSE for MRreal
## ..............................................
```

If you run this line (remember, if you haven't already you must first run `library(DLMtool)`) and see something similiar to the output shown here, then DLMtool is successfully working on your system!

If the MSE did not run successfully, repeat the previous steps, ensuring that you have the latest version of R and the DLMtool package. If still no success, please contact us with a description of the problem and we will try to help.

Once an MSE is run, the results can be examined visually using plotting functions, for example:

```
Pplot(myMSE)
```



Or quantified in various ways, for example:

```
summary(myMSE)
```

```
## Calculating Performance Metrics

##                                    Performance.Metrics
## 1                  Average Annual Variability in Yield
## 2  Average Long-Term Yield relative to Reference Yield
## 3          Probability Spawning Biomass above 10% BMSY
## 4                  Probability Spawning Biomass > BMSY
## 5          Probability Spawning Biomass above 50% BMSY
## 6                                 Probability F < FMSY
## 7 Average Short-Term Yield relative to Reference Yield
## 8                     Yield relative to Reference Yield
##
##
## Probability:
##          MP AAVY  LTY  P10 P100  P50  POF  STY Yield
## 1       AvC 1.00 0.49 0.80 0.67 0.75 0.68 0.76  0.67
## 2      DCAC 1.00 0.59 0.90 0.80 0.86 0.82 0.74  0.68
```

```
## 3   FMSYref 1.00 0.96 1.00 0.35 0.88 0.33 0.91  1.10
## 4      curE 0.25 0.74 0.94 0.70 0.84 0.72 0.84  0.84
## 5 matlenlim 0.25 0.63 1.00 0.93 0.98 0.96 0.44  0.65
## 6    MRreal 0.29 0.79 1.00 0.73 0.86 0.76 0.84  0.84
```

Chapter 4

The Operating Model

The Operating Model (OM) is the main component of the MSE framework. The OM is used to describe the characterstics of a fishery system and contains all the parameters required to simulate the population and fleet dynamics, the collection of data, and the application of a management procedure (e.g., implement a size regulation, effort control, spatial closure, or catch limit).

4.1 OM Components

An OM is built from four separate components, each containing a set of parameter values for different aspects of the simulation:

1. Stock - parameters describing the stock dynamics
2. Fleet - parameters describing the fishing fleet dynamics
3. Obs (Observation) - parameters describing the observation processes (how the observed fishery data is generated from the simulated data)
4. Imp (Implementation) - parameters describing the management implemetation (how well the management regulations are implemented)

There are a number of example Stock, Fleet, Obs, and Imp parameter sets built into DLMtool which make it easy to quickly construct an OM and run an MSE.

These parameter sets are referred to as *Objects* and have an associated *Class*.

4.1.1 Stock Object

The `avail` function can be used to examine the available Objects of a particular Class.

For example, to see the available objects of class *Stock*:

```
avail('Stock')
```

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

This shows that there are 12 objects of class *Stock*. We can confirm the class of this object by using the `class` function. For example, to examine the class of the object `Albacore`:

```
class(Albacore)
```

```
## [1] "Stock"
## attr(,"package")
## [1] "DLMtool"
```

As expected, the `Albacore` object is class *Stock*.

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

```
slotNames(Albacore)
```

```
##  [1] "Name"         "Common_Name"  "Species"      "maxage"
##  [5] "R0"           "M"            "M2"           "Mexp"
##  [9] "Msd"          "Mgrad"        "h"            "SRrel"
## [13] "Perr"         "AC"           "Period"       "Amplitude"
## [17] "Linf"         "K"            "t0"           "LenCV"
## [21] "Ksd"          "Kgrad"        "Linfsd"       "Linfgrad"
## [25] "L50"          "L50_95"       "D"            "a"
## [29] "b"            "Size_area_1"  "Frac_area_1"  "Prob_staying"
## [33] "Fdisc"        "Source"
```

The output tells us that there are 34 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] "Albacore"
```

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
```

4.1.2 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 DLMtoo:

```
avail('Fleet')
```

```
##  [1] "DecE_Dom"              "DecE_HDom"
##  [3] "DecE_NDom"             "FlatE_Dom"
##  [5] "FlatE_HDom"            "FlatE_NDom"
##  [7] "Generic_DecE"          "Generic_FlatE"
##  [9] "Generic_Fleet"         "Generic_IncE"
## [11] "IncE_HDom"             "IncE_NDom"
## [13] "Low_Effort_Non_Target" "Target_All_Fish"
## [15] "Targeting_Small_Fish"
```

Here we will look at the `Generic_Fleet` object.

```
class(Generic_Fleet)
```

```
## [1] "Fleet"
## attr(,"package")
## [1] "DLMtool"
```

```
slotNames(Generic_Fleet)
```

```
##  [1] "Name"        "nyears"      "Spat_targ"   "EffYears"    "EffLower"
##  [6] "EffUpper"    "Esd"         "qinc"        "qcv"         "L5"
## [11] "LFS"         "Vmaxlen"     "isRel"       "LR5"         "LFR"
## [16] "Rmaxlen"     "DR"          "SelYears"    "AbsSelYears" "L5Lower"
## [21] "L5Upper"     "LFSLower"    "LFSUpper"    "VmaxLower"   "VmaxUpper"
## [26] "CurrentYr"   "MPA"
```

There are 27 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
```

4.1.3 Obs 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* object?

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 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.

```
avail('Obs')
```

```
## [1] "Generic_Obs"        "Imprecise_Biased"   "Imprecise_Unbiased"
## [4] "Perfect_Info"       "Precise_Biased"     "Precise_Unbiased"
```

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

```
class(Imprecise_Unbiased)
```

```
## [1] "Obs"
## attr(,"package")
## [1] "DLMtool"
```

```
slotNames(Imprecise_Unbiased)
```

```
##  [1] "Name"          "Cobs"          "Cbiascv"       "CAA_nsamp"
##  [5] "CAA_ESS"       "CAL_nsamp"     "CAL_ESS"       "Iobs"
##  [9] "Ibiascv"       "Btobs"         "Btbiascv"      "beta"
## [13] "LenMbiascv"    "Mbiascv"       "Kbiascv"       "t0biascv"
## [17] "Linfbiascv"    "LFCbiascv"     "LFSbiascv"     "FMSYbiascv"
## [21] "FMSY_Mbiascv"  "BMSY_B0biascv" "Irefbiascv"    "Brefbiascv"
## [25] "Crefbiascv"    "Dbiascv"       "Dobs"          "hbiascv"
## [29] "Recbiascv"
```

There are 29 slots in *Obs* objects, each with information relating to the uncertainty of a data type.

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

```
Imprecise_Biased@LenMbiascv
```

```
## [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
```

4.1.4 Imp 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, and variability in size regulations.

```
avail('Imp')
```

```
## [1] "Overages"    "Perfect_Imp"
```

```
class(Overages)
```

```
## [1] "Imp"
## attr(,"package")
## [1] "DLMtool"
```

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

```
class?Imp
```

4.2 Building an OM from Component Objects

We will now look at how to combine objects of the four classes into an OM. For now we will work with the OM components that are built into DLMtool. In later sections of the user manual we will cover how to build your own Stock, Fleet, Obs, and Imp objects that characterises your fishery.

Objects of class *Stock*, *Fleet*, *Obs* and *Imp* are used to create an Operating Model object (class `OM`). The simplest way to do this is to use `new` command.

For example, here we are building a OM using the `Rockfish` Stock object, `Generic_Fleet` Fleet object, `Generic_Obs` Obs object, and `Perfect_Imp` Imp object and assigning it the name `myOM`:

```
myOM <- new("OM", Rockfish, Generic_Fleet, Generic_Obs, Perfect_Imp)
```

What is the class of our newly created objects `myOM`?

```
class(myOM)
```

```
## [1] "OM"
## attr(,"package")
## [1] "DLMtool"
```

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*, *Obs*, and *Imp* objects:

```
slotNames(myOM)
```

```
##   [1] "Name"          "Agency"        "Region"        "Sponsor"
##   [5] "Latitude"      "Longitude"     "nsim"          "proyears"
##   [9] "interval"      "pstar"         "maxF"          "reps"
##  [13] "cpars"         "seed"          "Source"        "Common_Name"
##  [17] "Species"       "maxage"        "R0"            "M"
##  [21] "M2"            "Mexp"          "Msd"           "Mgrad"
```

```
##  [25] "h"             "SRrel"         "Perr"          "AC"
##  [29] "Period"        "Amplitude"     "Linf"          "K"
##  [33] "t0"            "LenCV"         "Ksd"           "Kgrad"
##  [37] "Linfsd"        "Linfgrad"      "L50"           "L50_95"
##  [41] "D"             "a"             "b"             "Size_area_1"
##  [45] "Frac_area_1"   "Prob_staying"  "Fdisc"         "nyears"
##  [49] "Spat_targ"     "EffYears"      "EffLower"      "EffUpper"
##  [53] "Esd"           "qinc"          "qcv"           "L5"
##  [57] "LFS"           "Vmaxlen"       "isRel"         "LR5"
##  [61] "LFR"           "Rmaxlen"       "DR"            "SelYears"
##  [65] "AbsSelYears"   "L5Lower"       "L5Upper"       "LFSLower"
##  [69] "LFSUpper"      "VmaxLower"     "VmaxUpper"     "CurrentYr"
##  [73] "MPA"           "Cobs"          "Cbiascv"       "CAA_nsamp"
##  [77] "CAA_ESS"       "CAL_nsamp"     "CAL_ESS"       "Iobs"
##  [81] "Ibiascv"       "Btobs"         "Btbiascv"      "beta"
##  [85] "LenMbiascv"    "Mbiascv"       "Kbiascv"       "t0biascv"
##  [89] "Linfbiascv"    "LFCbiascv"     "LFSbiascv"     "FMSYbiascv"
##  [93] "FMSY_Mbiascv"  "BMSY_B0biascv" "Irefbiascv"    "Brefbiascv"
##  [97] "Crefbiascv"    "Dbiascv"       "Dobs"          "hbiascv"
## [101] "Recbiascv"     "TACFrac"       "TACSD"         "TAEFrac"
## [105] "TAESD"         "SizeLimFrac"   "SizeLimSD"
```

You can access individual slots in the OM object using the @ symbol and confirm that these values are the same as those in the Stock object used to create the OM:

```
Rockfish@M
```

```
## [1] 0.04 0.08
```

```
myOM@M
```

```
## [1] 0.04 0.08
```

In addition to the information from the Stock, Fleet, Obs, and Imp objects, the OM object also contains other values relating to the MSE, including the number of simulations to run (`nsim`), the number of projection years (`proyears`), and the management interval (`interval`):

```
myOM@nsim
```

```
## [1] 48
```

```
myOM@proyears
```

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

```
myOM@interval
```

```
## [1] 4
```

These slots all have default values that can be modified easily, for example:

```
myOM@proyears <- 60
myOM@proyears
```

```
## [1] 60
```

Remember, you can access the help information for objects by typing ? followed by the class name, for example:

```
class?OM
```

In later chapters we will cover a range of methods to build new Stock, Fleet, Obs, and Imp objects and constructing OMs that characterise your fishery.

4.3 Visualizing an OM

The newly created OM object `myOM` contains all the parameters that will be used to simulate our fishery, both the historical conditions and the future projections. The OM can visualized with the `plot` function (plots not shown here):

```
plot(myOM)
```

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 Available Management Procedure

All management procedures in DLMtool are objects (actually functions in this case) of class `MP`. There are a number of MPs built into DLMtool. The `avail` function can be used to provide a list of MPs that can be included in the MSE:

```
avail('MP')
```

```
##  [1] "AvC"          "AvC_MLL"      "BK"           "BK_CC"
##  [5] "BK_ML"        "CC1"          "CC4"          "CompSRA"
##  [9] "CompSRA4010"  "curE"         "curE75"       "DAAC"
## [13] "DBSRA"        "DBSRA_40"     "DBSRA4010"    "DCAC"
## [17] "DCAC_40"      "DCAC_ML"      "DCAC4010"     "DD"
## [21] "DD4010"       "DDe"          "DDe75"        "DDes"
## [25] "DepF"         "DTe40"        "DTe50"        "DynF"
## [29] "EtargetLopt"  "Fadapt"       "Fdem"         "Fdem_CC"
```

```
## [33] "Fdem_ML"      "FMSYref"      "FMSYref50"    "FMSYref75"
## [37] "Fratio"       "Fratio_CC"    "Fratio_ML"    "Fratio4010"
## [41] "GB_CC"        "GB_slope"     "GB_target"    "Gcontrol"
## [45] "HDAAC"        "Islope1"      "Islope4"      "IT10"
## [49] "IT5"          "Itarget1"     "Itarget1_MPA" "Itarget4"
## [53] "ItargetE1"    "ItargetE4"    "ITe10"        "ITe5"
## [57] "ITM"          "L95target"    "LstepCC1"     "LstepCC4"
## [61] "LstepCE1"     "LstepCE2"     "Ltarget1"     "Ltarget4"
## [65] "LtargetE1"    "LtargetE4"    "matlenlim"    "matlenlim2"
## [69] "MCD"          "MCD4010"      "minlenLopt1"  "MRnoreal"
## [73] "MRreal"       "NFref"        "Rcontrol"     "Rcontrol2"
## [77] "SBT1"         "SBT2"         "slotlim"      "SPmod"
## [81] "SPMSY"        "SPslope"      "SPSRA"        "SPSRA_ML"
## [85] "YPR"          "YPR_CC"       "YPR_ML"       "TCPUE"
## [89] "TCPUE_e"      "THC"
```

As you can see, there are 90 MPs built into the DLMtool.

DLMtool is extensible and it is relatively straightforward to develop your own MPs and include them in the MSE. This is covered in Developing Custom Management Procedures.

5.3 Types of Management Procedure

In previous versions of DLMtool, the MPs were divided into two classes: Output controls which returned a total allowable catch (**TAC**) and Input controls which allow regulation of **fishing effort**, **size selecitivty**, or **spatial area**.

Since DLMtool V5.1 it is possible to include MPs that provide a combination of input and output controls.

All MPs in DLMtool are now class `MP`, but the MPs are divided into four types: **Input** which allow regulation offishing effort, size selectivity, or spatial area but not a TAC, **Output** which return only a TAC recommendation, **Mixed** which return a combination of one or several input controls *and* a TAC, and **Reference** which are MPs that have been designed to be used as reference management procedures (e.g `FMSYref` which uses perfect information of FMSY and abundance).

The `MPtype` function can be used to display the type for a particular MP, for example:

```
MPtype("DCAC")
```

```
##     MP   Type
## 1 DCAC Output
```

Or for all available MPs:

```
MPtype(avail('MP'))
```

```
##              MP      Type
## 10         curE     Input
## 11       curE75     Input
## 22          DDe     Input
## 23        DDe75     Input
## 24         DDes     Input
## 26        DTe40     Input
## 27        DTe50     Input
## 29  EtargetLopt     Input
## 53    ItargetE1     Input
## 54    ItargetE4     Input
```

```
## 55        ITe10     Input
## 56         ITe5     Input
## 61     LstepCE1     Input
## 62     LstepCE2     Input
## 65    LtargetE1     Input
## 66    LtargetE4     Input
## 67    matlenlim     Input
## 68   matlenlim2     Input
## 71  minlenLopt1     Input
## 72     MRnoreal     Input
## 73       MRreal     Input
## 79      slotlim     Input
## 89      TCPUE_e     Input
## 2       AvC_MLL     Mixed
## 51 Itarget1_MPA     Mixed
## 1           AvC    Output
## 3            BK    Output
## 4         BK_CC    Output
## 5         BK_ML    Output
## 6           CC1    Output
## 7           CC4    Output
## 8       CompSRA    Output
## 9   CompSRA4010    Output
## 12         DAAC    Output
## 13        DBSRA    Output
## 14     DBSRA_40    Output
## 15    DBSRA4010    Output
## 16         DCAC    Output
## 17      DCAC_40    Output
## 18      DCAC_ML    Output
## 19     DCAC4010    Output
## 20           DD    Output
## 21       DD4010    Output
## 25         DepF    Output
## 28         DynF    Output
## 30       Fadapt    Output
## 31         Fdem    Output
## 32      Fdem_CC    Output
## 33      Fdem_ML    Output
## 37       Fratio    Output
## 38    Fratio_CC    Output
## 39    Fratio_ML    Output
## 40   Fratio4010    Output
## 41        GB_CC    Output
## 42     GB_slope    Output
## 43    GB_target    Output
## 44     Gcontrol    Output
## 45        HDAAC    Output
## 46      Islope1    Output
## 47      Islope4    Output
## 48         IT10    Output
## 49          IT5    Output
## 50     Itarget1    Output
## 52     Itarget4    Output
```

```
## 57          ITM    Output
## 58    L95target    Output
## 59     LstepCC1    Output
## 60     LstepCC4    Output
## 63     Ltarget1    Output
## 64     Ltarget4    Output
## 69          MCD    Output
## 70       MCD4010    Output
## 75     Rcontrol    Output
## 76    Rcontrol2    Output
## 77         SBT1    Output
## 78         SBT2    Output
## 80        SPmod    Output
## 81        SPMSY    Output
## 82      SPslope    Output
## 83        SPSRA    Output
## 84     SPSRA_ML    Output
## 85          YPR    Output
## 86       YPR_CC    Output
## 87       YPR_ML    Output
## 88        TCPUE    Output
## 90          THC    Output
## 34      FMSYref Reference
## 35    FMSYref50 Reference
## 36    FMSYref75 Reference
## 74        NFref Reference
```

You can access help documentation for the MPs in the usual fashion, for example:

```
?DCAC
```

5.3.1 Input Control MPs

Input controls allow some combination of adjustments to **fishing effort**, **size selecitivty**, or **spatial area**.

The available input control MPs are:

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

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

Remember, to access help documentation:

```
?matlenlim
```

More information on input control MPs can be found in Beyond the Catch Limit.

5.3.2 Output Control MPs

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.

The available output controls are:

```
avail('Output')
```

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

5.3.3 Mixed MPs

Mixed MPs return a combination of input and output controls. Currently there are only a few mixed MPs in DLMtool, and these were developed simply for demonstration purposes. They may not work very well! See Developing Custom Management Procedures for more information on developing your own mixed MPs. And please share them with us, we'd love to add them to DLMtool!

```
avail('Mixed')
```

```
## [1] "AvC_MLL"      "Itarget1_MPA"
```

5.3.4 Reference MPs

The final type is the reference MPs. These MPs are not designed to be used in practice, but are useful for providing a reference for comparing for the performance of other MPs. For example, the `FMSYref` and `NFref` methods (fishing perfectedly at F[MSY] and no fishing at all) can be useful for framing realistic performance with respect to a set of management objectives.

The available reference MPs are:

```
avail('Reference')
```

```
## [1] "FMSYref"   "FMSYref50" "FMSYref75" "NFref"
```

Chapter 6

Running the MSE

We have now covered the two main components of the MSE: the Operating Model (OM) and the Management Procedures (MPs). To run a MSE we need to specify the OM and the set of MPs that we wish to test.

Here we will create an OM from the built-in objects and choose 2 MPs of each type to test in our demonstration MSE.

6.1 Specify an Operating Model

First, we will construct the OM using a different set of built-in Stock, Fleet, Obs, and Imp objects:

```
myOM <- new("OM", Albacore, DecE_Dom, Imprecise_Unbiased, Overages)
```

6.2 Choose the Management Procedures

Next, we'll select 8 MPs to test in our MSE:

```
myMPs <- c('AvC', 'Itarget1',
           'matlenlim', 'ITe10',
           'AvC_MLL', 'Itarget1_MPA',
           'FMSYref', 'NFref')

MPtype(myMPs)
```

```
##             MP      Type
## 3    matlenlim     Input
## 4        ITe10     Input
## 5      AvC_MLL     Mixed
## 6 Itarget1_MPA     Mixed
## 1          AvC    Output
## 2     Itarget1    Output
## 7      FMSYref Reference
## 8        NFref Reference
```

6.3 Run the MSE

Now that we have specified an OM and chosen a set of management procedures we are ready to run the MSE:

```
myMSE <- runMSE(OM=myOM, MPs=myMPs)
```

```
## Loading operating model
## Optimizing for user-specified movement
## Optimizing for user-specified depletion
## Calculating historical stock and fishing dynamics
## Calculating MSY reference points
## Calculating Blow reference points
## Calculating reference yield - best fixed F strategy
## Calculating MSY reference points for each projection year
## ................................................
## 1/8 Running MSE for AvC
## ................................................
## 2/8 Running MSE for Itarget1
## ................................................
## 3/8 Running MSE for matlenlim
## ................................................
## 4/8 Running MSE for ITe10
## ................................................
## 5/8 Running MSE for AvC_MLL
## ................................................
## 6/8 Running MSE for Itarget1_MPA
## ................................................
## 7/8 Running MSE for FMSYref
## ................................................
## 8/8 Running MSE for NFref
## ................................................
```

Chapter 7

Checking Convergence

It is important to ensure that we have included enough simulations in the MSE for the results to be stable.

The `Converge` function can be used 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 and the performance statistics have stablized, i.e., they won't change significantly if the model was run with more simulations.

The purpose of the `Converge` function is to answer the question: have I run enough simulations?

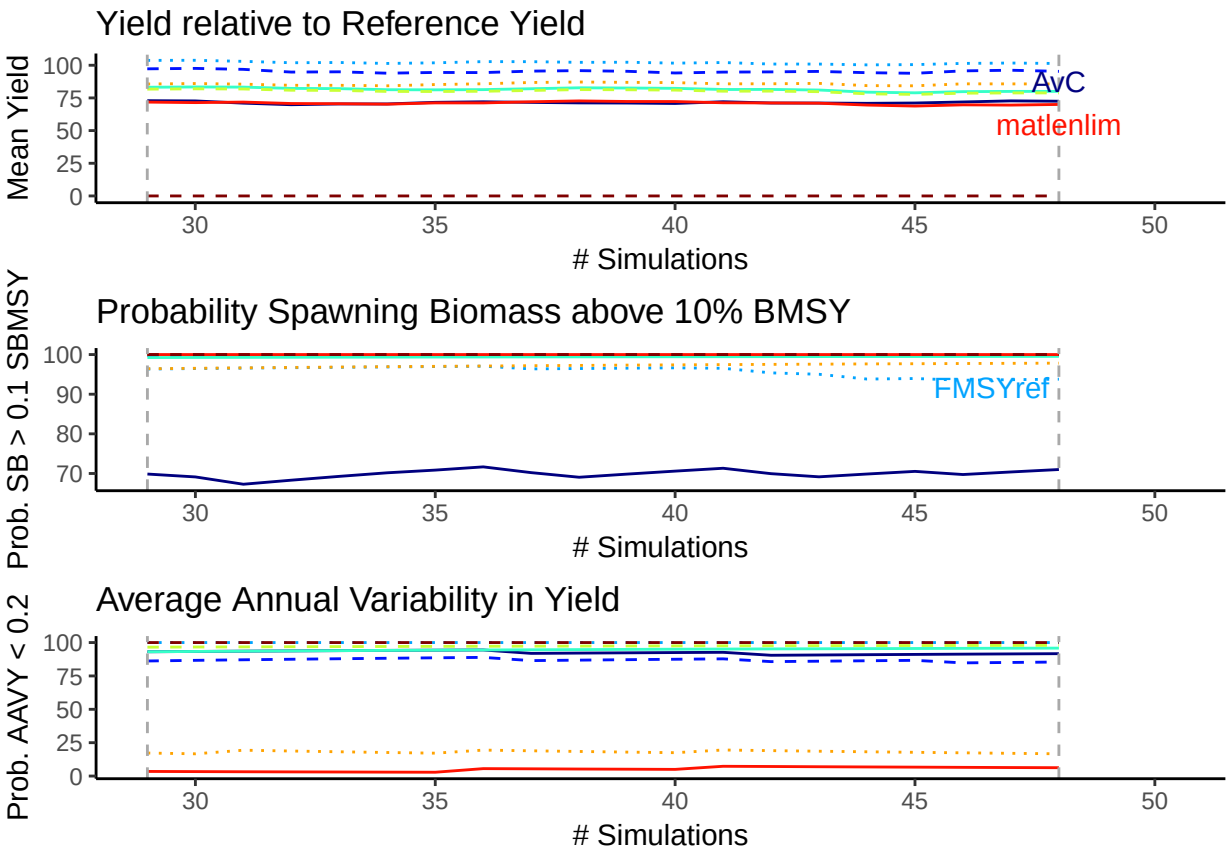
By default the `Converge` function includes three commonly used performance metrics, and plots the performance statistics against the number of simulations. The convergence diagnostics are:

1. Does the order of the MPs change as more simulations are added? By default this is calculated over the last 20 simulations.
2. Is the average difference in the performance statistic over the last 20 simulations changing by more than 2%?

The number of simulations to calculate the convergence statistics, the minimum change threshold, and the performance metrics to use can be specified as arguments to the function. See the help documentation for more details (`?Converge`).

```
Converge(myMSE)
```

```
## Checking if order of MPs is changing in last 20 iterations

## Checking average difference in PM over last 20 iterations is > 2

## Plotting MPs 1 - 8

##
## Yield relative to Reference Yield

## Order over last 20 iterations is not consistent for:
##  AvC matlenlim

##
## Probability Spawning Biomass above 10% BMSY

## Mean difference over last 20 iterations is > 2 for:
## FMSYref
```



Have we run enough simulations?

The convergence plot reveals that both the order of the MPs and the performance statistics are not stable. This suggests that 48 simulations is not enough to produce reliable results.

Let's increase the number of simulations and try again:

```
myOM@nsim <- 200
```

```
myMSE_200 <- runMSE(OM=myOM, MPs=myMPs)
```

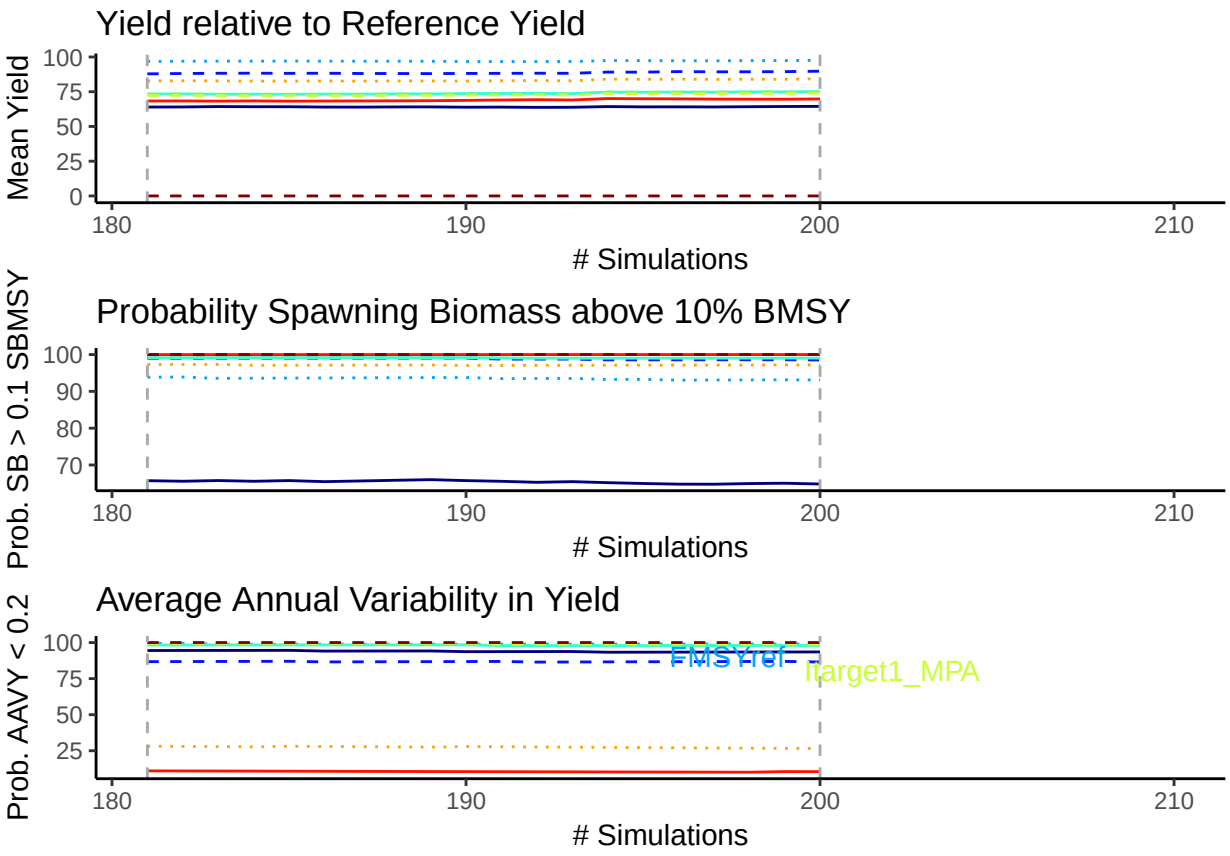
```
## Suggest using 'parallel = TRUE' for large number of simulations
## Loading operating model
## Optimizing for user-specified movement
## Optimizing for user-specified depletion
## 2 simulations have final biomass that is not close to sampled depletion
## Re-sampling depletion, recruitment error, and fishing effort
## Calculating historical stock and fishing dynamics
## Calculating MSY reference points
## Calculating Blow reference points
## Calculating reference yield - best fixed F strategy
## Calculating MSY reference points for each projection year
## ................................................
```

```
## 1/8 Running MSE for AvC

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

## 2/8 Running MSE for Itarget1

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

## 3/8 Running MSE for matlenlim

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

## 4/8 Running MSE for ITe10

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

## 5/8 Running MSE for AvC_MLL

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

## 6/8 Running MSE for Itarget1_MPA

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

## 7/8 Running MSE for FMSYref

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

## 8/8 Running MSE for NFref

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

Is 200 simulations enough?

```
Converge(myMSE_200)
```

```
## Checking if order of MPs is changing in last 20 iterations

## Checking average difference in PM over last 20 iterations is > 2

## Plotting MPs 1 - 8

##
## Average Annual Variability in Yield

## Order over last 20 iterations is not consistent for:
##  FMSYref Itarget1_MPA
```


Yield relative to Reference Yield
Mean Yield
100
75
50
25
0
180
190
200
210
Simulations
Probability Spawning Biomass above 10% BMSY
Prob. SB > 0.1 SBMSY
100
90
80
70
180
190
200
210
Simulations
Average Annual Variability in Yield
Prob. AAVY < 0.2
100
75
50
25
FMSYref
Target1_MPA
180
190
200
210
Simulations

Chapter 8

Plotting the MSE Results

The DLMtool has many built-in plotting functions which can be used to examine the performance of the Management Procedures. Advanced users can also develop their own plotting and summary functions. See the Performance Metrics chapter for more details.

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

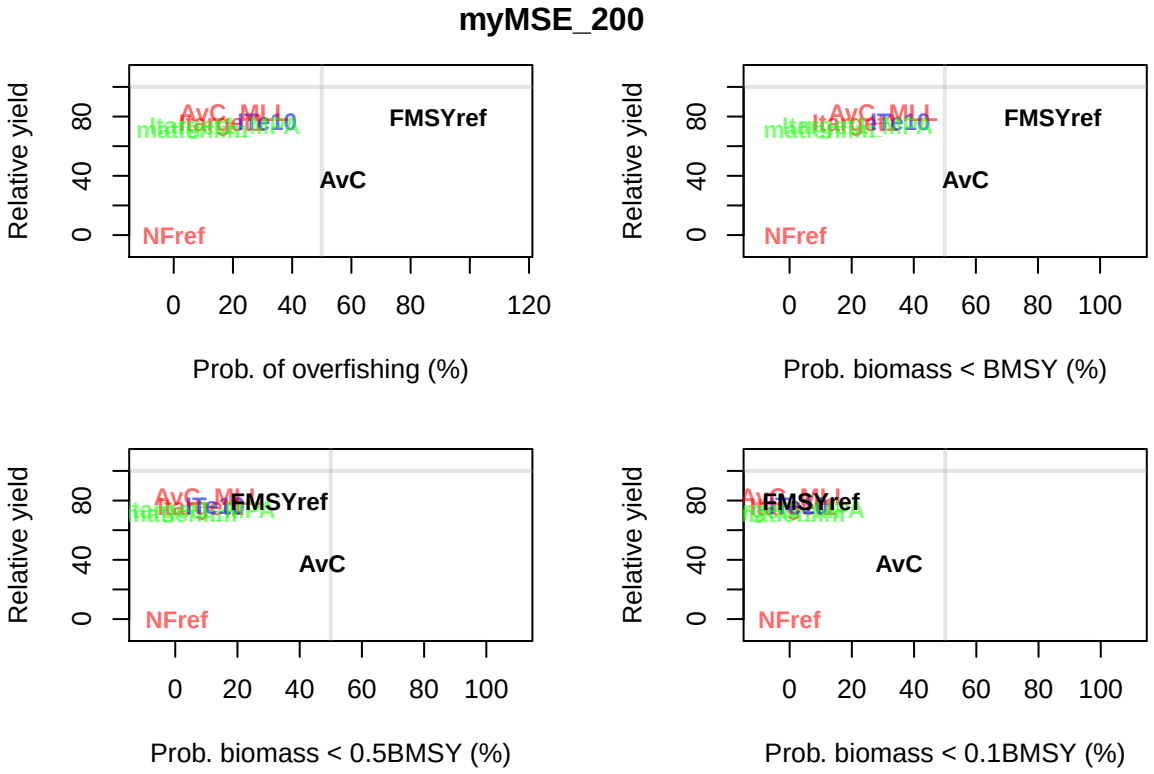
```
plotFun()
```

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

## barplot COSEWIC_plot Cplot DFO_hist DFO_plot
## DFO_plot2 DFO_proj IOTC_plot Kplot NOAA_plot
## plotOFL Pplot Pplot2 PWhisker Tplot
## Tplot2 Tplot3 TradePlot VOI VOI2
## VOIplot wormplot
```

8.1 Trade-Off Plots

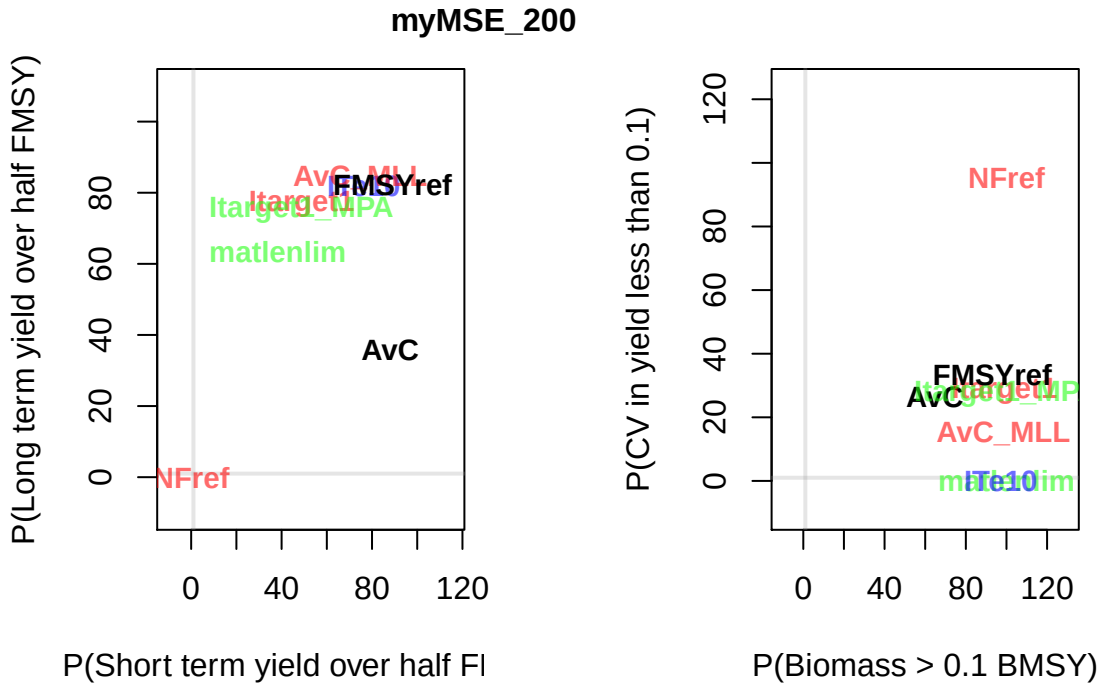
The `Tplot` function creates four plots that show the trade-off between the expected relative yield and the probability of overfishing and the probability of the biomass being below three different reference points:

```
Tplot(myMSE_200)
```



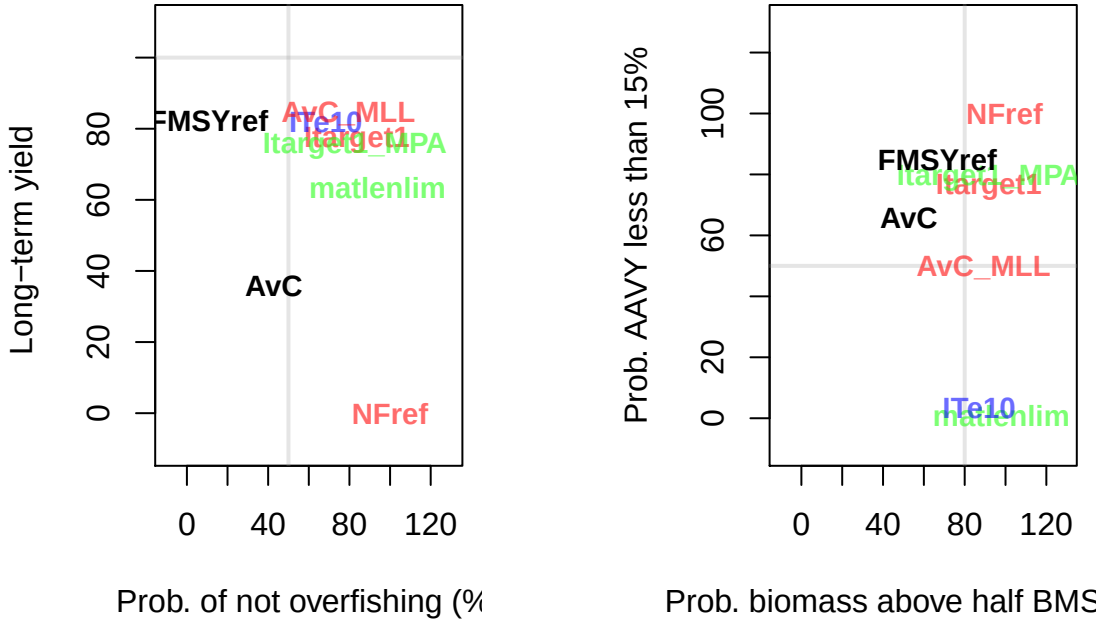
The `Tplot2` function shows the trade-off between long-term and short-term yield, and the trade-off between biomass being above $0.1B_{MSY}$ and the expected variability in the yield:

```
Tplot2(myMSE_200)
```



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%:

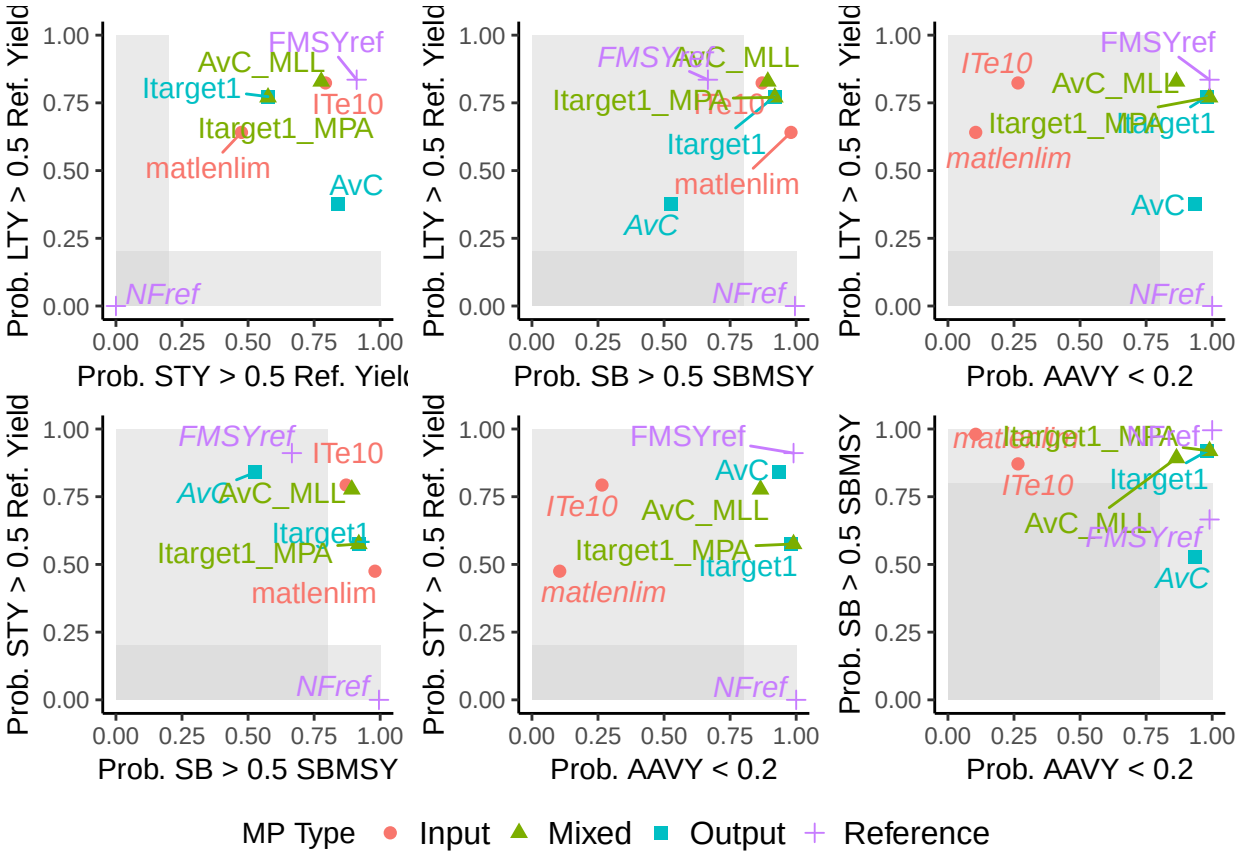
```
NOAA_plot(myMSE_200)
```




```
##                PNOF  B50  LTY    VY
## AvC            42.8 52.7 35.8  66.0
## Itarget1       83.7 91.8 76.9  76.0
## matlenlim      93.8 98.0 63.5   1.0
## ITe10          68.4 87.1 82.0   3.5
## AvC_MLL        79.5 89.3 84.3  49.5
## Itarget1_MPA   82.7 91.9 75.3  79.0
## FMSYref        10.7 66.6 82.4  85.0
## NFref         100.0 99.5  0.0 100.0
```

The `Tplot3` function has been designed to take a list of Performance Metrics objects, plot the performance metric against each other, and report if an MP has met the minimum conditions for all performance metrics:

```
Tplot3(myMSE_200)
```

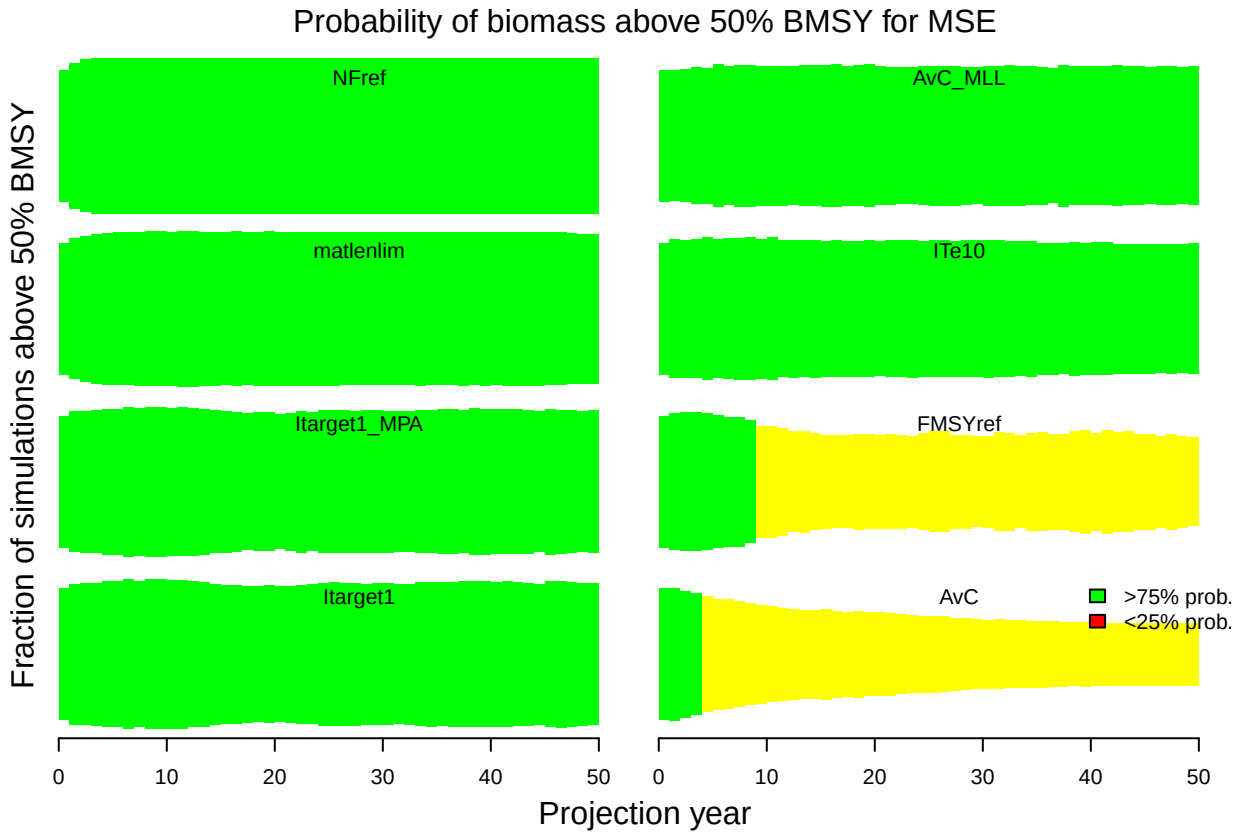



```
##             MP  LTY  STY  P50 AAVY Satisificed
## 1          AvC 0.38 0.84 0.53 0.94        FALSE
## 2     Itarget1 0.77 0.58 0.92 0.98         TRUE
## 3    matlenlim 0.64 0.48 0.98 0.10        FALSE
## 4        ITe10 0.82 0.79 0.87 0.26        FALSE
## 5      AvC_MLL 0.83 0.78 0.89 0.86         TRUE
## 6 Itarget1_MPA 0.77 0.58 0.92 0.99         TRUE
## 7      FMSYref 0.84 0.91 0.67 0.99        FALSE
## 8        NFref 0.00 0.00 1.00 1.00        FALSE
```

8.2 Wormplot

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

```
wormplot(myMSE_200)
```



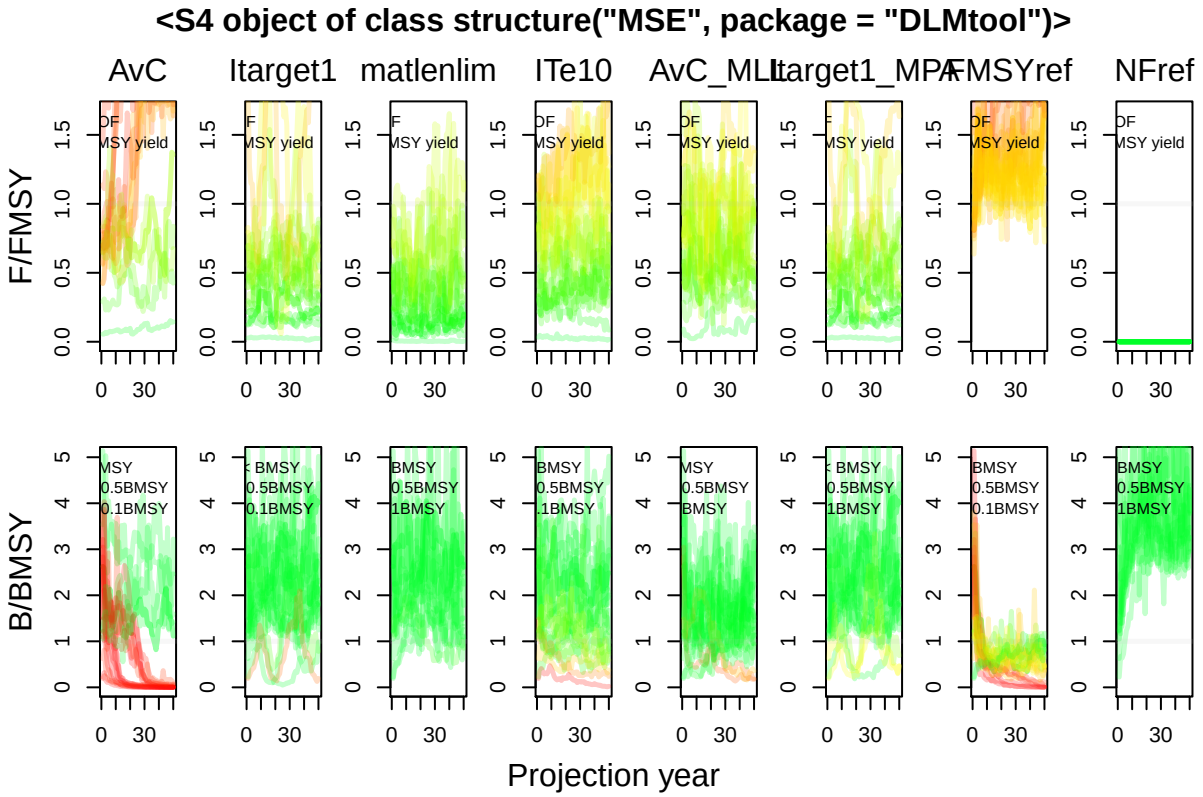
The arguments to the `wormplot` function allow you to choose the reference level for the biomass relative to B_{MSY}, as well as the upper and lower bounds of the colored bands.

8.3 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:

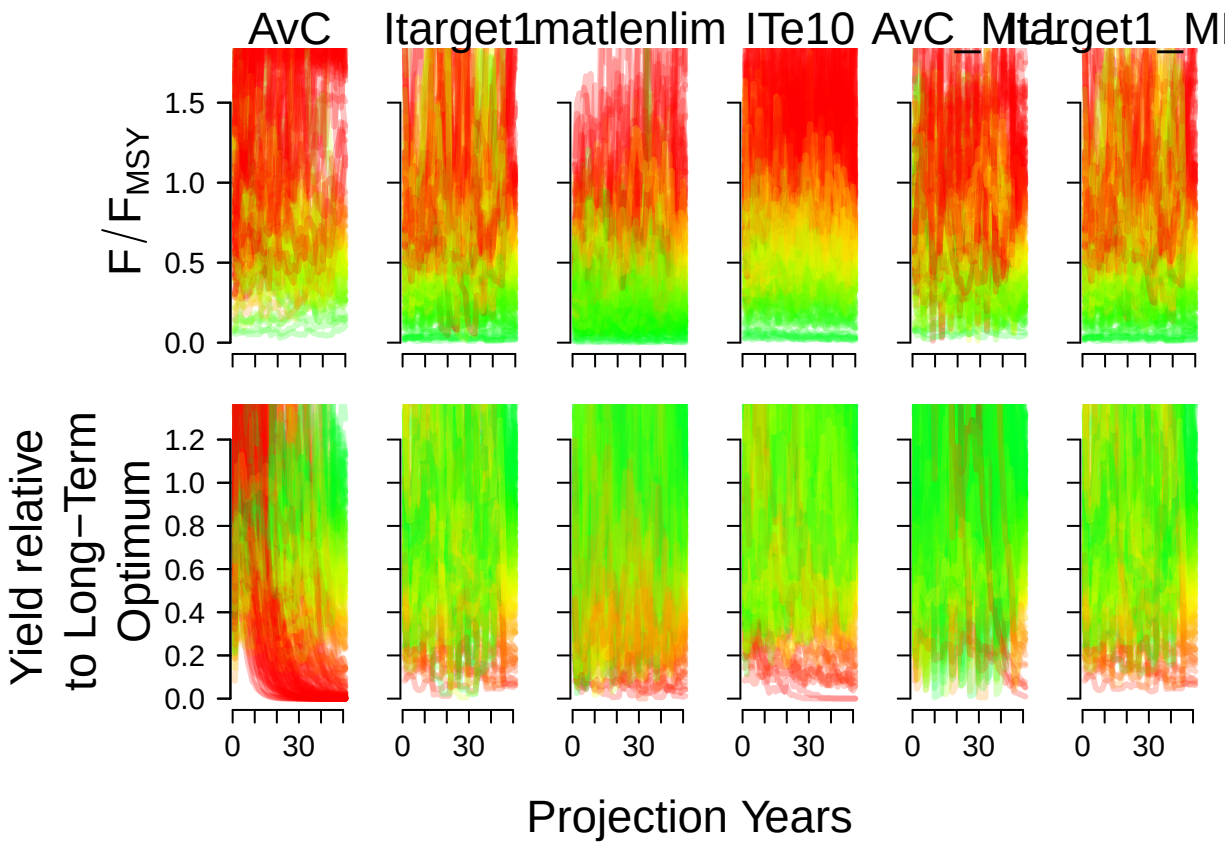
```
Pplot(myMSE_200)
```



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(myMSE_200, YVar=c("B_BMSY", "F_FMSY", "Yield"))
```

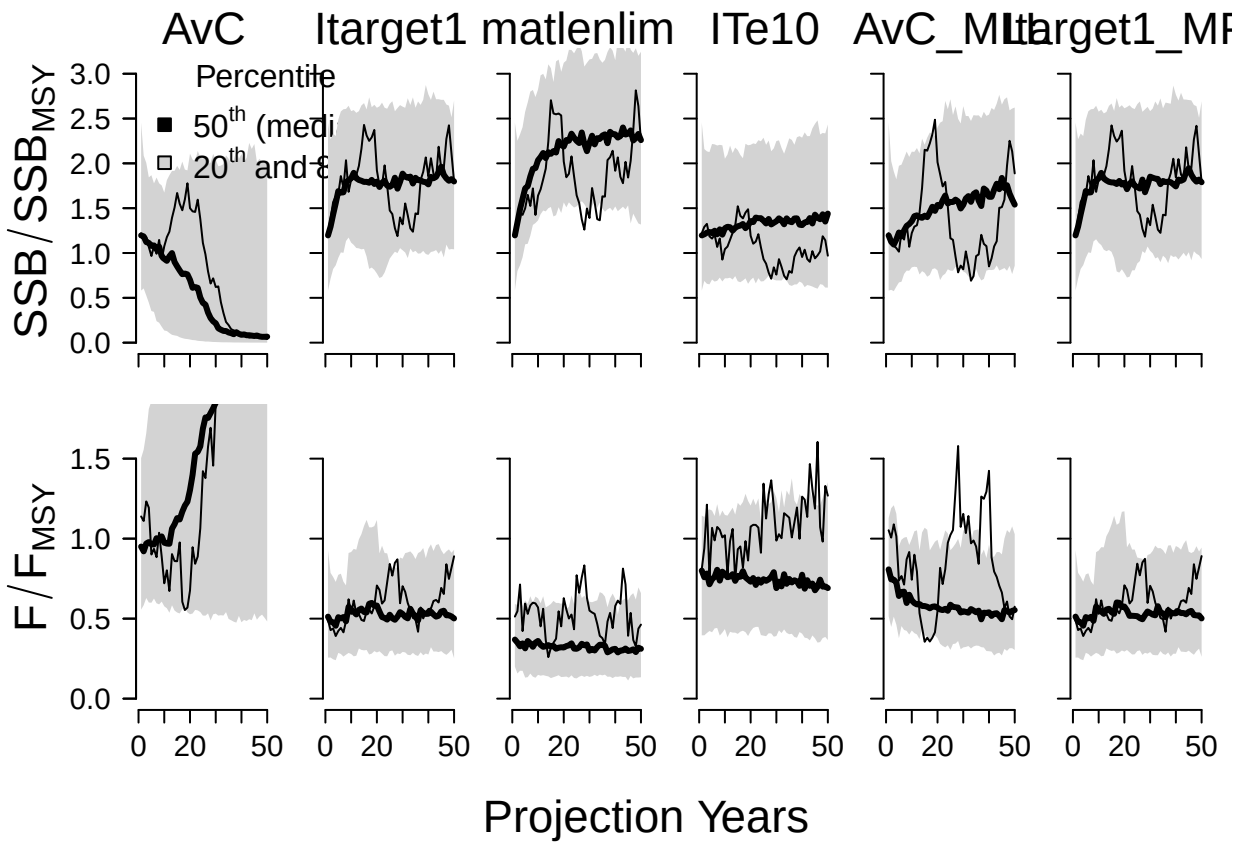
```
## MSE object has more than 6 MPs. Plotting the first 6
```



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

```
Pplot2(myMSE_200, traj="quant", quants=c(0.2, 0.8))
```

```
## MSE object has more than 6 MPs. Plotting the first 6
```

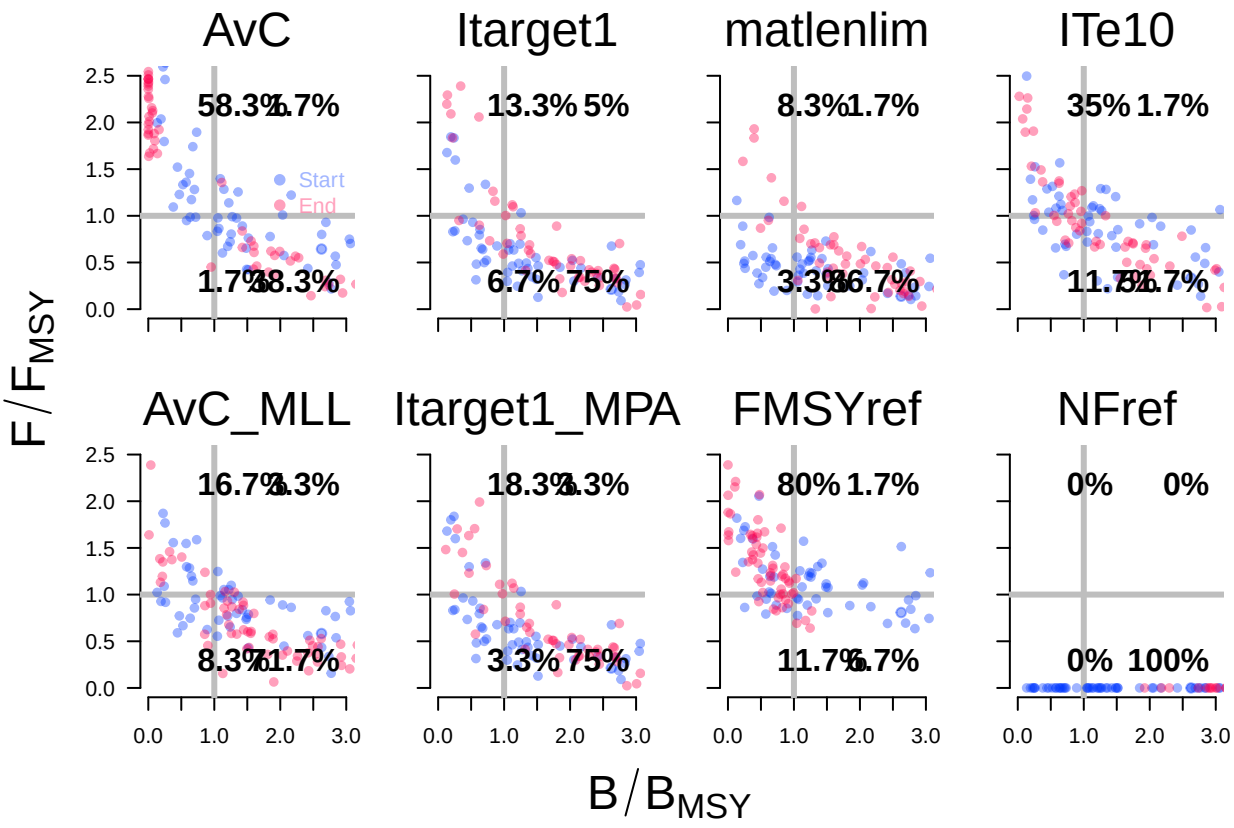


Details on additional controls for the `Pplot` and `Pplot2` functions can be found in the help documentation associated with this function.

8.4 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(myMSE_200)
```

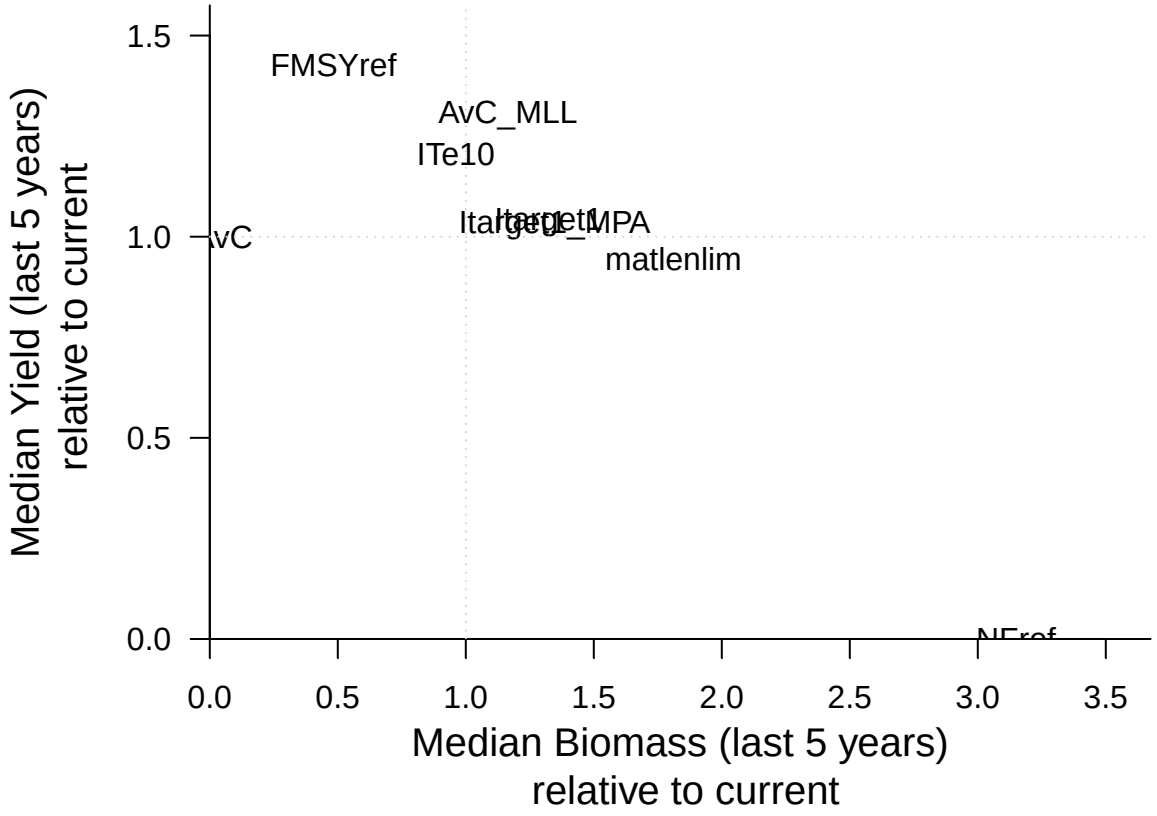


8.5 Compare to Current Conditions

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(myMSE_200, ShowLabs=TRUE)
```

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


Median Yield (last 5 years)
relative to current
1.5
1.0
0.5
0.0
FMSYref
AvC_MLL
ITe10
vC
matlenlim
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Median Biomass (last 5 years)
relative to current

Chapter 9

Parallel Processing

Parallel processing increases the speed of running the MSE in DLMtool significantly. The use of parallel processing in DLMtool has changed slightly from previous versions of the package.

By default the MSE runs without using parallel processing. We recommend running a few test runs of your MSE with a low number of simulations and without parallel processing. Once you are satisfied the model is running correctly for your operating model, you can increase the number of simulations and use parallel processing.

9.1 Setting up Parallel Processing

The `setup` function is used to set up parallel processing.

```
setup()
```

```
##
## Stopping cluster
```

```
## snowfall 1.84-6.1 initialized (using snow 0.4-2): parallel execution on 4 CPUs.
```

```
## Library DLMtool loaded.
```

```
## Library DLMtool loaded in cluster.
```

By default the `setup` function initializes 4 processors as we have found this to be the most efficient for most systems. You can change the number of processors by specifying the `cpu` argument, e.g., `setup(cpu=6)`.

See Determining Optimal Number of Processors for more details on calculating the optimal number of processors to use on your system.

9.2 Running MSE with Parallel Processing

Use the `parallel=TRUE` argument in `runMSE` to use parallel processing. Note that you must run `setup()` first.

You will notice that the usual update messages are not printed to the console when parallel processing is used. This is why it is important to initially test your MSE with a small number of simulations without parallel processing.

```
myMSE_200P <- runMSE(myOM, parallel = TRUE)
```

```
## Running MSE in parallel on 4 processors
```

```
## MSE completed
```

Parallel processing can increase the speed of running the MSE considerably. For example, although in this demonstration we are only running a low number of simulations (`nsim=48`), run time decreased from 2 to 31 seconds when using parallel processing on 4 processors.

9.3 Determining Optimal Number of Processors

The `optCPU` function can be used to evaluate the relationship between number of processors and run time:

```
optCPU()
```

```
## Running MSE with 96 simulations and 1 of 8 cpus
```

```
##
## Stopping cluster
```

```
## Running MSE with 96 simulations and 2 of 8 cpus
```

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

```
## Library DLMtool loaded.
```

```
## Library DLMtool loaded in cluster.
```

```
## MSE completed
```

```
## Running MSE with 96 simulations and 3 of 8 cpus
```

```
##
## Stopping cluster
```

```
## snowfall 1.84-6.1 initialized (using snow 0.4-2): parallel execution on 3 CPUs.
```

```
## Library DLMtool loaded.
```

```
## Library DLMtool loaded in cluster.
```

```
## MSE completed
```

```
## Running MSE with 96 simulations and 4 of 8 cpus
```

```
##
## Stopping cluster
```

```
## snowfall 1.84-6.1 initialized (using snow 0.4-2): parallel execution on 4 CPUs.
```

```
## Library DLMtool loaded.
```

```
## Library DLMtool loaded in cluster.
```

```
## MSE completed
```

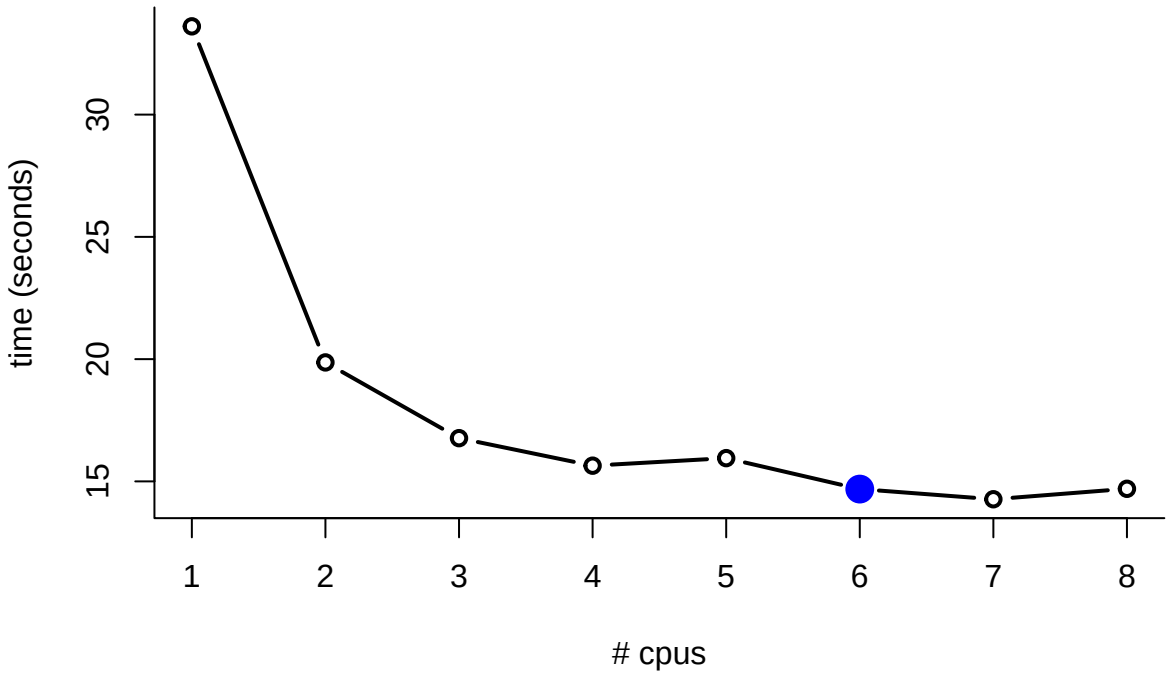
```
## Running MSE with 96 simulations and 5 of 8 cpus
```

```
##
## Stopping cluster
```

```
## snowfall 1.84-6.1 initialized (using snow 0.4-2): parallel execution on 5 CPUs.
```

```
## Library DLMtool loaded.
```

```
## Library DLMtool loaded in cluster.

## MSE completed

## Running MSE with 96 simulations and 6 of 8 cpus

##
## Stopping cluster

## snowfall 1.84-6.1 initialized (using snow 0.4-2): parallel execution on 6 CPUs.

## Library DLMtool loaded.

## Library DLMtool loaded in cluster.

## MSE completed

## Running MSE with 96 simulations and 7 of 8 cpus

##
## Stopping cluster

## snowfall 1.84-6.1 initialized (using snow 0.4-2): parallel execution on 7 CPUs.

## Library DLMtool loaded.

## Library DLMtool loaded in cluster.

## MSE completed

## Running MSE with 96 simulations and 8 of 8 cpus

##
## Stopping cluster

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

## Library DLMtool loaded.

## Library DLMtool loaded in cluster.

## MSE completed
```




```
##   ncpu     time
## 1    1 33.60998
## 2    2 19.86593
## 3    3 16.76860
## 4    4 15.64286
## 5    5 15.95117
## 6    6 14.68545
## 7    7 14.27019
## 8    8 14.69693
```

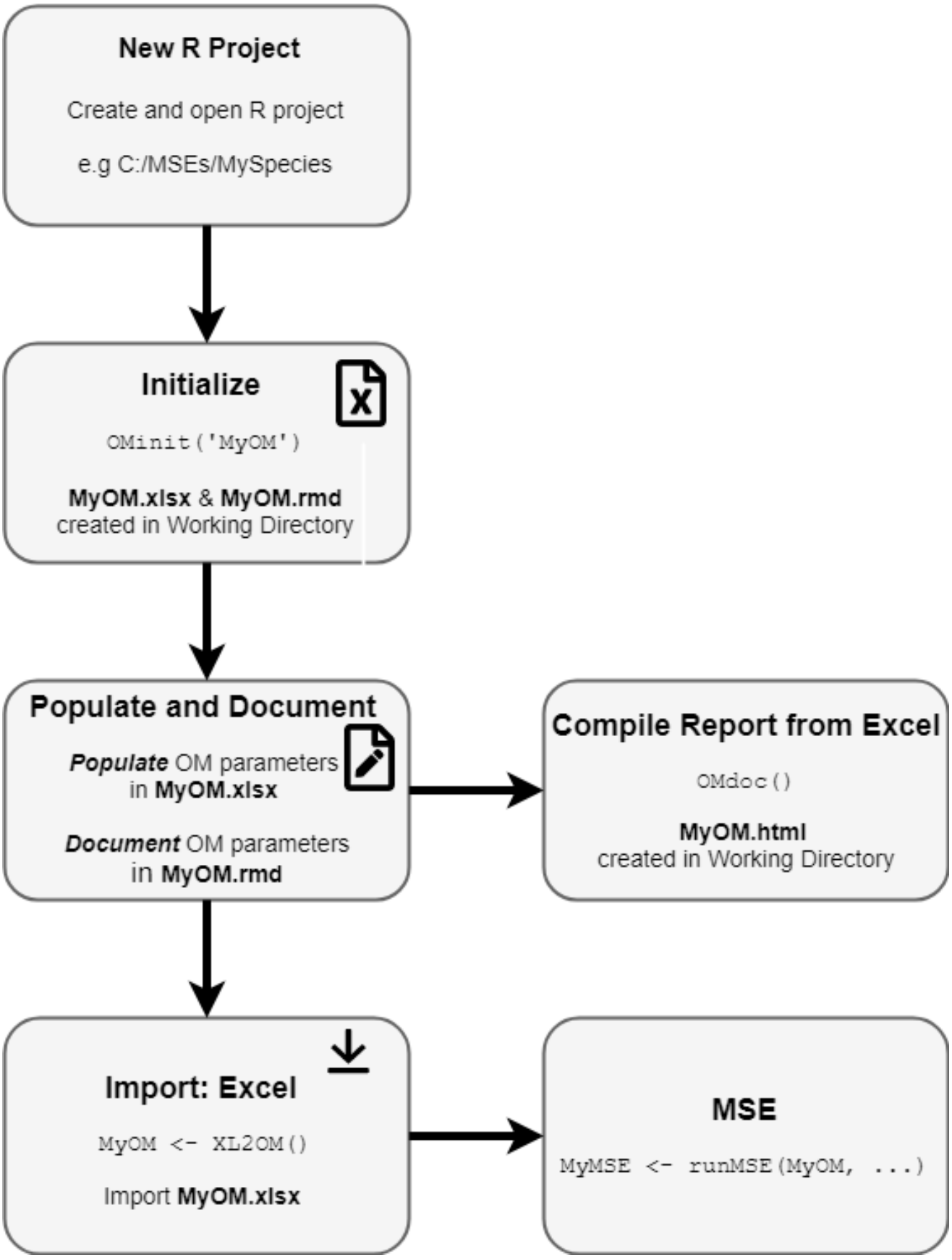
Creating an Operating Model

Chapter 10

Creating a New Operating Model

10.1 An Example WorkFlow

The figure below shows our recommended workflow creating a new Operating Model (OM) in DLMtool.


New R Project
Create and open R project
e.g C:/MSEs/MySpecies
Initialize
OMinit('MyOM')
MyOM.xlsx & MyOM.rmd
created in Working Directory
Populate and Document
Populate OM parameters
in MyOM.xlsx
Document OM parameters
in MyOM.rmd
Compile Report from Excel
OMdoc()
MyOM.html
created in Working Directory
Import: Excel
MyOM <- XL2OM()
Import MyOM.xlsx
MSE
MyMSE <- runMSE(MyOM, ...)

10.2 Create a New Project

We recommend creating a new directory for each OM. Each new R session should start by setting the working directory to this location. One of the easiest ways to do this is to create a new project in RStudio (File > New Project) and open this each time you revisit the analysis.

Alternatively, you can set the working directory with RStudio (Session > Set Working Directory) or directly in the R console, for example:

```
setwd("C:/MSE/MyOM")
```

10.3 Initialize a New OM

The `OMinit` function is used to create a blank OM spreadsheet and a skeleton OM documentation file in the working directory. This is only required the first time a new OM is created.

The `OMinit` function requires one argument, a name for the OM, and wil create two files in the working directory. For example `OMinit('MyOM')` will create **MyOM.xlsx** and **MyOM.rmd** in the working directory.

MyOM.xlsx is a spreadsheet with sheets corresponding to the components of an OM: **Stock**, **Fleet**, **Obs**, and **Imp**, and **OM** worksheets. The first column in each sheet is populated with the names of the slots of the respective objects (Stock, Fleet, etc) and all slots are empty (except the OM sheet which has default values).

The filled grey cells represent optional parameters - these are not required to run the MSE but may be used to further customized the OM (e.g age-dependant M).

Values are required for all other parameters.

The **MyOM.rmd** file can be opened in any text editor or RStudio, and contains a skeleton for the OM documentation.

The `OMinit` function also creates several folders in the working directory: data, docs, images, and robustness. These sub-directories can be used to store data, documents, images, and other information that is reference in the OM Report.

10.3.1 Using Templates

> **Note:** This feature requires additional software and may not be available on all systems. Specifically, it requires a zip application on the system PATH. Possibly the easiest way for this is to install Rtools on your system. However, please note that this feature is not required to use DLMtool.

Some users may wish to build an operating model based on other pre-existing OM, Stock, Fleet, Obs, or Imp objects.

For example, `OMinit('Albacore2', Albacore)` will result in a **Albacore2.xlsx** file being created with the **Stock** sheet populated with the values from the **Albacore** Stock object.

Other examples:

```
OMinit('StockAndFleet', Albacore, Generic_FlatE) # using existing Stock and Fleet objects

OMinit('ObsAndImp', Generic_Obs, Perfect_Imp) # using existing Obs and Imp objects

OMinit('BorrowOM', OtherOM) # using an existing OM
```

10.3.2 An Example

In this example we are going to create an OM called 'MyOM' using existing OM objects:

```
OMinit('MyOM', Albacore, Generic_FlatE, Imprecise_Unbiased, Perfect_Imp)
```

We did this so that we can demonstrate the populated Excel and RMarkdown Files.

To create a blank OM called 'MyOM' you would simply write:

```
OMinit('MyOM')
```

10.4 Populate and Document OM

Next we open Excel workbook and populate the OM.

Because we used templates our workbook is already populated. You can download and inspect the populated OM workbook we created in the previous step.

To assist in documenting the rationale for the OM parameters, we recommended adding a short but informative description or rationale for the OM values in the RMarkdown file while the OM Excel file is being populated (open the RMarkdown file and edit with any text editor or RStudio).

Once complete, the RMarkdown file can be compiled into a HTML report and provides a complete documentation for the OM.

The RMarkdown file we created earlier can be accessed here and opened in RStudio.

You will see that the RMarkdown file has a series of headings (marked by #, ##, and ### for first, second and third level respectively) followed by some text, in this case default text is mainly instructions on how to fill the document.

The instruction text should be deleted and replaced with the relevant information for your operating model. For example, below the line "# Introduction" you would delete the instructions and provide a brief introduction to your fishery and the purpose of the OM and MSE.

It is important not to delete any of the headings.

After the Introduction section, the document has four first level headings corresponding to the Stock, Fleet, Obs and Imp components of the operating model.

Each section has a series of second level headings (e.g., ## M) which correspond to the slots of that object. In this example, the text below these headings indicates that this parameter was borrowed from another object (e.g 'Borrowd from: Albacore').

If the parameters in the OM workbook are modified from those borrowed from the existing object (in this case 'Albacore'), you would delete this text and replace it will your own justification.

If you did not initialize your OM using existing objects as templates, it will say something like 'No justification provided'.

It is not neccessary to include the actual values in the justification text. The RMarkdown file containing the justifications/rationale will be compiled together with the OM Excel workbook containing the OM parameter values into a OM Report that contains both the justification text, the OM values, and a series of plots to visualize the OM parameters and properties.

The OM documentation file should be updated whenever values in the OM are changed.

10.5 Compile the OM Report

Once the OM has been specified in the spreadsheet and documented in the RMarkdown file, it can be compiled into a OM Report using the `OMdoc` function.

The `OMdoc` function

```
OMdoc('MyOM')
```

In most cases it is not neccessary to provide the name of the RMarkdown file to `OMdoc`. By default the `OMdoc` function will look for a file with the extension '.rmd' in working directory. For example, if the Excel file is named *MyOM* then `OMdoc` will look for *MyOM.rmd*, which is default name created by `OMinit`.

Additionally, if there is only one *xlsx* file in the working directory the name of the OM is not required, i.e., `OMdoc()`.

The resulting MyOM.html can opened in any web browser. Because we have not replaced any of the default text in the RMarkdown file, the resulting OM Report contains the same text. In your case, this default text should be replaced with information relevant to your OM.

It is also be possible to compile the OM report into a pdf using `OMdoc('MyOM', output="pdf_document")`, although this may require the installation of additional software on your system.

10.6 Import the OM into R

The OM can be imported from the Excel file using the `XL2OM` function.

For example, to import the example OM created in the previous section:

```
OM <- XL2OM('MyOM')
```

The OM is now ready to be used for analysis, for example:

```
# Plot the OM
plot(OM)

# Run an MSE using default MPs
MyMSE <- runMSE(OM)
```

10.7 Documenting an Existing OM

To document existing OMs that don't use the Excel workbook the `OMinit` function can be used to create just the RMarkdown documentation file in the working directory.

The `OMdoc` function can be used to generate an OM report directly from an OM object and a RMarkdown file. In this case it is necessary to provide the name of the Rmarkdown file to `OMdoc`.

For example, here we create an OM using existing objects from DLMtool, generate the RMarkdown documentation skeleton (only required once), and compile the OM report:

```
BlueSharkOM <- new('OM', Blue_shark, Generic_Fleet, Imprecise_Biased, Perfect_Imp)
OMinit('BlueSharkOM', files='rmd', BlueSharkOM)

# - Enter OM details in BlueSharkOM.rmd -
OMdoc(BlueSharkOM, 'BlueSharkOM.rmd')
```

The same process is used if you are using the `cpars` feature to provide custom parameters to the MSE (see Custom Parameters section for more details).

Chapter 11

Operating Model Library

We are in the process of developing an online library of DLMtool Operating Models.

This library includes the OM Report, the OM Excel workbook, and the OM R Data file for many of the fisheries where DLMtool OMs have been built. The idea behind the OM library is to develop a resource for DLMtool users to learn from other applications as well as to provide OM templates which users can borrow and modify to suit their own fishery.

The OM library is still being developed and we are continuing to add OMs that we have constructed. If you have built a DLMtool OM and are happy to make it public, please contact us through the website or email us directly, we would love to include it on our website.

Interpreting MSE Results

Chapter 12

Examining the MSE object

In this chapter we will examine the MSE object in more detail. First we will run an MSE so that we have an MSE object to work with:

12.1 Run an MSE

We create an OM based on the Blue Shark stock object and other built-in objects:

```
OM <- new('OM', Blue_shark, Generic_Fleet, Imprecise_Biased, Perfect_Imp, nsim=200)
```

Note that we have increased the number of simulations from the default 48 to 200:

```
OM@nsim
```

```
## [1] 200
```

Let's choose an arbitrary set of MPs:

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

Set up parallel processing:

```
setup()
```

```
## Library DLMtool loaded.
```

And run the MSE using parallel processing and save the output to an object called `BSharkMSE`:

```
BSharkMSE <- runMSE(OM, MPs, parallel = TRUE)
```

```
## Running MSE in parallel on 4 processors
```

```
## MSE completed
```

12.2 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"
## [25] "Misc"
```

As you can see, `MSE` objects contain all of the information from the MSE, stored in 25 slots.

12.2.1 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_Imp"
```

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`).

12.2.2 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"       "Ksd"          "ageM"
## [25] "L5"           "LFS"          "Vmaxlen"      "LFC"
## [29] "OFLreal"      "Spat_targ"    "Size_area_1"  "Frac_area_1"
## [33] "Prob_staying" "AC"           "L50"          "L95"
## [37] "B0"           "N0"           "SSB0"         "BMSY_B0"
## [41] "TACSD"        "TACFrac"      "TAESD"        "TAEFrac"
## [45] "SizeLimSD"    "SizeLimFrac"  "Blow"         "BMSY"
## [49] "SSBMSY"       "Mexp"         "Fdisc"        "LR5"
## [53] "LFR"          "Rmaxlen"      "DR"
```

If you use the `dim` function to report the dimensions of the `OM` data frame, you'll notice that there are 55 columns, corresponding to the 55 parameters in the Operating Model, and 200 rows, each corresponding to a single simulation of the MSE.

12.2.3 The `Obs` Slot

The `Obs` slot contains another data frame, this one with 26 columns corresponding to the values drawn from the Observation model:

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

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

```
## [26] "BMSY_B0bias"
```

The `Obs` data frame also has 200 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.

12.2.4 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`):

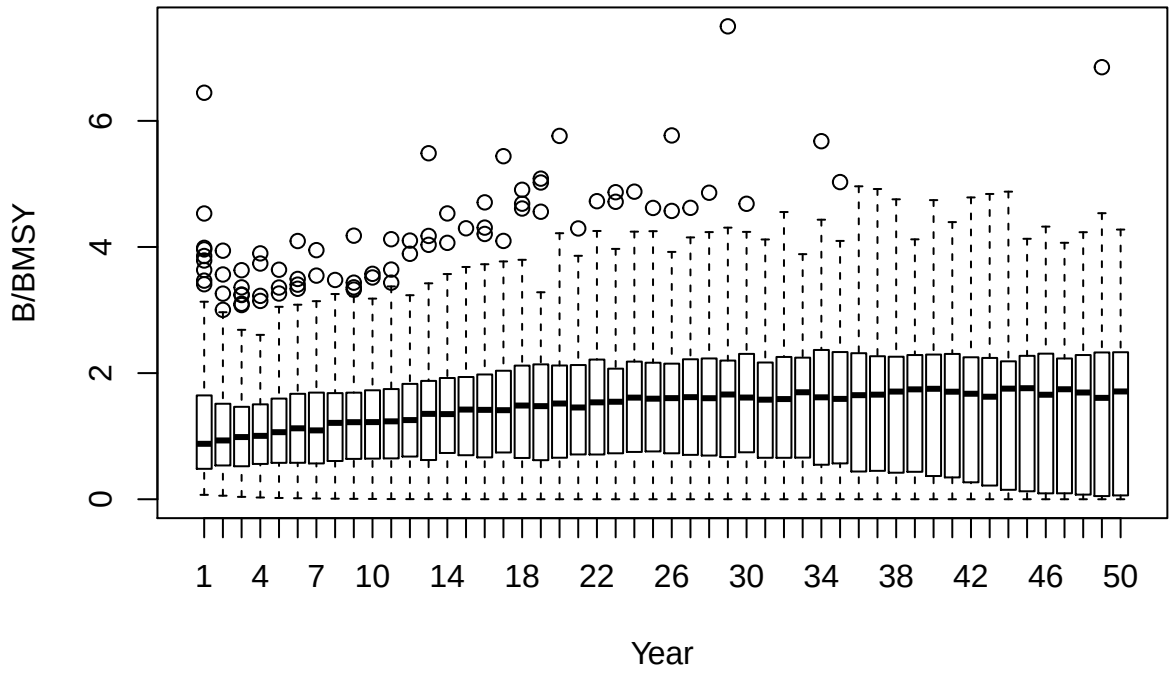
```
dim(BSharkMSE@B_BMSY)
```

```
## [1] 200   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 0.88 in the first year to 1.71 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.

12.2.5 The `B`, `FM`, `C` and `TAC` 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.

The TAC slot contains the TAC recommendation for each simulation, MP, and projection year. In cases where a TAC was not set (e.g for a size limit), the value will be NA. The values in TAC may be different to those in the catch (C) slot due to implementation error of the total catch limit.

12.2.6 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.

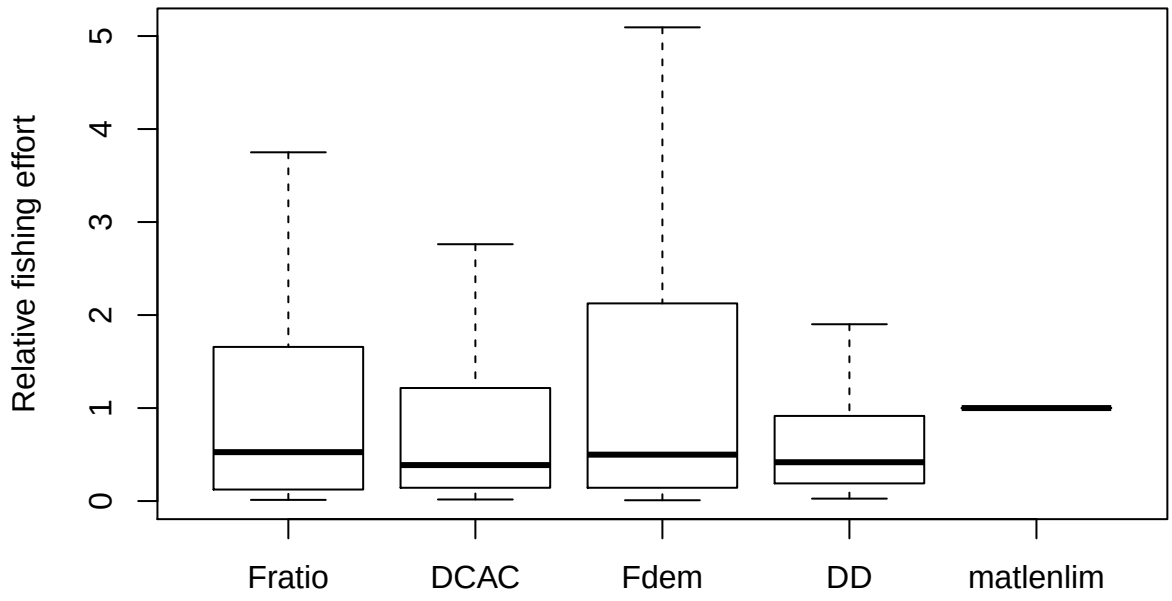
The `apply` function can be used to aggregate these data over the age-classes or spatial areas.

12.2.7 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:

```
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 0.39 to 0.53 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.

Chapter 13

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.

13.1 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.

13.2 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).

13.2.1 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).

13.3 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.

13.3.1 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).

13.3.2 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.

13.3.3 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.

13.3.4 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)
```

```
## Calculating Performance Metrics
```

```
##                                    Performance.Metrics
## 1                   Average Annual Variability in Yield
## 2  Average Long-Term Yield relative to Reference Yield
## 3          Probability Spawning Biomass above 10% BMSY
## 4                  Probability Spawning Biomass > BMSY
## 5          Probability Spawning Biomass above 50% BMSY
## 6                                 Probability F < FMSY
## 7 Average Short-Term Yield relative to Reference Yield
## 8                     Yield relative to Reference Yield
##
##
## Probability:
##          MP AAVY  LTY  P10 P100  P50  POF  STY Yield
## 1    Fratio 0.86 0.48 0.82 0.49 0.66 0.59 0.61  0.72
## 2      DCAC 0.96 0.48 0.83 0.65 0.76 0.70 0.67  0.67
## 3      Fdem 0.78 0.47 0.76 0.43 0.59 0.51 0.62  0.71
## 4        DD 0.96 0.80 0.92 0.53 0.77 0.67 0.68  0.86
## 5 matlenlim 0.34 0.82 0.99 0.69 0.89 0.78 0.71  0.95
```

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

13.3.5 Plotting MSE Results

The performance of the MPs can also be examined visually. See the Plotting the MSE Results section for examples on DLMtool plotting functions for the MSE object.

Advanced users may wish to develop their own plotting and summary functions. See the Custom Performance Metrics section for more details on this.

Chapter 14

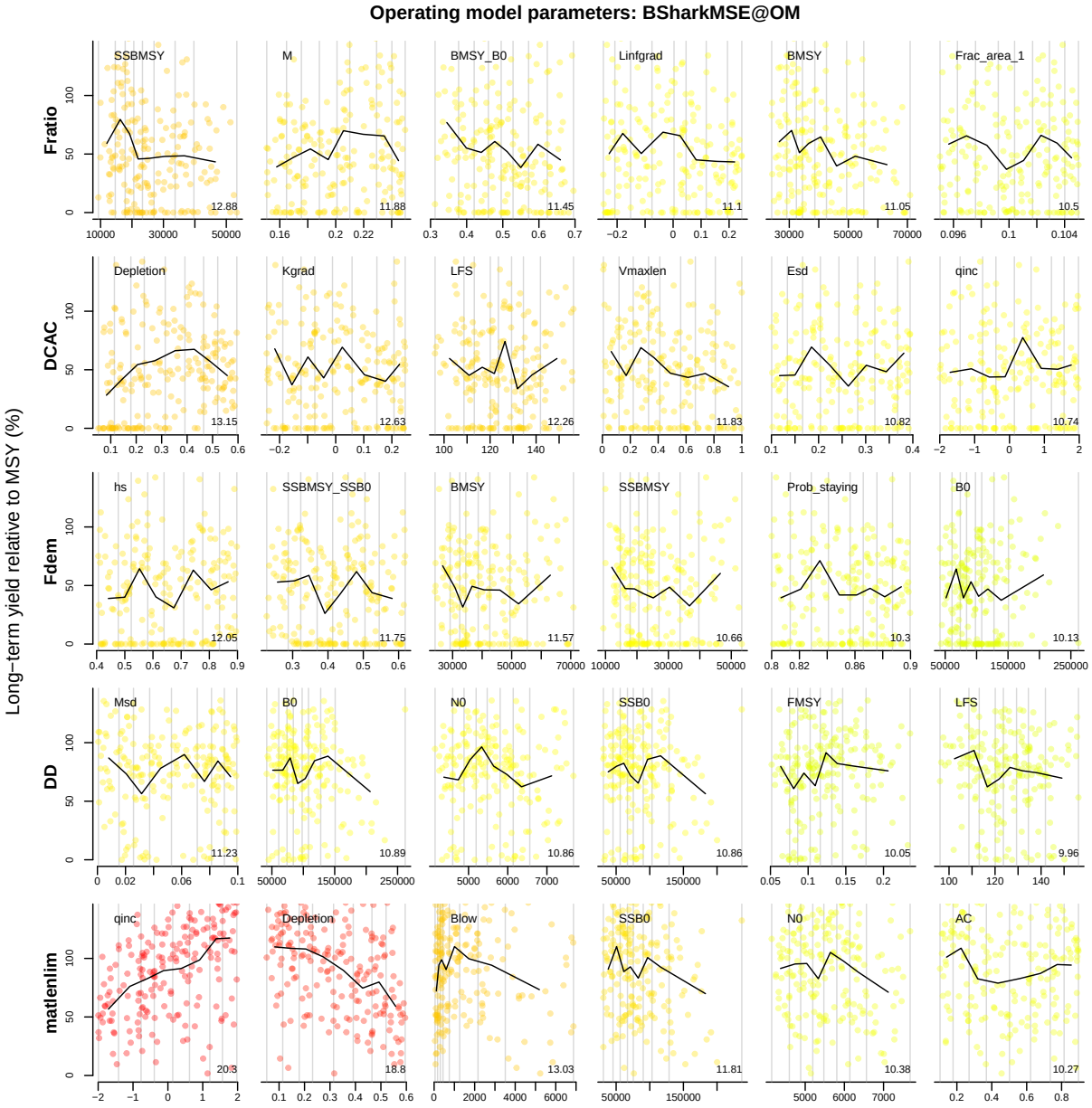
Value of Information

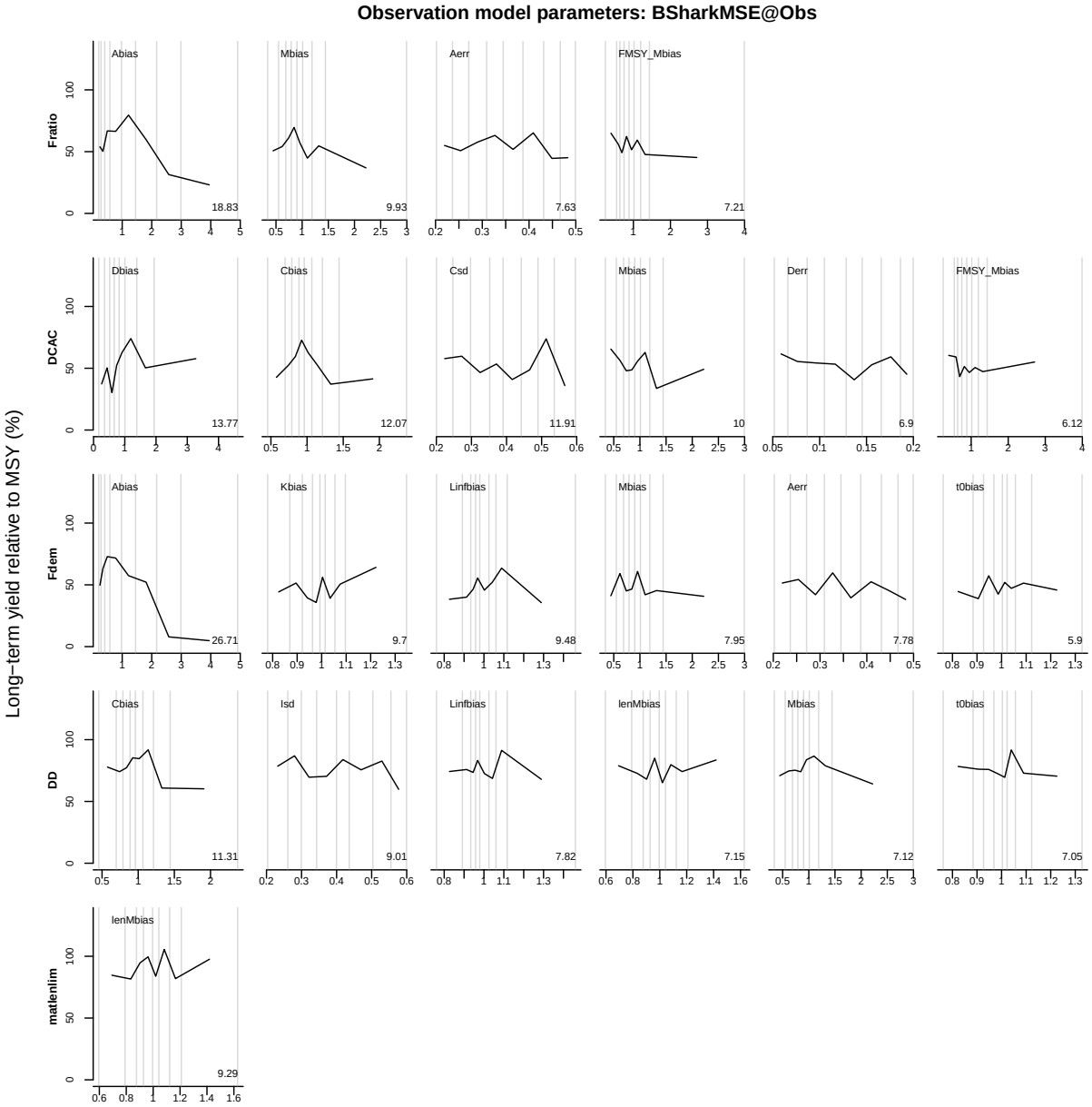
The Value of Information (VOI) functions have been designed to explore the sensitivity of the performance of the Management Procedures to variability in the observation processes and operating model parameters.

There are several VOI functions in DLMtool.

The `VOI` function generates two plots, one corresponding to the operating model parameters, and the other to the observation model parameters, showing the gradient in long-term yield with respect to the individual parameters:

```
VOI(BSharkMSE)
```


Operating model parameters: BSharkMSE@OM
Long−term yield relative to MSY (%)
Fratio
SSBMSY
M
BMSY_B0
Linfgrad
BMSY
Frac_area_1
DCAC
Depletion
Kgrad
LFS
Vmaxlen
Esd
qinc
Fdem
hs
SSBMSY_SSB0
BMSY
SSBMSY
Prob_staying
B0
DD
Msd
B0
N0
SSB0
FMSY
LFS
matlenlim
qinc
Depletion
Blow
SSB0
N0
AC

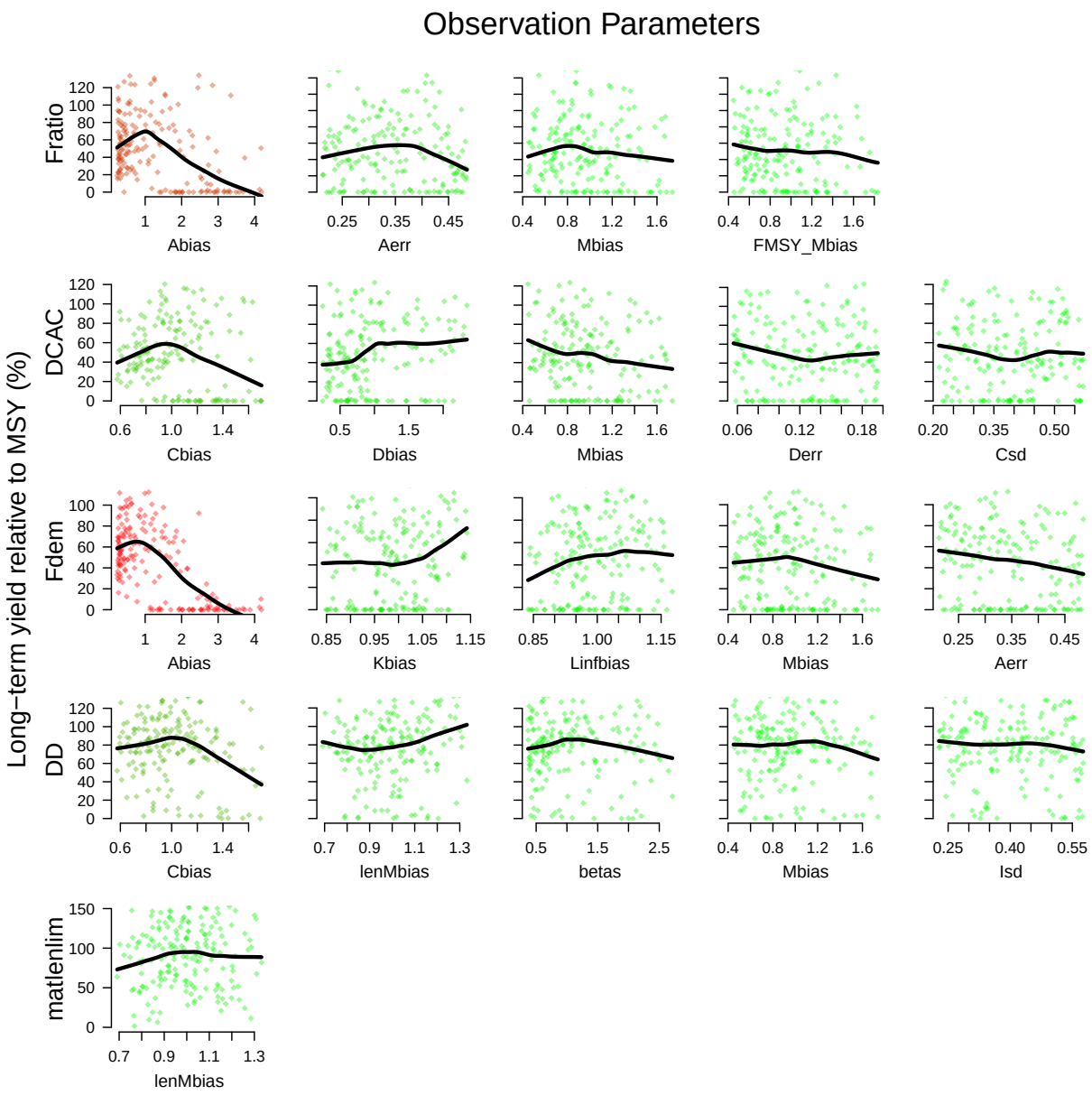



```
## [[1]]
##           MP         1           2       3        4           5
## 1     Fratio    SSBMSY           M BMSY_B0 Linfgrad        BMSY
## 2                12.88       11.88   11.45     11.1       11.05
## 3       DCAC Depletion       Kgrad     LFS  Vmaxlen         Esd
## 4                13.15       12.63   12.26     11.83      10.82
## 5       Fdem        hs SSBMSY_SSB0    BMSY   SSBMSY Prob_staying
## 6                12.05       11.75   11.57     10.66       10.3
## 7         DD       Msd          B0      N0     SSB0       FMSY
## 8                11.23       10.89   10.86     10.86      10.05
## 9  matlenlim      qinc   Depletion    Blow     SSB0         N0
## 10               20.3        18.8   13.03     11.81      10.38
##              6
## 1  Frac_area_1
```

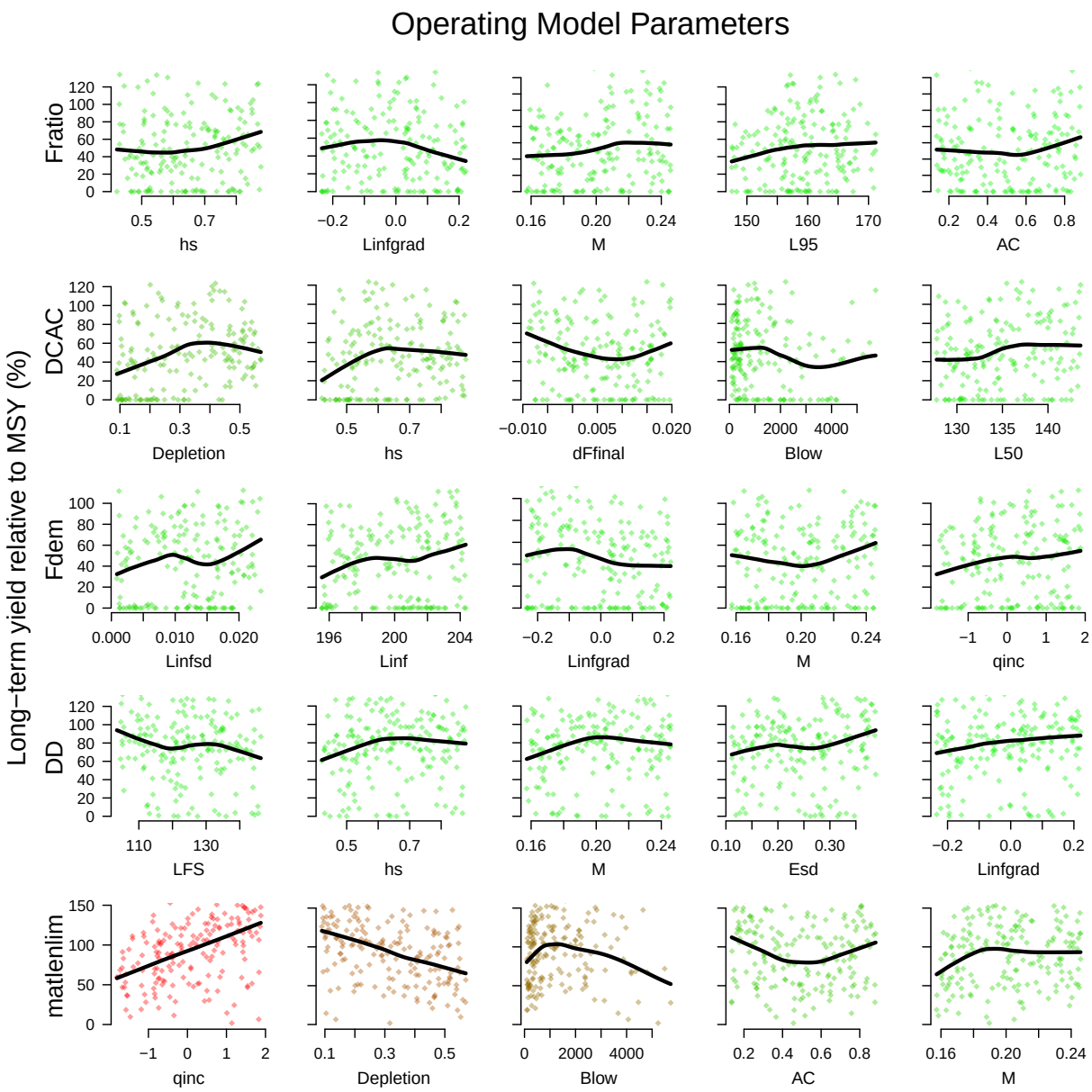
```
## 2         10.5
## 3         qinc
## 4        10.74
## 5           B0
## 6        10.13
## 7          LFS
## 8         9.96
## 9           AC
## 10       10.27
##
## [[2]]
##           MP        1     2        3          4     5          6
## 1     Fratio    Abias Mbias     Aerr FMSY_Mbias
## 2               18.83  9.93     7.63       7.21
## 3       DCAC    Dbias Cbias      Csd      Mbias  Derr FMSY_Mbias
## 4               13.77 12.07    11.91         10   6.9       6.12
## 5       Fdem    Abias Kbias Linfbias      Mbias  Aerr     t0bias
## 6               26.71   9.7     9.48       7.95  7.78        5.9
## 7         DD    Cbias   Isd Linfbias   lenMbias Mbias     t0bias
## 8               11.31  9.01     7.82       7.15  7.12       7.05
## 9  matlenlim lenMbias
## 10              9.29
```

The `VOIplot` function shows something similar, but has an argument to specify either the Observation or Operating Model parameters:

```
# Observation Parameters
VOIplot(BSharkMSE, nMP=5)
```




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

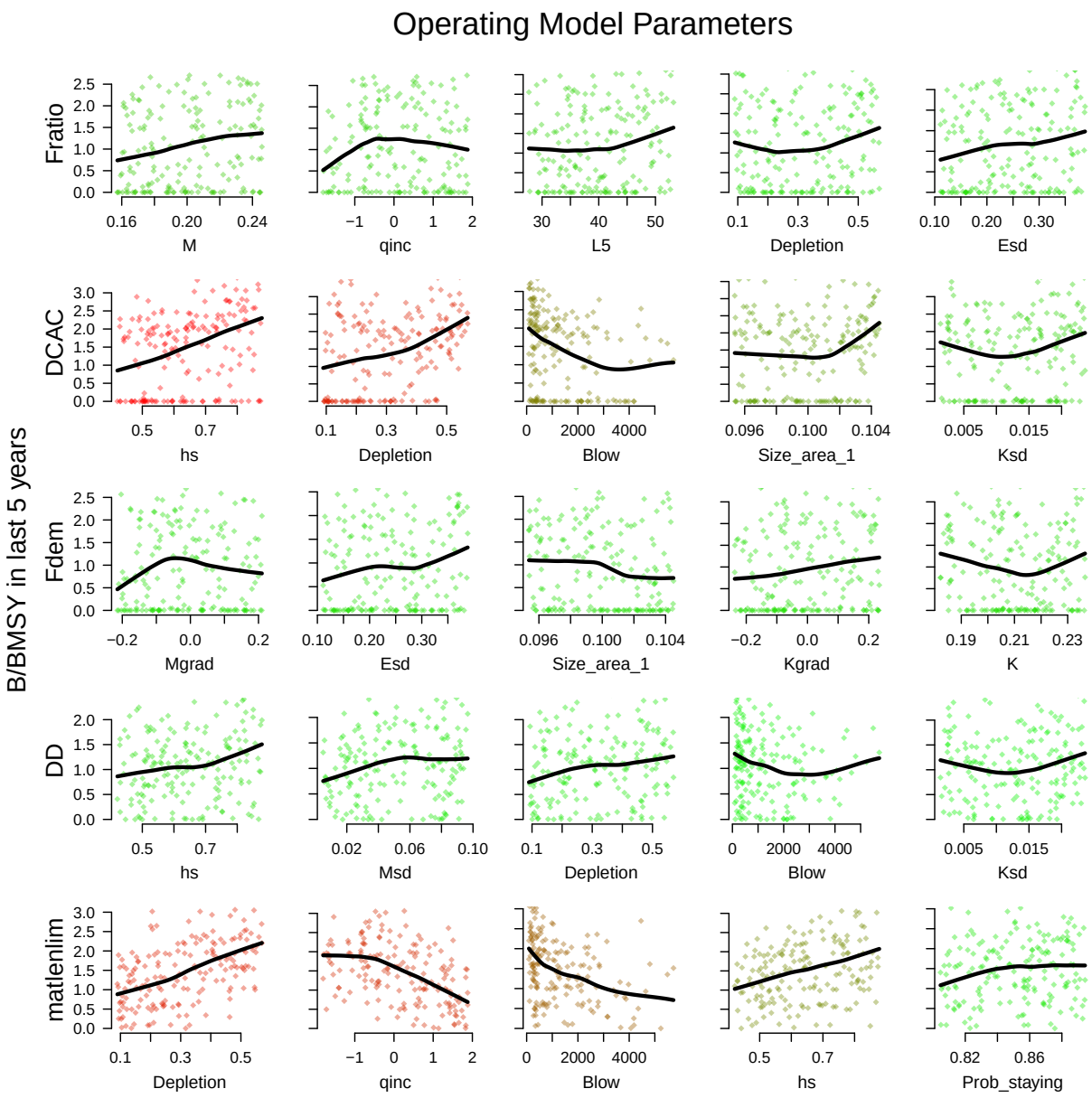


By default, the `VOIplot` function only shows the four Management Procedures with the greatest sensitivity. Here we've made it show all five methods using the `nMP` argument.

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.

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

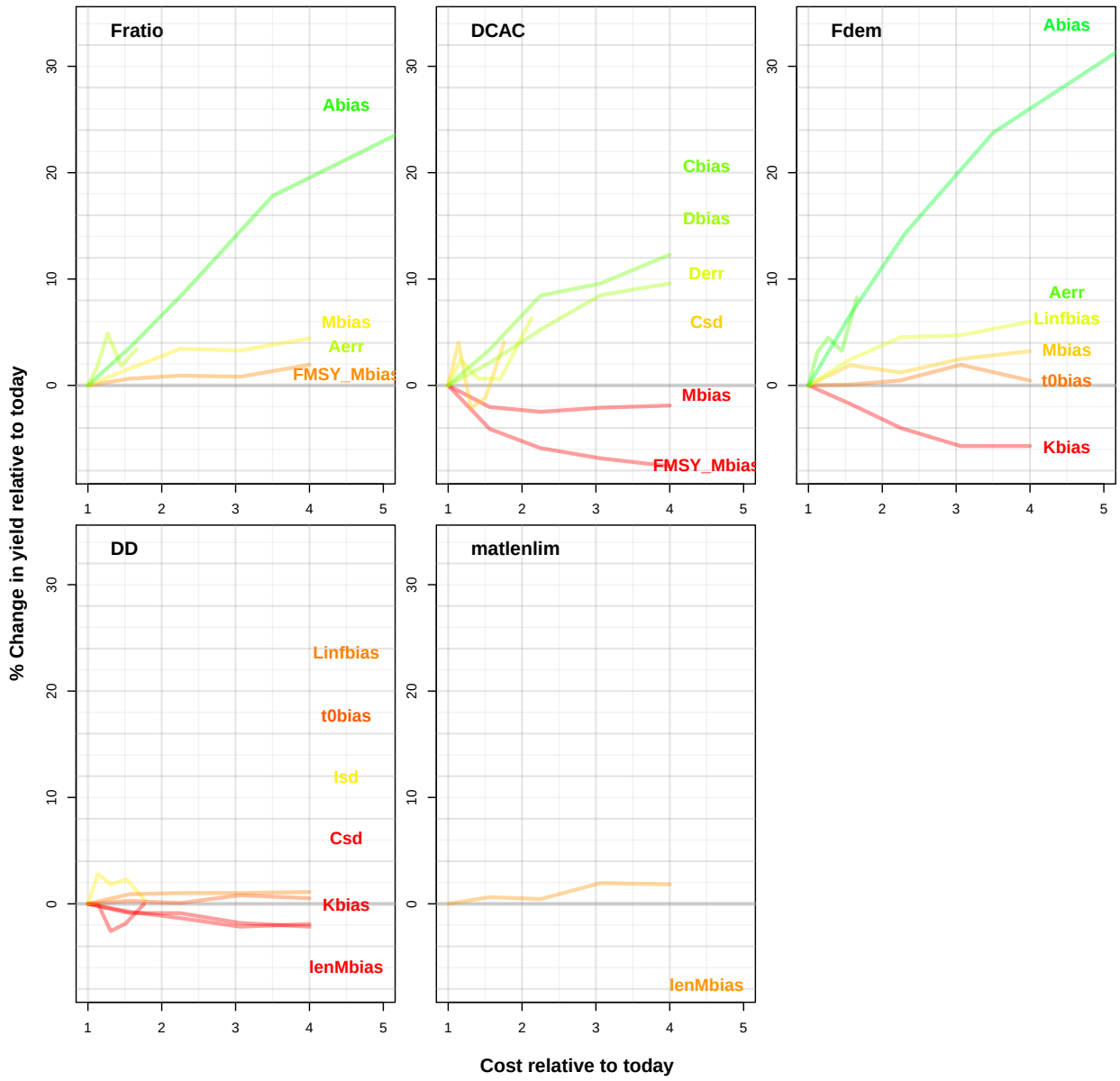
```
VOIplot(BSharkMSE, Par="OM", nMP=5, YVar="B")
```



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 `VOI2` function relates the operating model parameters and parameters of the observation model to relative yield (yield over last 5 years of projection relative to a 'best F' scenario that maximizes yield.

```
VOI2(BSharkMSE)
```




```
## [[1]]
## , , 1
##
##        [,1]     [,2]     [,3]     [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
## [1,] 1.5625 1.562500 1.589613 1.119555     NA     NA     NA     NA     NA
## [2,] 1.5625 1.140612 1.562500 1.171614 1.5625 1.5625     NA     NA     NA
## [3,] 1.5625 1.589613 1.119555 1.562500 1.5625 1.5625     NA     NA     NA
## [4,] 1.5625 1.140612 1.136572 1.487903 1.5625 1.5625 1.5625 1.5625 1.5625
## [5,] 1.5625       NA       NA       NA     NA     NA     NA     NA     NA
##      [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
```

```
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##      [,21] [,22] [,23] [,24] [,25]
## [1,]    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA
##
## , , 2
##
##      [,1]     [,2]     [,3]     [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
## [1,] 2.25 2.250000 2.308293 1.268207   NA   NA   NA   NA   NA    NA    NA
## [2,] 2.25 1.307538 2.250000 1.398430 2.25 2.25   NA   NA   NA    NA    NA
## [3,] 2.25 2.308293 1.268207 2.250000 2.25 2.25   NA   NA   NA    NA    NA
## [4,] 2.25 1.307538 1.311405 2.105272 2.25 2.25 2.25 2.25 2.25    NA    NA
## [5,] 2.25       NA       NA       NA   NA   NA   NA   NA   NA    NA    NA
##      [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22]
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##      [,23] [,24] [,25]
## [1,]    NA    NA    NA
## [2,]    NA    NA    NA
## [3,]    NA    NA    NA
## [4,]    NA    NA    NA
## [5,]    NA    NA    NA
##
## , , 3
##
##        [,1]     [,2]     [,3]     [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
## [1,] 3.0625 3.062500 3.499592 1.451277     NA     NA     NA     NA     NA
## [2,] 3.0625 1.506877 3.062500 1.707539 3.0625 3.0625     NA     NA     NA
## [3,] 3.0625 3.499592 1.451277 3.062500 3.0625 3.0625     NA     NA     NA
## [4,] 3.0625 1.506877 1.524016 3.412822 3.0625 3.0625 3.0625 3.0625 3.0625
## [5,] 3.0625       NA       NA       NA     NA     NA     NA     NA     NA
##      [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##      [,21] [,22] [,23] [,24] [,25]
## [1,]    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA
##
## , , 4
##
##      [,1]     [,2]     [,3]     [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
## [1,]    4 4.000000 5.748110 1.656126   NA   NA   NA   NA   NA    NA    NA
```

```
## [2,]    4 1.757649 4.000000 2.126836    4    4   NA   NA   NA    NA    NA
## [3,]    4 5.748110 1.656126 4.000000    4    4   NA   NA   NA    NA    NA
## [4,]    4 1.757649 1.774084 6.266443    4    4    4    4    4    NA    NA
## [5,]    4       NA       NA       NA   NA   NA   NA   NA   NA    NA    NA
##      [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22]
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##      [,23] [,24] [,25]
## [1,]    NA    NA    NA
## [2,]    NA    NA    NA
## [3,]    NA    NA    NA
## [4,]    NA    NA    NA
## [5,]    NA    NA    NA
##
##
## [[2]]
## , , 1
##
##           [,1]        [,2]     [,3]       [,4]        [,5]        [,6]
## [1,] 1.5601054  0.63110134 3.708660  1.624825          NA          NA
## [2,] 3.3789766  4.04496626 2.133689  2.450301 -2.03545411 -4.0998095
## [3,] 1.9054129  6.90069594 3.119943 -1.723752  0.07738356  2.4409120
## [4,] 2.7233727 -0.06782785 2.799963  1.205309  1.80827209 -0.8660319
## [5,] 0.6275872          NA       NA         NA          NA          NA
##           [,7]      [,8]      [,9] [,10] [,11] [,12] [,13] [,14] [,15]
## [1,]         NA        NA        NA    NA    NA    NA    NA    NA    NA
## [2,]         NA        NA        NA    NA    NA    NA    NA    NA    NA
## [3,]         NA        NA        NA    NA    NA    NA    NA    NA    NA
## [4,] -0.7413871 0.2740917 0.9016601    NA    NA    NA    NA    NA    NA
## [5,]         NA        NA        NA    NA    NA    NA    NA    NA    NA
##      [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##
## , , 2
##
##           [,1]       [,2]     [,3]       [,4]       [,5]        [,6]
## [1,] 3.4443331  0.9202632 8.779032  4.8885799         NA          NA
## [2,] 8.4446808 -2.1244629 5.241968  0.6578662 -2.4795834 -5.8912070
## [3,] 1.2096198 14.2838537 4.482447 -4.0050425  0.4771663  4.5290904
## [4,] 4.4629684 -2.5587204 1.815247  1.7395715  2.8613410 -0.8772127
## [5,] 0.4456357         NA       NA         NA         NA          NA
##          [,7]       [,8]     [,9] [,10] [,11] [,12] [,13] [,14] [,15]
## [1,]       NA         NA       NA    NA    NA    NA    NA    NA    NA
## [2,]       NA         NA       NA    NA    NA    NA    NA    NA    NA
## [3,]       NA         NA       NA    NA    NA    NA    NA    NA    NA
## [4,] -1.34093 0.07610088 1.014479    NA    NA    NA    NA    NA    NA
## [5,]       NA         NA       NA    NA    NA    NA    NA    NA    NA
```

```
##      [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##
## , , 3
##
##          [,1]       [,2]      [,3]       [,4]      [,5]      [,6]
## [1,] 3.277736  0.8191016 17.806639  1.8378369        NA        NA
## [2,] 9.585252 -1.1541685  8.487364  0.6117235 -2.098361 -6.851080
## [3,] 2.492436 23.7483193  3.247910 -5.7026925  1.939986  4.721445
## [4,] 6.634815 -1.8728754  2.302950  4.4201934  3.649993 -1.803083
## [5,] 1.950804         NA        NA         NA        NA        NA
##           [,7]      [,8]     [,9] [,10] [,11] [,12] [,13] [,14] [,15]
## [1,]        NA        NA       NA    NA    NA    NA    NA    NA    NA
## [2,]        NA        NA       NA    NA    NA    NA    NA    NA    NA
## [3,]        NA        NA       NA    NA    NA    NA    NA    NA    NA
## [4,] -2.147004 0.8120631 1.017772    NA    NA    NA    NA    NA    NA
## [5,]        NA        NA       NA    NA    NA    NA    NA    NA    NA
##      [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##
## , , 4
##
##           [,1]        [,2]       [,3]      [,4]       [,5]      [,6]
## [1,]  4.433257  1.94543869 25.5440125  3.423210         NA        NA
## [2,] 12.294601  4.03972358  9.5882164  6.330707 -1.8903521 -7.594869
## [3,]  3.221369 33.94060147  8.2631202 -5.698272  0.4557958  5.992324
## [4,]  8.655068 -0.04733839  0.3096741  8.130447  4.6465157 -2.145352
## [5,]  1.825975          NA         NA        NA         NA        NA
##           [,7]      [,8]     [,9] [,10] [,11] [,12] [,13] [,14] [,15]
## [1,]        NA        NA       NA    NA    NA    NA    NA    NA    NA
## [2,]        NA        NA       NA    NA    NA    NA    NA    NA    NA
## [3,]        NA        NA       NA    NA    NA    NA    NA    NA    NA
## [4,] -1.901028 0.5188852 1.115311    NA    NA    NA    NA    NA    NA
## [5,]        NA        NA       NA    NA    NA    NA    NA    NA    NA
##      [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##
##
## [[3]]
## , , 1
##
##           [,1]      [,2]      [,3]      [,4]       [,5]       [,6]
```

```
## [1,] 0.3450813 0.3985905 0.9877312 0.3295544         NA         NA
## [2,] 0.2624556 0.3691106 0.5727031 0.1167828 0.34508128 0.39859050
## [3,] 0.3450813 0.9877312 0.3295544 0.0778791 0.08208062 0.08589388
## [4,] 0.2624556 0.3691106 0.3764067 0.6062044 0.34508128 0.15045590
## [5,] 0.1504559        NA        NA        NA         NA         NA
##           [,7]       [,8]       [,9] [,10] [,11] [,12] [,13] [,14] [,15]
## [1,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
## [2,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
## [3,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
## [4,] 0.0778791 0.08208062 0.08589388    NA    NA    NA    NA    NA    NA
## [5,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
##      [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##
## , , 2
##
##           [,1]      [,2]      [,3]       [,4]       [,5]       [,6]
## [1,] 0.2875677 0.3321588 0.8196699 0.30963843         NA         NA
## [2,] 0.2187130 0.3447453 0.4772526 0.10689339 0.28756774 0.33215875
## [3,] 0.2875677 0.8196699 0.3096384 0.06489925 0.06840052 0.07157824
## [4,] 0.2187130 0.3447453 0.3504187 0.50962684 0.28756774 0.12537991
## [5,] 0.1253799        NA        NA         NA         NA         NA
##           [,7]       [,8]       [,9] [,10] [,11] [,12] [,13] [,14] [,15]
## [1,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
## [2,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
## [3,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
## [4,] 0.06489925 0.06840052 0.07157824    NA    NA    NA    NA    NA    NA
## [5,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
##      [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##
## , , 3
##
##           [,1]      [,2]      [,3]       [,4]       [,5]       [,6]
## [1,] 0.2464866 0.2847075 0.6656957 0.28945075         NA         NA
## [2,] 0.1874683 0.3211343 0.4090737 0.09673551 0.24648663 0.28470750
## [3,] 0.2464866 0.6656957 0.2894508 0.05562793 0.05862902 0.06135277
## [4,] 0.1874683 0.3211343 0.3250582 0.40026678 0.24648663 0.10746850
## [5,] 0.1074685        NA        NA         NA         NA         NA
##           [,7]       [,8]       [,9] [,10] [,11] [,12] [,13] [,14] [,15]
## [1,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
## [2,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
## [3,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
## [4,] 0.05562793 0.05862902 0.06135277    NA    NA    NA    NA    NA    NA
## [5,]        NA         NA         NA    NA    NA    NA    NA    NA    NA
##      [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
```

```
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##
## , , 4
##
##           [,1]      [,2]      [,3]       [,4]       [,5]       [,6]
## [1,] 0.21567580 0.2491191 0.5194239 0.27095880         NA         NA
## [2,] 0.16403475 0.2973442 0.3579395 0.08667708 0.21567580 0.24911907
## [3,] 0.21567580 0.5194239 0.2709588 0.04867444 0.05130039 0.05368368
## [4,] 0.16403475 0.2973442 0.3012788 0.29539003 0.21567580 0.09403493
## [5,] 0.09403493        NA        NA         NA         NA         NA
##           [,7]       [,8]       [,9] [,10] [,11] [,12] [,13] [,14] [,15]
## [1,]         NA         NA         NA    NA    NA    NA    NA    NA    NA
## [2,]         NA         NA         NA    NA    NA    NA    NA    NA    NA
## [3,]         NA         NA         NA    NA    NA    NA    NA    NA    NA
## [4,] 0.04867444 0.05130039 0.05368368    NA    NA    NA    NA    NA    NA
## [5,]         NA         NA         NA    NA    NA    NA    NA    NA    NA
##      [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## [1,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [2,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [3,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [4,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## [5,]    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
##
##
## [[4]]
##           [,1]      [,2]     [,3]     [,4]      [,5]     [,6] [,7]
## [1,] 1.1543571 0.4058942 4.427783 2.121113 0.0000000 0.000000    0
## [2,] 3.1266614 0.8598067 2.416878 1.699546 0.0000000 0.000000    0
## [3,] 0.7972984 6.0618427 3.576324 0.000000 0.2724501 1.594544    0
## [4,] 2.1055480 0.0000000 1.134459 1.241971 1.1868695 0.000000    0
## [5,] 0.4641186 0.0000000 0.000000 0.000000 0.0000000 0.000000    0
##           [,8]      [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17]
## [1,] 0.0000000 0.0000000     0     0     0     0     0     0     0     0
## [2,] 0.0000000 0.0000000     0     0     0     0     0     0     0     0
## [3,] 0.0000000 0.0000000     0     0     0     0     0     0     0     0
## [4,] 0.1569811 0.3427198     0     0     0     0     0     0     0     0
## [5,] 0.0000000 0.0000000     0     0     0     0     0     0     0     0
##      [,18] [,19] [,20] [,21] [,22] [,23] [,24] [,25]
## [1,]     0     0     0     0     0     0     0     0
## [2,]     0     0     0     0     0     0     0     0
## [3,]     0     0     0     0     0     0     0     0
## [4,]     0     0     0     0     0     0     0     0
## [5,]     0     0     0     0     0     0     0     0
##
## [[5]]
## [[5]][[1]]
## [1] "Mbias"      "FMSY_Mbias" "Abias"      "Aerr"
##
## [[5]][[2]]
## [1] "Cbias"      "Csd"        "Dbias"      "Derr"       "Mbias"
```

```
## [6] "FMSY_Mbias"
##
## [[5]][[3]]
## [1] "Mbias"    "Abias"    "Aerr"     "Kbias"    "t0bias"   "Linfbias"
##
## [[5]][[4]]
## [1] "Cbias"    "Csd"      "Isd"      "betas"    "Mbias"    "lenMbias"
## [7] "Kbias"    "t0bias"   "Linfbias"
##
## [[5]][[5]]
## [1] "lenMbias"
##
##
## [[6]]
## [1] "Fratio"    "DCAC"      "Fdem"      "DD"        "matlenlim"
```

`VOI2` assumes that relative cost for each type of improvement in data is linearly related to the number of samples (e.g. nCAAobs) or square function of improved precision and bias e.g.: relative cost = $\frac{1}{(\text{newCV}/\text{oldCV})^2}$

The VOI features of DLMtool are continuing to be developed and more VOI functions will be added soon.

Using Fishery Data

Chapter 15

The Fishery Data Object (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 `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.

15.1 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`).

The second argument for all Management Procedures in the DLMtool is the `Data` object.

For example, the arguments to the `AvC` MP are:

```
args(AvC)
```

```
## function (x, Data, reps = 100)
## NULL
```

The Developing Custom Management Procedures section describes the arguments and internal workings of the Management Procedure functions in more detail.

15.2 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.

Chapter 16

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"        "SimulatedData"      "Simulation_1"
## [10] "China_rockfish2"    "Data"               "Madeup"
## [13] "Recs"
```

Chapter 17

Creating Your Own 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.

17.1 Creating a Data File in Excel

Probably the easiest way to get your data into the DLMtool is to populate a data table in an Excel workbook.

You can create a Data workbook using the `DataInit` function, for example:

```
DataInit("MyData")
```

This will create a file 'MyData.xlsx' in your current working directory, which can be populate with your fishery data.

Remember, to see the help documentation for information on the slots in the Data object:

```
?class("Data")
```

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.

17.2 Importing the Data object

Once populated, the Excel Data file can be imported into R:

```
MyData <- XL2Data('MyData')
```

In this case we get an error because the Data file is empty: we haven't populated it with an data yet. Luckily for us, DLMtool includes several example Data files.

17.3 Example Fishery Data Files

One example Data file is the China rockfish. You can download this Data file to your current working directory and import into R:

```
China_rockfish <- XL2Data("China_rockfish.csv")
```

The CSV files for the other example Fishery Data objects are also 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-3.4.3/library/DLMtool"
```

We can then load one of the example CSV files using the `XL2Data` function:

```
China_rockfish2 <- XL2Data("China_rockfish.csv", dir=DLMDataDir())
```

```
## Reading China_rockfish.csv
```

or the `new` function:

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

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

17.4 Populating a Data Object in R

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"
```

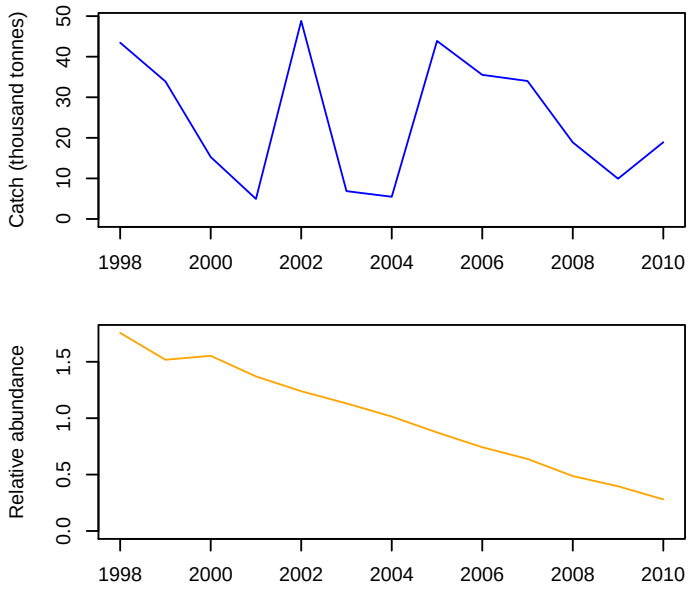
```
Madeup@Name <- 'Test'                                  #  Name it
Madeup@Cat <- matrix(20:11*rlnorm(10,0,0.2),nrow=1)    #  Generate fake catch data
Madeup@Units <- "Million metric tonnes"                #  State units of catch
Madeup@AvC <- mean(Madeup@Cat)                         #  Average catches 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
```

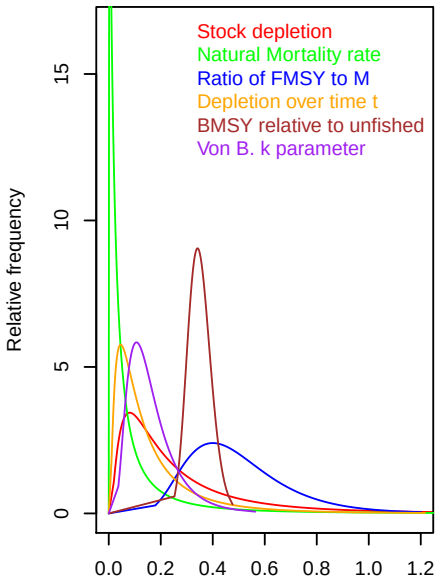
Chapter 18

Plotting Data Objects

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

```
summary(Atlantic_mackerel)
```





This feature is currently being developed.

Chapter 19

Determining Which MPs Can Be Applied

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:

The `Can` function generates a list of the MPs that are available to be applied to the Data object:

```
Can(Atlantic_mackerel)
```

```
##  [1] "AvC"          "AvC_MLL"      "BK"           "CC1"
##  [5] "CC4"          "curE"         "curE75"       "DAAC"
##  [9] "DBSRA"        "DBSRA_40"     "DBSRA4010"    "DCAC"
## [13] "DCAC_40"      "DCAC4010"     "DD"           "DD4010"
## [17] "DDe"          "DDe75"        "DDes"         "DepF"
## [21] "DTe40"        "DTe50"        "DynF"         "Fadapt"
## [25] "Fdem"         "Fratio"       "Fratio4010"   "GB_slope"
## [29] "Gcontrol"     "HDAAC"        "Itarget1"     "Itarget1_MPA"
## [33] "Itarget4"     "ItargetE1"    "ItargetE4"    "matlenlim"
## [37] "matlenlim2"   "MCD"          "MCD4010"      "minlenLopt1"
## [41] "MRnoreal"     "MRreal"       "NFref"        "Rcontrol"
## [45] "Rcontrol2"    "SBT1"         "slotlim"      "SPmod"
## [49] "SPMSY"        "SPslope"      "SPSRA"        "YPR"
## [53] "TCPUE_e"      "THC"
```

The `Cant` function displays a list of the MPs that cannot be applied to the Data object together with the reason why:

```
Cant(Atlantic_mackerel)
```

```
##       [,1]          [,2]
##  [1,] "BK_CC"       "Missing data: CAA"
##  [2,] "BK_ML"       "Missing data: CAL"
##  [3,] "CompSRA"     "Missing data: CAA"
##  [4,] "CompSRA4010" "Missing data: CAA"
##  [5,] "DCAC_ML"     "Missing data: CAL"
##  [6,] "EtargetLopt" "Missing data: ML"
##  [7,] "Fdem_CC"     "Missing data: CAA"
##  [8,] "Fdem_ML"     "Missing data: CAL"
##  [9,] "FMSYref"     "Missing data: OM"
## [10,] "FMSYref50"   "Missing data: OM"
```

```
## [11,] "FMSYref75"   "Missing data: OM"
## [12,] "Fratio_CC"   "Missing data: CAA"
## [13,] "Fratio_ML"   "Missing data: CAL"
## [14,] "GB_CC"       "Missing data: Cref"
## [15,] "GB_target"   "Missing data: Cref, Iref"
## [16,] "Islope1"     "Missing data: MPrec"
## [17,] "Islope4"     "Missing data: MPrec"
## [18,] "IT10"        "Missing data: Iref, MPrec"
## [19,] "IT5"         "Missing data: Iref, MPrec"
## [20,] "ITe10"       "Missing data: Iref"
## [21,] "ITe5"        "Missing data: Iref"
## [22,] "ITM"         "Missing data: Iref, MPrec"
## [23,] "L95target"   "Missing data: ML"
## [24,] "LstepCC1"    "Missing data: ML, MPrec"
## [25,] "LstepCC4"    "Missing data: ML, MPrec"
## [26,] "LstepCE1"    "Missing data: ML"
## [27,] "LstepCE2"    "Missing data: ML"
## [28,] "Ltarget1"    "Missing data: ML"
## [29,] "Ltarget4"    "Missing data: ML"
## [30,] "LtargetE1"   "Missing data: ML"
## [31,] "LtargetE4"   "Missing data: ML"
## [32,] "SBT2"        "Missing data: Rec, Cref"
## [33,] "SPSRA_ML"    "Missing data: CAL"
## [34,] "YPR_CC"      "Missing data: CAA"
## [35,] "YPR_ML"      "Missing data: CAL"
## [36,] "TCPUE"       "Missing data: MPrec"
```

Finally, the `Needed` function displays the additional data slots that must be populated for the unavailable MPs to work:

```
Needed(Atlantic_mackerel)
```

```
##             [,1]
## BK_CC       "Missing data: CAA"
## BK_ML       "Missing data: CAL"
## CompSRA     "Missing data: CAA"
## CompSRA4010 "Missing data: CAA"
## DCAC_ML     "Missing data: CAL"
## EtargetLopt "Missing data: ML"
## Fdem_CC     "Missing data: CAA"
## Fdem_ML     "Missing data: CAL"
## FMSYref     "Missing data: OM"
## FMSYref50   "Missing data: OM"
## FMSYref75   "Missing data: OM"
## Fratio_CC   "Missing data: CAA"
## Fratio_ML   "Missing data: CAL"
## GB_CC       "Missing data: Cref"
## GB_target   "Missing data: Cref, Iref"
## Islope1     "Missing data: MPrec"
## Islope4     "Missing data: MPrec"
## IT10        "Missing data: Iref, MPrec"
## IT5         "Missing data: Iref, MPrec"
## ITe10       "Missing data: Iref"
## ITe5        "Missing data: Iref"
## ITM         "Missing data: Iref, MPrec"
```

```
## L95target   "Missing data: ML"
## LstepCC1    "Missing data: ML, MPrec"
## LstepCC4    "Missing data: ML, MPrec"
## LstepCE1    "Missing data: ML"
## LstepCE2    "Missing data: ML"
## Ltarget1    "Missing data: ML"
## Ltarget4    "Missing data: ML"
## LtargetE1   "Missing data: ML"
## LtargetE4   "Missing data: ML"
## SBT2        "Missing data: Rec, Cref"
## SPSRA_ML    "Missing data: CAL"
## YPR_CC      "Missing data: CAA"
## YPR_ML      "Missing data: CAL"
## TCPUE       "Missing data: MPrec"
```

Chapter 20

Applying Management Procedures

The `runMP` function can be used to apply a MP to a Data object. For example, to apply the `AvC` method:

```
runMP(Atlantic_mackerel, "AvC", reps=1000)
```

```
##       TAC
## AvC 24.33
```

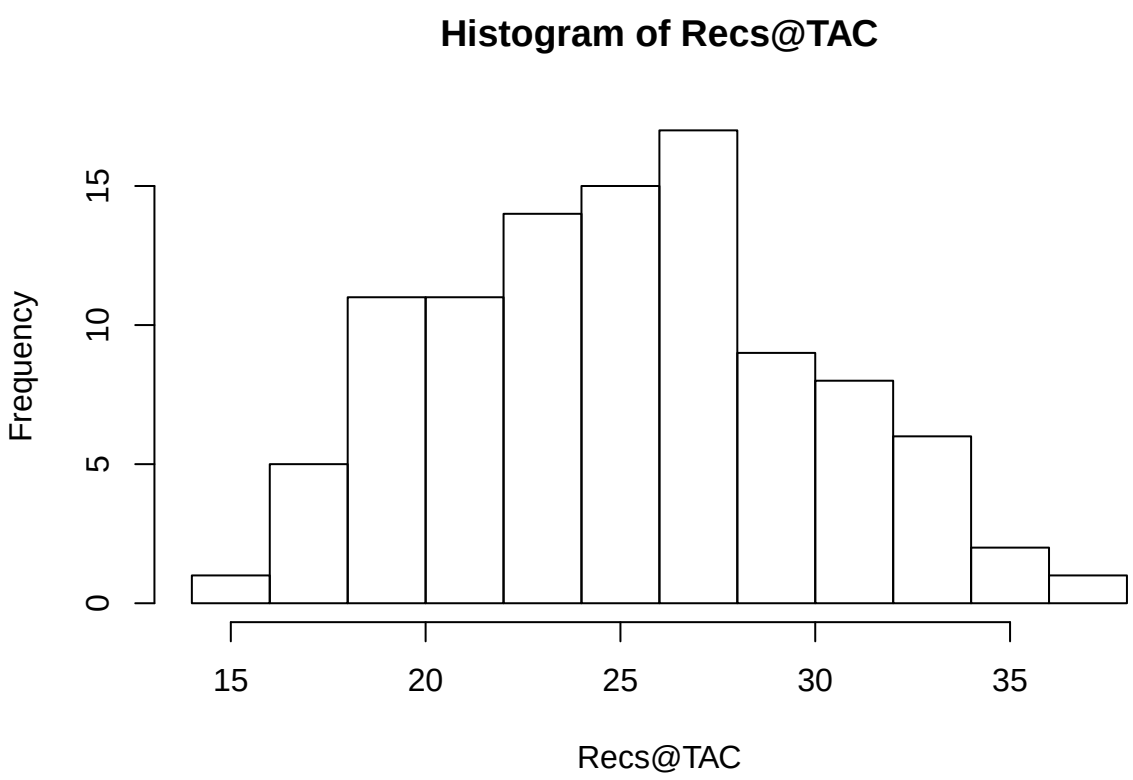
The `runMP` prints out the MP recommendations to the console. In the case of a TAC, where multiple repititions where (see `reps` = 1000 above) used the `runMP` function prints the median TAC recommendation.

Although it only displays a summary, `runMP` invisibly returns the Data object with the TAC slot populated:

```
Recs <- runMP(Atlantic_mackerel, "AvC")
```

```
##       TAC
## AvC 24.76
```

```
hist(Recs@TAC)
```



runMP can be used to run several MPs:

```
runMP(Atlantic_mackerel, c("AvC", "AvC_MLL"))
```

```
##           TAC   LR5 LFR
## AvC     24.54
## AvC_MLL 23.88 90.25  95
```

Or all available MPs:

```
Atlantic_mackerel <- runMP(Atlantic_mackerel, reps=1000)
```

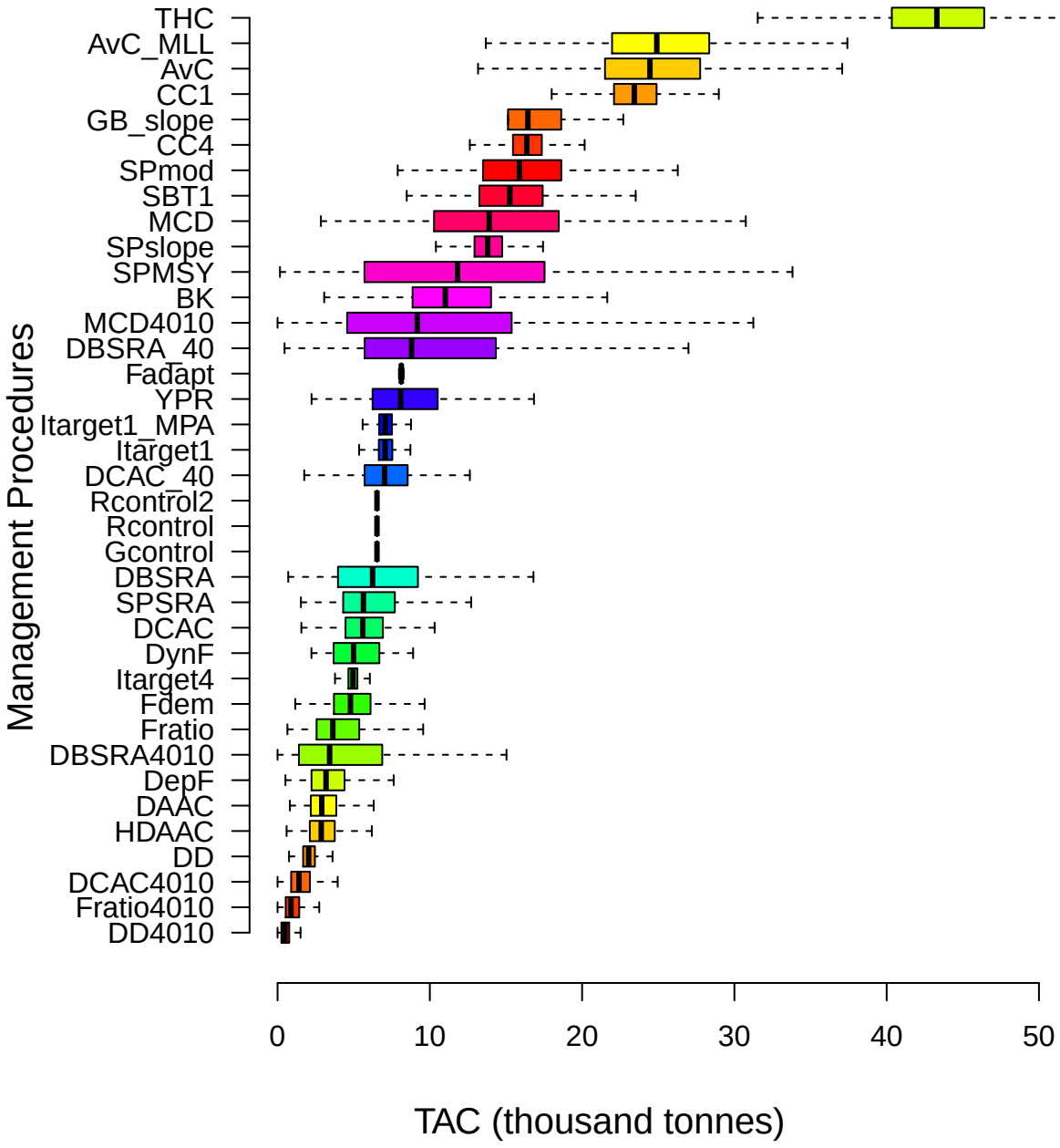
```
## running all available MPs
```

```
##                TAC Effort   LR5    LFR     HS Area 1 Area 2
## AvC          24.45
## AvC_MLL      24.90        90.25  95.00
## BK           11.02
## CC1          23.43
## CC4          16.37
## curE                1.00
## curE75              0.75
## DAAC          2.90
## DBSRA         6.24
## DBSRA_40      8.81
## DBSRA4010     3.42
## DCAC          5.60
## DCAC_40       7.03
## DCAC4010      1.41
```

```
## DD            2.04
## DD4010        0.48
## DDe                 0.01
## DDe75               0.01
## DDes                0.90
## DepF          3.18
## DTe40               0.90
## DTe50               0.90
## DynF          5.00
## Fadapt        8.13
## Fdem          4.79
## Fratio        3.63
## Fratio4010    0.88
## GB_slope     16.44
## Gcontrol      6.53
## HDAAC         2.87
## Itarget1      7.06
## Itarget1_MPA  7.08                                  0      1
## Itarget4      4.96
## ItargetE1           0.85
## ItargetE4           0.80
## matlenlim                 90.25  95.00
## matlenlim2                99.28 104.50
## MCD          13.90
## MCD4010       9.18
## minlenLopt1               89.75  99.72
## MRnoreal                                            0      1
## MRreal                                              0      1
## NFref         0.01
## Rcontrol      6.53
## Rcontrol2     6.53
## SBT1         15.26
## slotlim                   99.28 104.50 142.55
## SPmod        15.89
## SPMSY        11.82
## SPslope      13.80
## SPSRA         5.64
## YPR           8.08
## TCPUE_e             0.95
## THC          43.30
```

The TAC recommendations from each Output control can be plotted:

```
boxplot(Atlantic_mackerel)
```


Management Procedures
THC
AvC_MLL
AvC
CC1
GB_slope
CC4
SPmod
SBT1
MCD
SPslope
SPMSY
BK
MCD4010
DBSRA_40
Fadapt
YPR
Itarget1_MPA
Itarget1
DCAC_40
Rcontrol2
Rcontrol
Gcontrol
DBSRA
SPSRA
DCAC
DynF
Itarget4
Fdem
Fratio
DBSRA4010
DepF
DAAC
HDAAC
DD
DCAC4010
Fratio4010
DD4010
0
10
20
30
40
50
TAC (thousand tonnes)

Advanced DLMtool

Chapter 21

Customizing the Operating Model

21.1 Accounting for Historical Changes in Fishing

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, historical changes in the selectivity pattern, and the existence of MPAs.

Remember to update and recompile the OM documentation whenever the OM is modified.

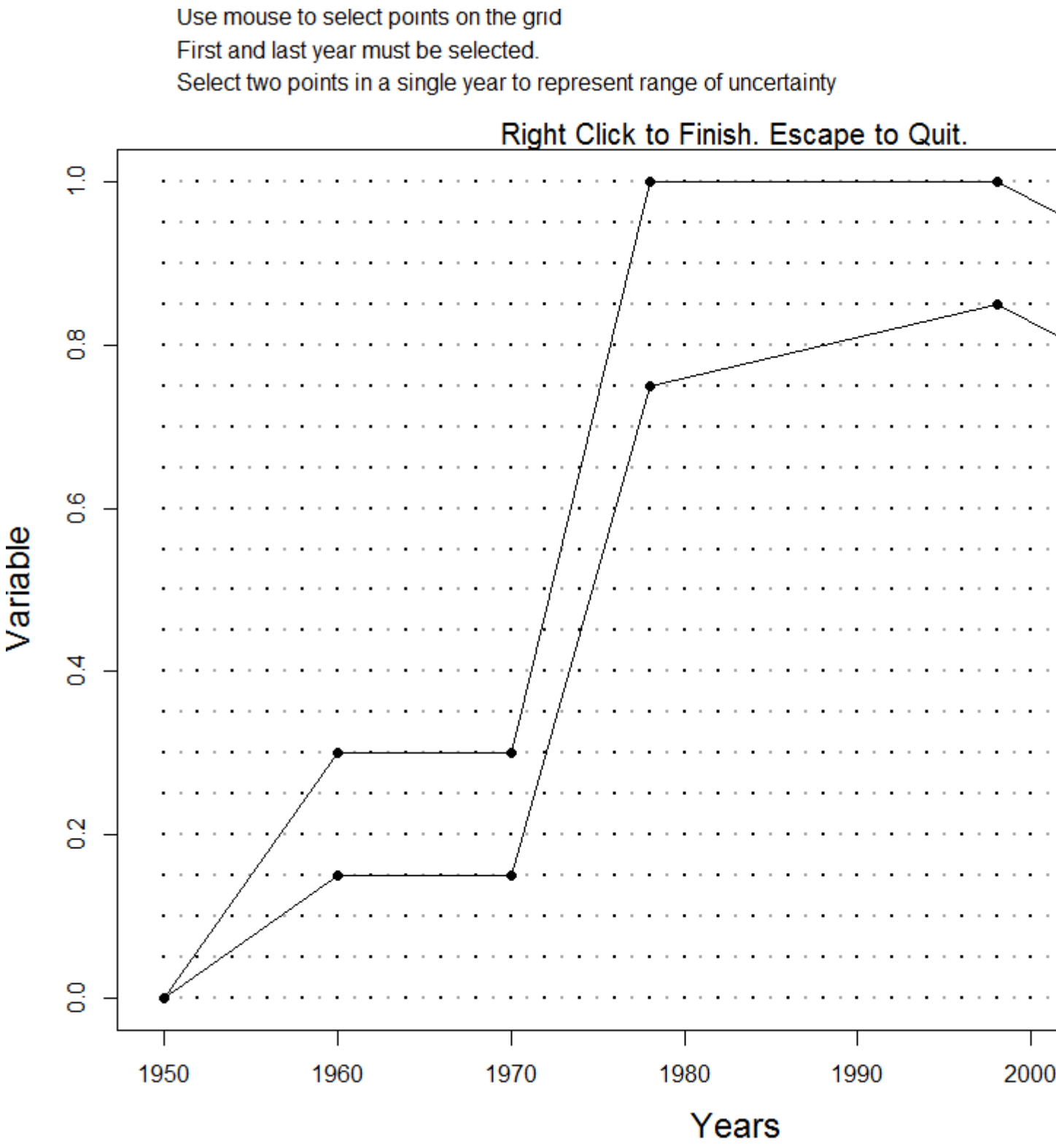
21.1.1 Historical Trends in Fishing Mortality

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.

```
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.

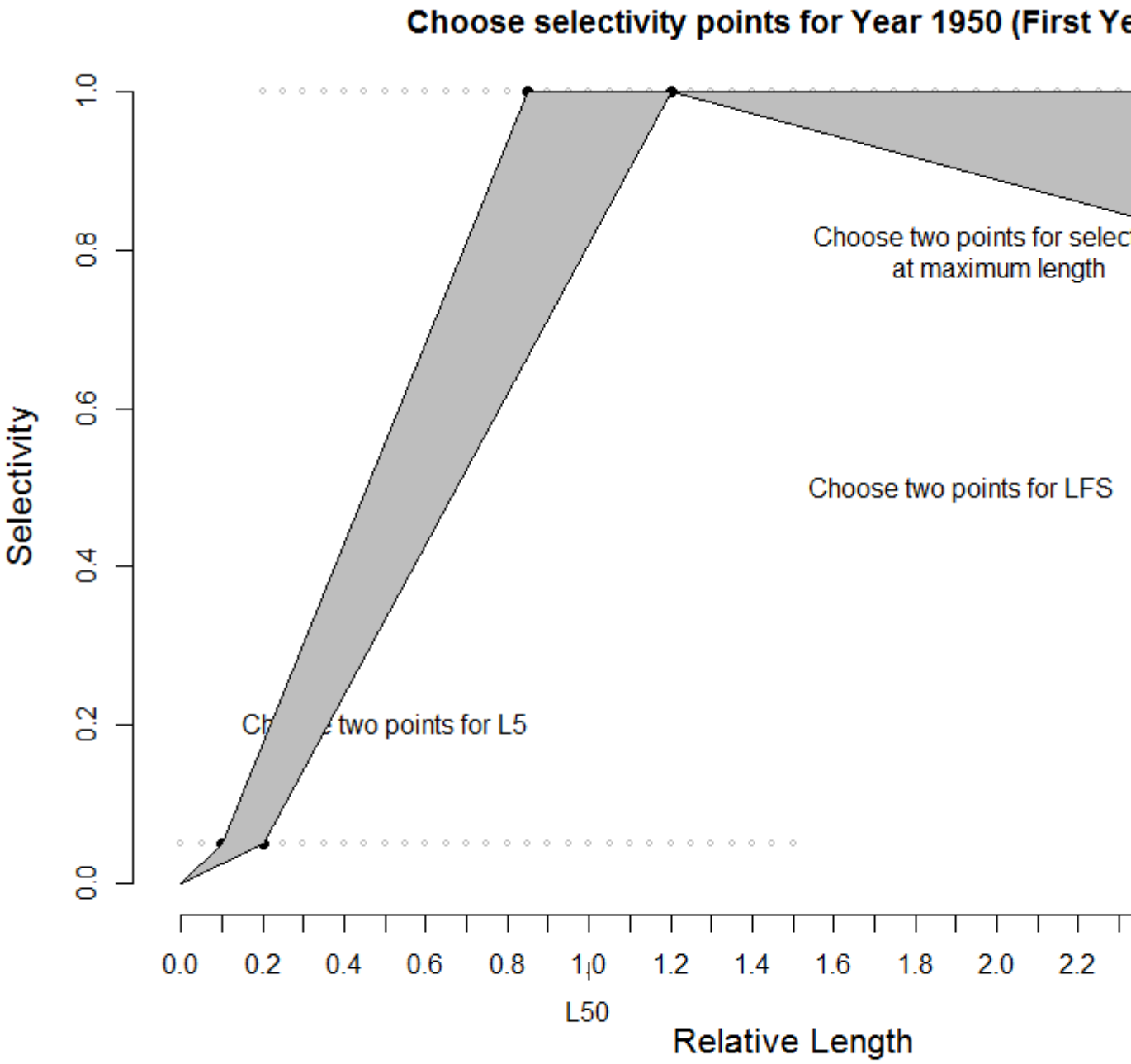
21.1.2 Historical Trends in Selectivity Pattern

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 object 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.

21.1.3 Including Existing MPAs

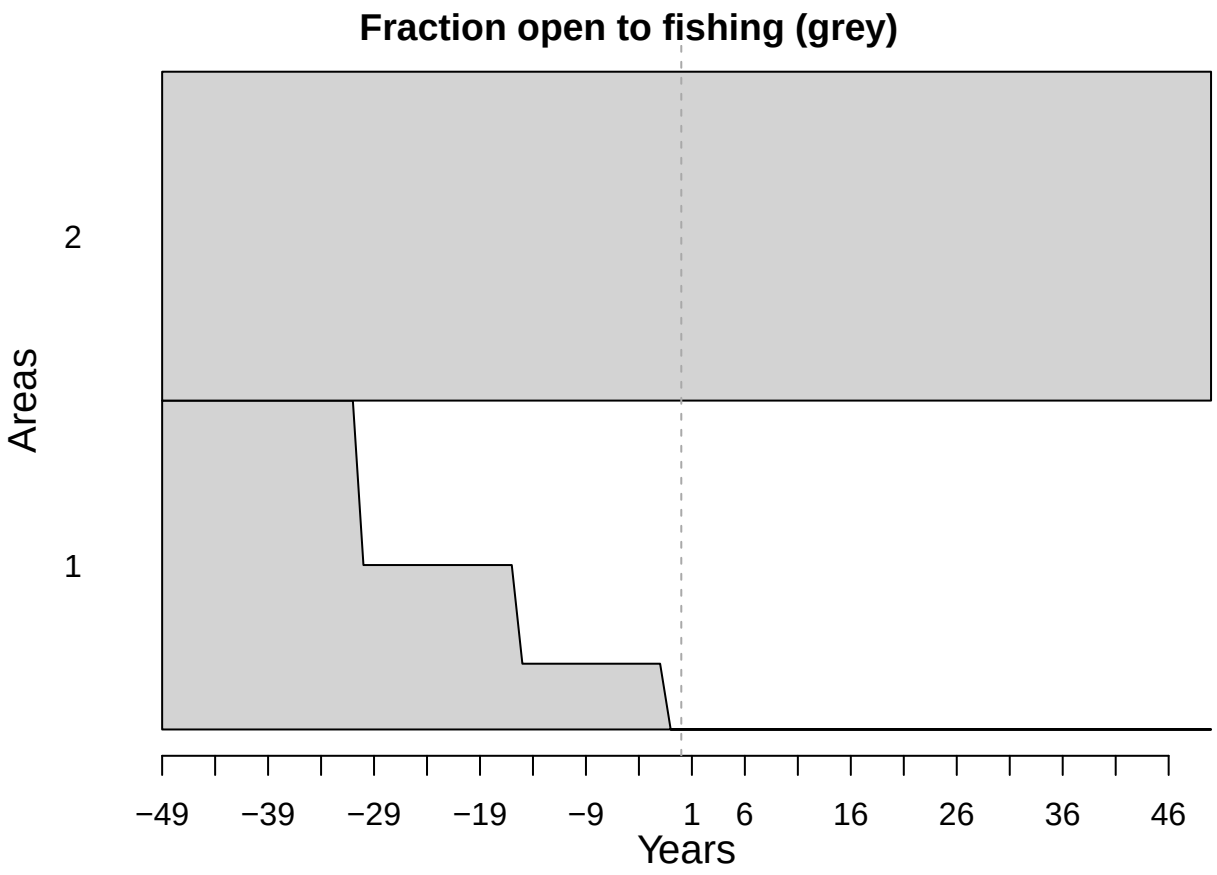
By default the MSE assumes that there are no spatial closures in the historical period. Existing spatial closures can be accounted for with the `MPA` slot in the `Fleet` or `OM` object.

To account for historical MPAs, the `MPA` slot should be a matrix with each row should containing a year index (e.g 10 for 10th historical year) followed by fraction of area closed to fishing for each area. i.e. each row represents a change and the number of columns is `nareas` (default is 2) + 1.

The spatial closures are assumed to remain in place for the future projections unless changed by a MP. Default (if left blank) is all areas are open to fishing in historical period.

For example:

```
OM <- new("OM", Albacore, Generic_Fleet, Perfect_Info, Perfect_Imp)

## 50% of Area 1 was closed 30 years ago
cl1 <- c(OM@nyears-30, 0.5, 1)
## 80% of Area 1 was closed 15 years ago
cl2 <- c(OM@nyears-15, 0.2, 1)
## 100% of Area 1 was closed last year
cl3 <- c(OM@nyears-1, 0, 1)

OM@MPA <- matrix(c(cl1, cl2, cl3), ncol=3, byrow=TRUE)
plotMPA(OM)
```



21.2 Size-Specific Natural Mortality

21.2.1 Constant M at age/size

By default DLMtool assumes that natural mortality (M) is constant across age and size classes. However, in many species M is known to vary by size, and is often assumed to be higher for smaller age-classes and reduces as individuals age and grow.

A number of users requested the option to include age or size-specific M and this has now been added to DLMtool.

There are a number of ways to specify age or size-specific M in DLMtool.

21.2.2 Lorenzen function of weight

Natural mortality is often assumed to be a function of weight. Size-specific M can be included in DLMtool following the approach of Lorenzen (1996):

$$M_w = M \left(\frac{W}{W_\infty} \right)^b$$

where M_w is the natural mortality at weight W, M is the natural mortality rate of adult fish, W_∞ is the asymptotic weight, and b is the allometric scaling factor (`Stock@Mexp`). Lorezen (1996) found that the

exponent b had an average value of -0.288, with 90% confidence intervals of -0.315 – -0.261 for fish from natural systems.

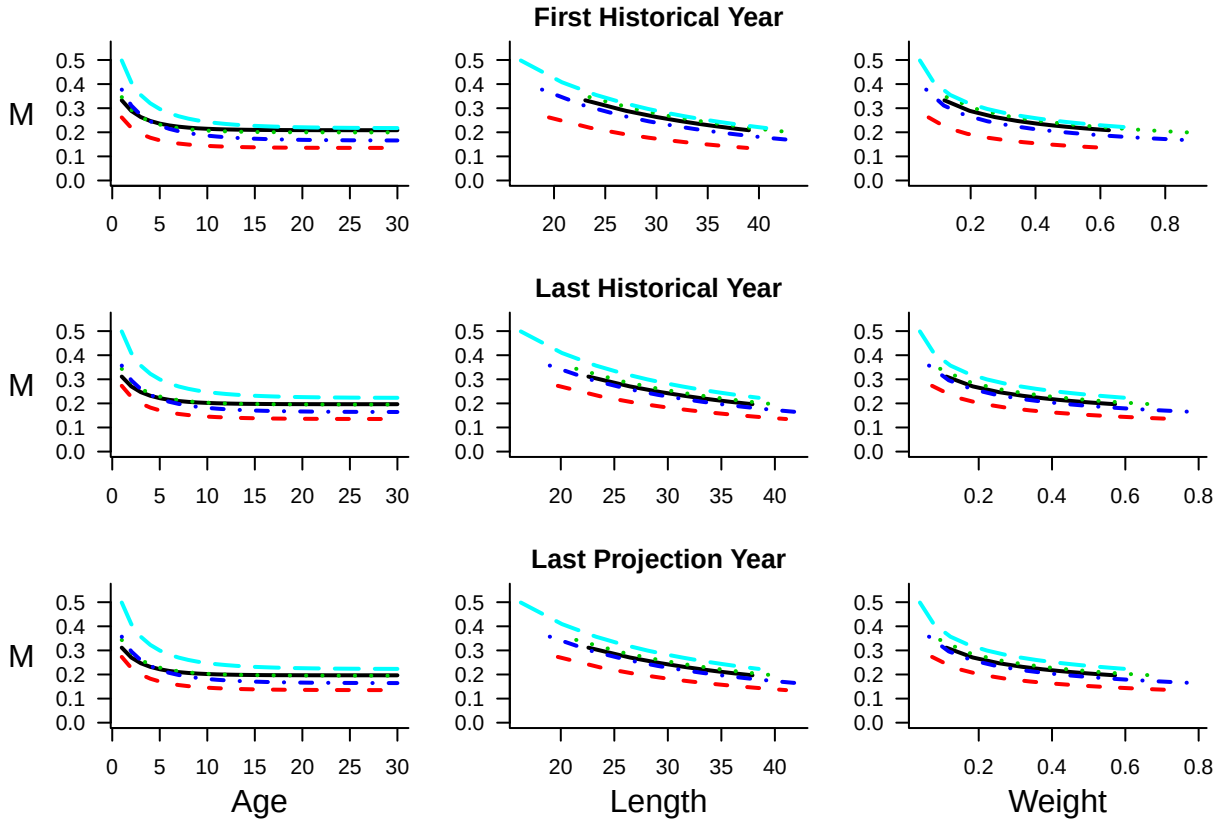
Because DLMtool uses an age-structured model, M is calculated as a function of age:

$$M_a = M \left(\frac{W_a}{W_\infty} \right)^b$$

where M_a is natural mortality at age a and W_a is the mean weight at age a. M-at-age is then rescaled so that the mean M of adult age classes (mean age of maturity and greater) is equal to the natural mortality rate sampled from the stock object (`Stock@M`).

The `plotM` function can be used to visually inspect samples of the M-at-age, -length, and -weight that are generated by the model:

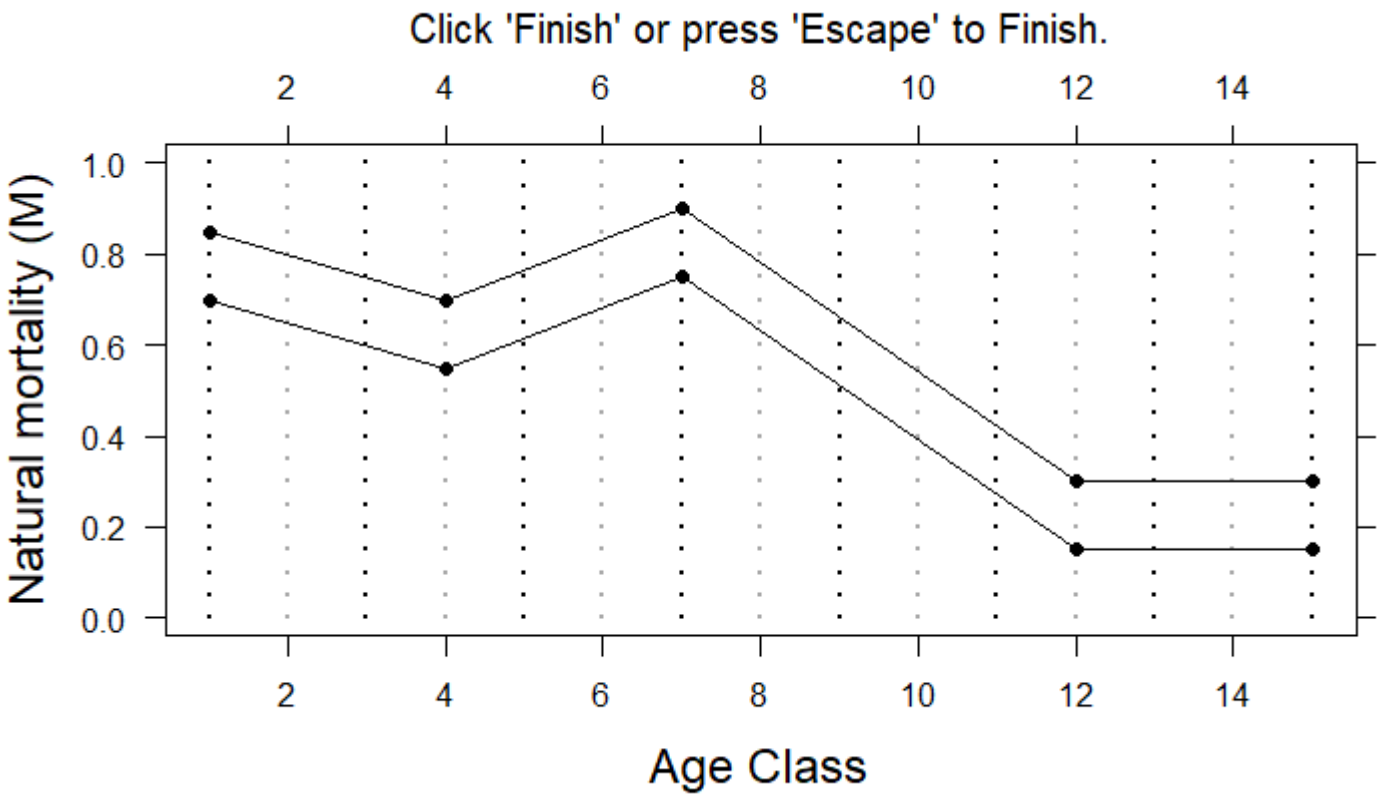
```
Mackerel@Mexp <- c(-0.315, -0.261)
plotM(Mackerel)
```



21.2.3 Map Age-Specific M

Usually the `M` slot contains two values, a lower and upper bound for the constant M-at-age. Users who wish for more control of M-at-age can use the `M` and `M2` slots in the `Stock` object to directly input values for M-at-age (`M` for lower bound and `M2` for upper bound). `maxage` values of M must be supplied for slots `M` and `M2`. One way to do this is to use the `ChooseM` function to map out the bounds for age-specific M:

```
OM <- new("OM", Blue_shark, Generic_FlatE, Generic_Obs, Perfect_Imp)
OM <- ChooseM(OM)
```



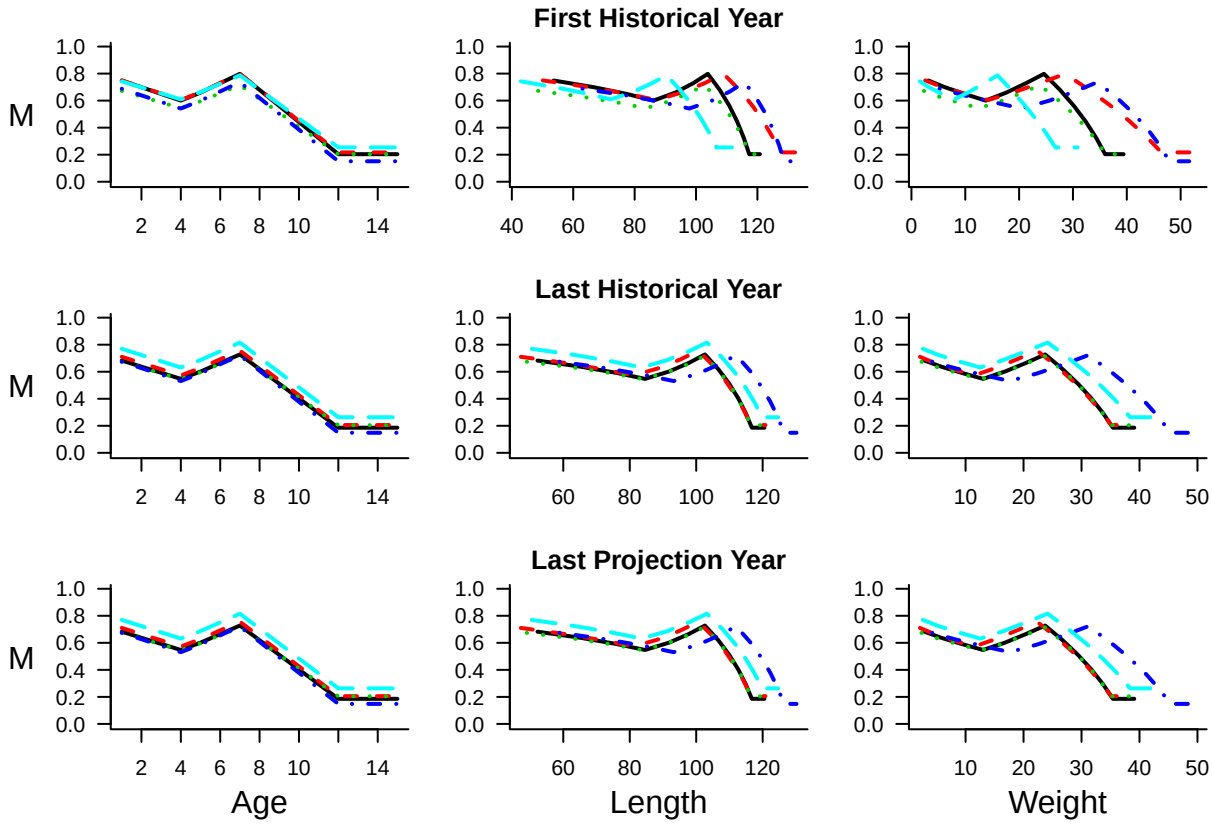
Click here for a larger version of the image.

Alternatively, users can input the values directly into the `M` and `M2` slots (must be length `maxage`):

```
OM <- new("OM", Albacore, Generic_FlatE, Generic_Obs, Perfect_Imp)
OM@M <- c(0.7, 0.65, 0.60, 0.55, 0.61, 0.68, 0.75, 0.63, 0.51, 0.39, 0.27, 0.15, 0.15, 0.15, 0.15)
OM@M2 <- c(0.85, 0.8, 0.75, 0.7, 0.76, 0.83, 0.9, 0.78, 0.66, 0.54, 0.42, 0.3, 0.3, 0.3, 0.3)
```

The `plotM` function can then be used to visually display samples of the resulting M at age and size:

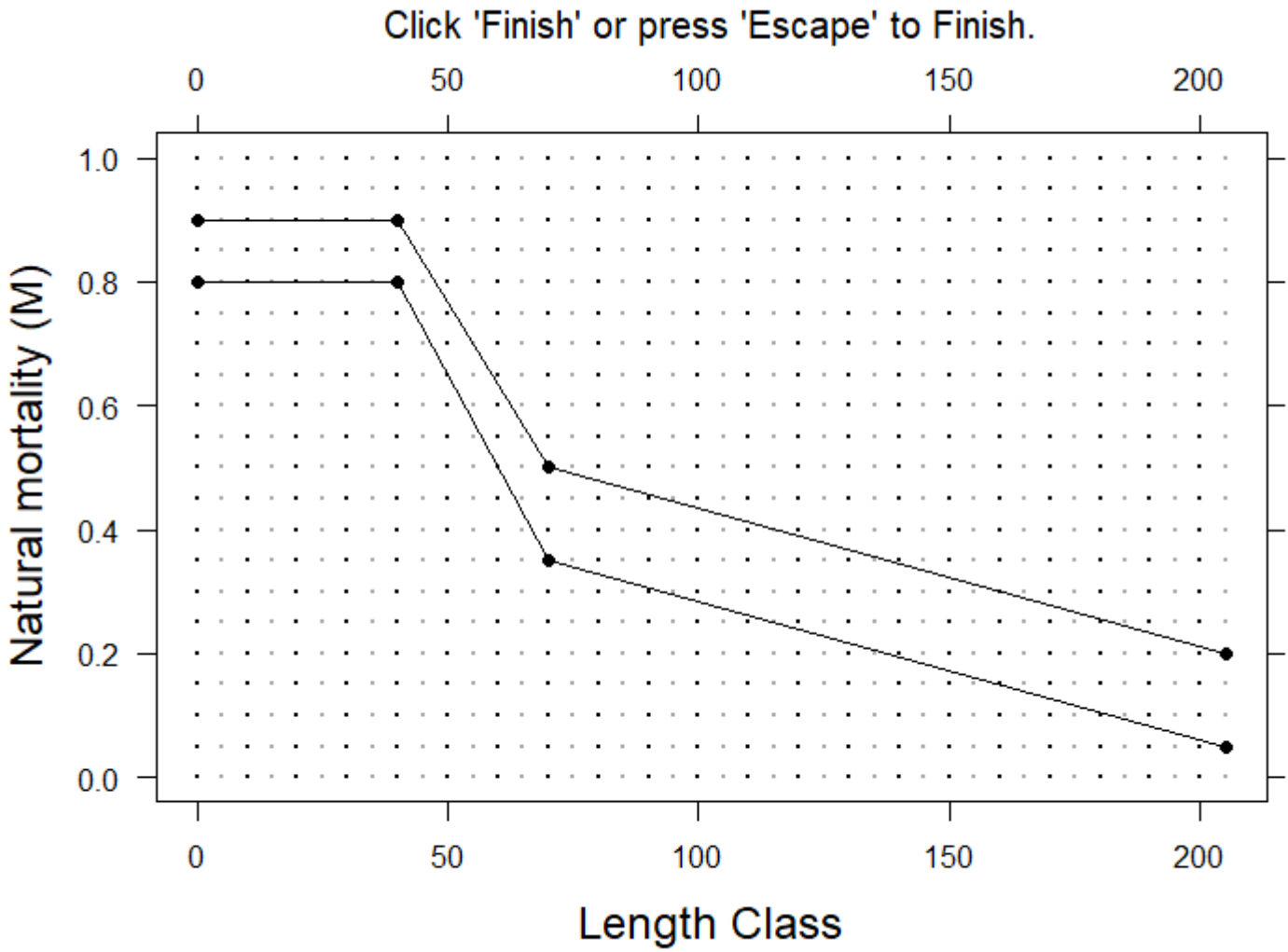
```
plotM(OM)
```



21.2.4 Map Length-Specific M

There is also the option to map length-specific M using the plotting tool:

```
OM <- new("OM", Albacore, Generic_FlatE, Generic_Obs, Perfect_Imp)
OM <- ChooseM(OM, "length")
```



Click here for a larger version of the image.

This option uses the Custom Parameters feature of DLMtool:

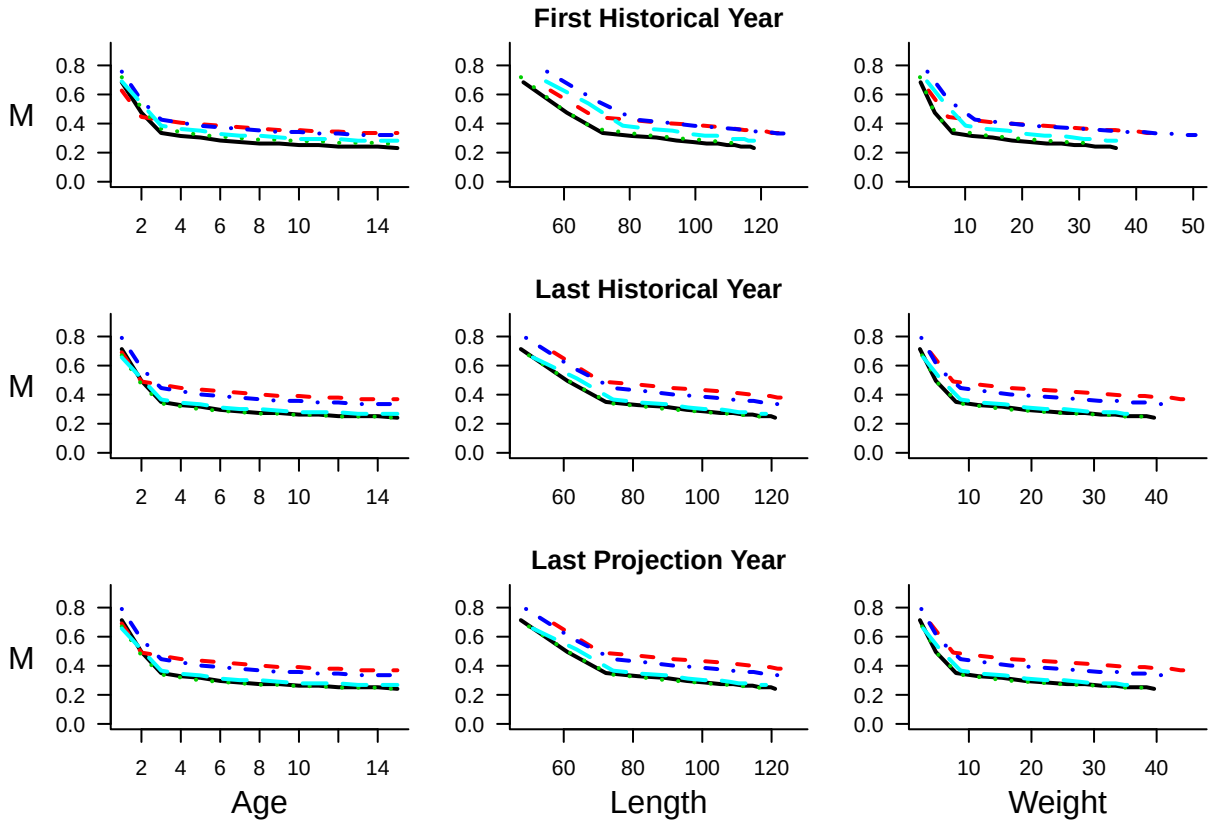
```
str(OM@cpars)
```

```
## List of 1
##  $ M_at_Length:'data.frame': 42 obs. of  3 variables:
##   ..$ Lens: int [1:42] 0 5 10 15 20 25 30 35 40 45 ...
##   ..$ M1  : num [1:42] 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.725 ...
##   ..$ M2  : num [1:42] 0.9 0.9 0.9 0.9 0.9 ...
```

Again, samples of the resulting M at age and size can be plotted:

```
plotM(OM)
```

```
## valid custom parameters (OM@cpars) found:
## M_at_Length
```



21.3 Selection, Retention and Discard Mortality

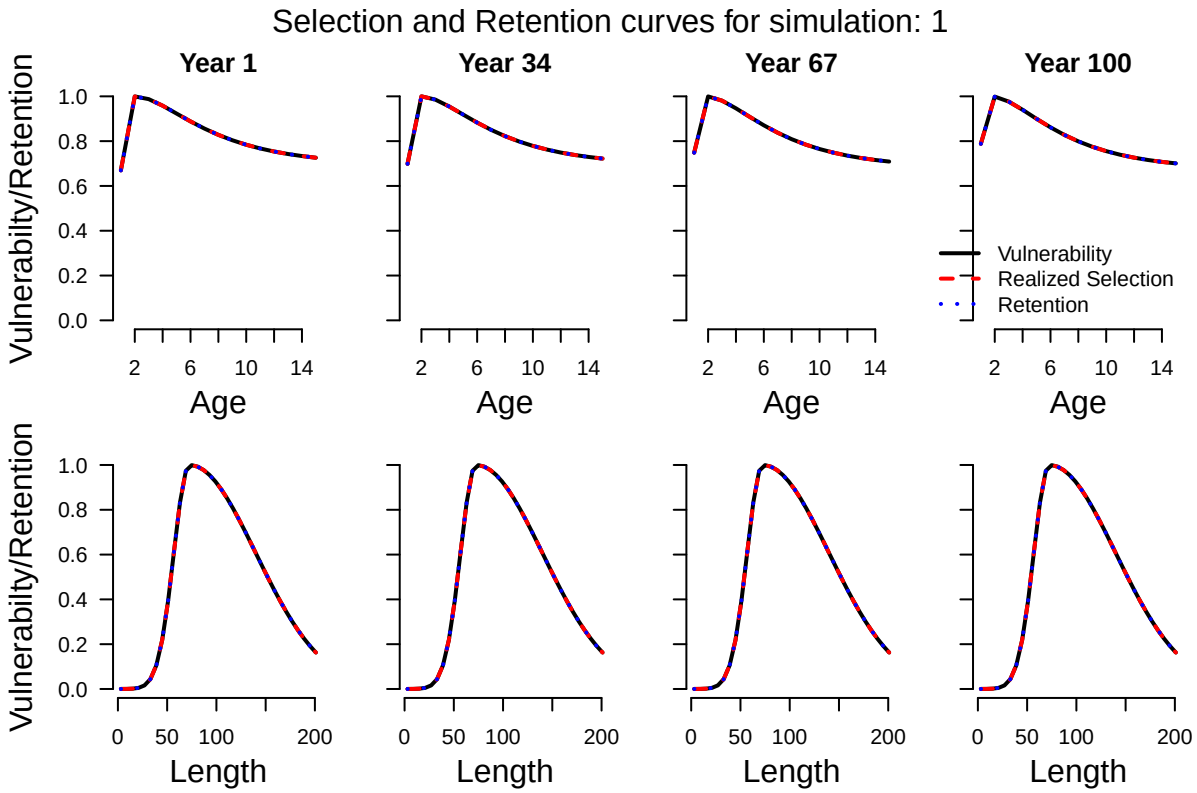
21.3.1 Fishery Selection Curve

The fishery selection or vulnerability to the fishing gear in DLMtool is modelled using a double-normal curve and the parameters in the `Fleet` object: `L5` - smallest length at 5% selection, `LFS` - smallest length at full selection, and `Vmaxlen` the vulnerability of the largest length class (defined as expected length at maximum age `Stock@maxage`).

Here we set up a Operating Model with dome-shaped selectivity and plot a sample of the selectivity-at-age and -length using the `plotSelect` function:

```
OM <- new("OM", Albacore, FlatE_Dom, Generic_Obs, Perfect_Imp, nsim=5)
plotSelect(OM, sim=1)
```

```
## Optimizing for user-specified movement
```



The plot shows three curves - vulnerability, realized selection and retention - in each panel. In this case they are all the same, because the default setting of DLMtool is to assume that all selected fish are retained in the catch.

21.3.2 Fishery Retention Curve

In some cases the fishing gear selects fish (often small sizes) that are not retained in the catch and are discarded at sea. The fishery-retention curve can be specified following the same approach as selectivity, using the following slots in the `Fleet` or `OM` object:

- `LR5` - the smallest length at 5% retention
- `LFR` - the smallest length at full selection
- `Rmaxlen` - the retention of the largest size class (defined as expected length at maximum age `Stock@maxage`).

The default values for these parameters are:

```
OM@LR5
```

```
## [1] 0 0
```

```
OM@LFR
```

```
## [1] 0 0
```

```
OM@Rmaxlen
```

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

meaning that the default assumption is that all size classes are fully retained by the fishery.

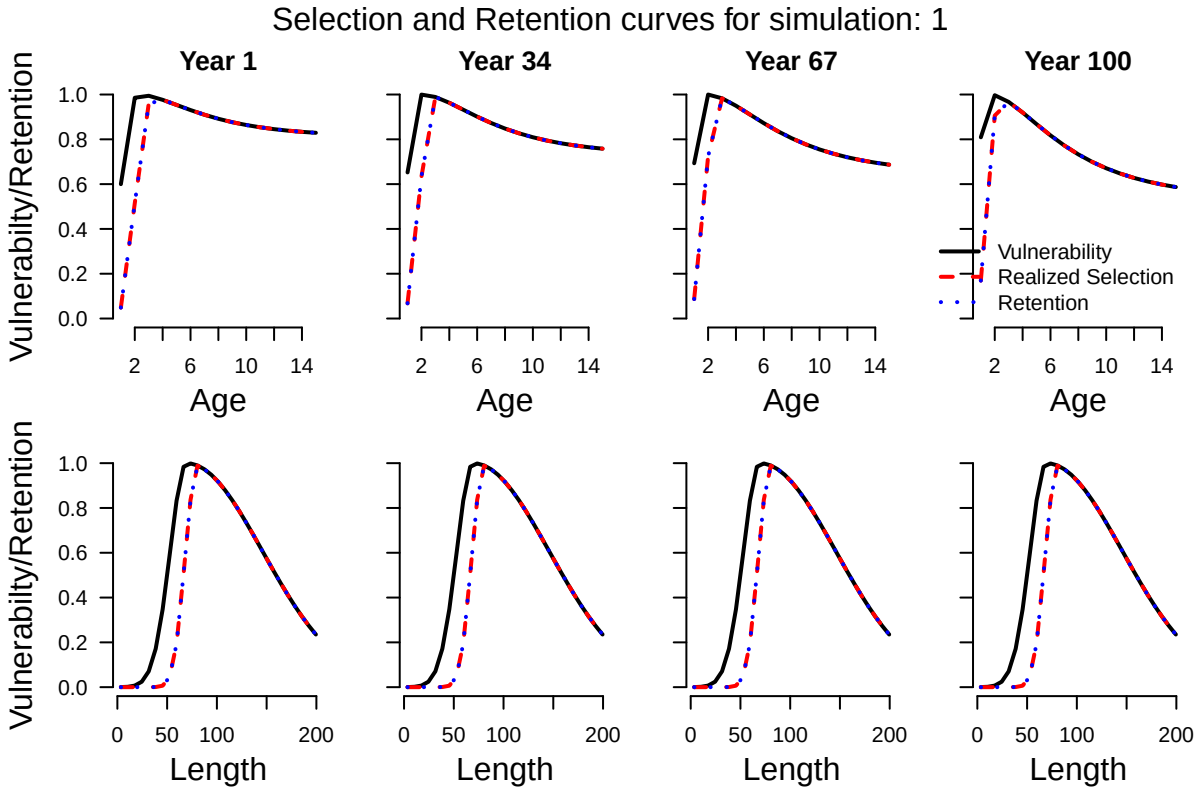
The retention curve can be modified by providing values for these slots:

```
OM@LR5 <- c(0.6, 0.7)
OM@LFR <- c(0.9, 1)
```

Note that the values in the `LR5` and `LFR` slots must be in the same units as those in the `L5` and `LFS` slots. Here we are specifying the values relative to the size of maturity, and assuming that the fishery discards the smaller sized fish:

```
plotSelect(OM, sim=1)
```

```
## Optimizing for user-specified movement
```



The plot shows that the retention curve for the fishery has shifted to the right, towards larger and older fish, while the vulnerability of the fishing gear remains the same.

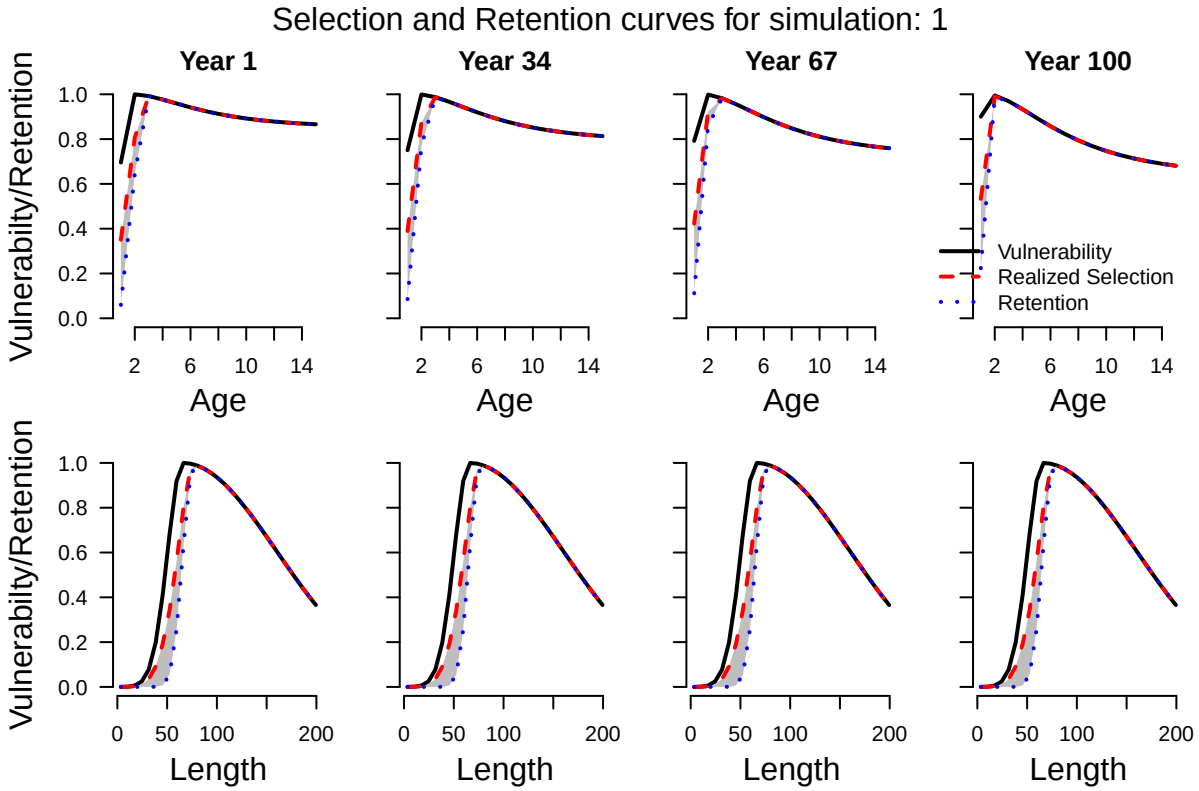
Because we are assuming no discard mortality in this case, the realized selection and retention curves are equivalent. This means that although fish of age/length between the vulnerability and retention curves are selected by the fishery, they are discarded with 100% survival and therefore are not removed from the population.

21.3.3 Discard Mortality

The assumption of 100% survival of discarded fish may be unrealistic in many situations. Discard mortality can be specified by the `Fdisc` slot in the `Stock` or `OM` object. The `Fdisc` slot represents the fraction of discarded fish that die, or 1 − survival. Here we assume that between 30 and 50% of discarded fish suffer fishing mortality:

```
OM@Fdisc <- c(0.3, 0.5)
plotSelect(OM, sim=1)
```

```
## Optimizing for user-specified movement
```



We can see now that the realized selection and the retention curves are different for the age/size classes that are discarded by the fishery. The realized selection curve (dashed red line) represents the actual selectivity of the fish removed from the population.

The retention curve (dotted blue line) shows the age/size classes that are retained by the fishery and appear in the total cathc, catch-at-age, and catch-at-length fishery data.

The shaded gray area between these two curves represents that age/size classes that are caught and killed by the fishery but are discarded and do not appear in the catch statistics.

The gear vulnerability curve remains unchanged, and shows that some individuals in the smaller age/size classes are caught and discarded alive back into the population.

21.3.4 General Discarding

General discarding across all age or size classes can be included using the discarding rate slot `DR` in the `Fleet` or `OM` object.

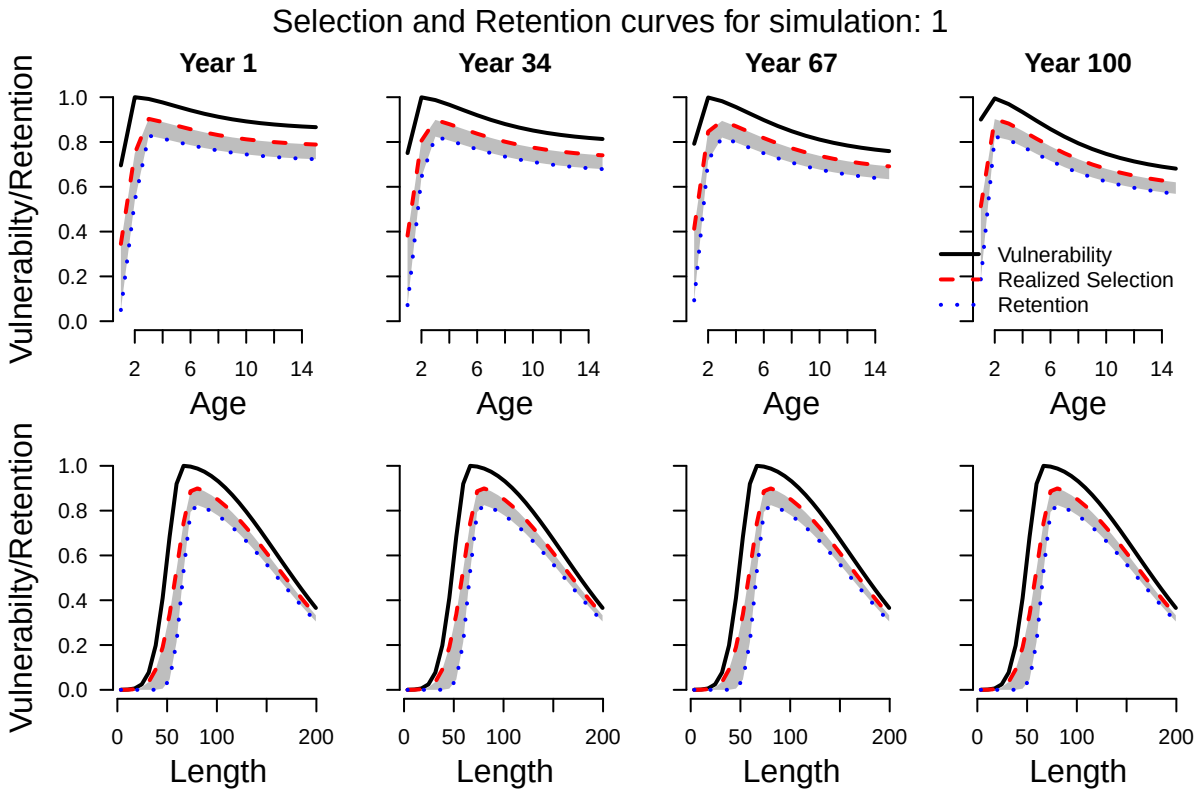
For example, here we assume that between 10 and 20% of all age/size classes are discarded by the fishery:

```
OM@DR <- c(0.1, 0.2)
```

Plotting the selectivity and retention curves shows that a proportion of all age and size classes are now discarded, with the survival rate determined by the `Fdisc` parameter:

```
plotSelect(OM, sim=1)
```

```
## Optimizing for user-specified movement
```



Chapter 22

Developing Custom Management Procedures

DLMtool was designed to be extensible in order to promote the development of new Management Procedures. In this chapter we design a series of new Management Procedures that include spatial controls and input controls in the form of size limit restrictions.

If you wish, you can also add your newly developed MPs to the DLMtool package so they are accessible to other uses. Of course you will be credited as the author. Please contact us for details how to do this.

As we saw before, real data are stored in a class of objects `Data`.

The DLMtool MSE function generates simulated data and puts it in exactly the same format as real data. This is highly desirable because it means that the same MP code that is tested in the MSE can then be used to make management recommendations.

If an MP is coded incorrectly it may catastrophically fail MSE testing and will therefore be excluded from use in management.

22.1 The Anatomy of an MP

Let's examine an existing output MP to identify the MP data requirements.

```
avail('Output')
```

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

Since we've seen it used as a default MP in lots of the examples above, lets learn more about DCAC

```
?DCAC
```

We can even see all the code for this MP by simply typing the name of the MP into the console (this is a fantastic advantage of using R - there is complete transparency about package functions):

```
DCAC
```

```
## function (x, Data, reps = 100)
## {
##     dependencies = "Data@AvC, Data@t, Data@Mort, Data@CV_Mort, Data@FMSY_M, Data@CV_FMSY_M, Data@Dt,
##     C_tot <- Data@AvC[x] * Data@t[x]
##     Mdb <- trlnorm(reps, Data@Mort[x], Data@CV_Mort[x])
##     FMSY_M <- trlnorm(reps, Data@FMSY_M[x], Data@CV_FMSY_M[x])
##     Bt_K <- trlnorm(reps, Data@Dt[x], Data@CV_Dt[x])
##     if (any(is.na(c(Data@BMSY_B0[x], Data@CV_BMSY_B0[x]))))
##         return(NA)
##     BMSY_K <- rbeta(reps, alphaconv(Data@BMSY_B0[x], Data@BMSY_B0[x] *
##         Data@CV_BMSY_B0[x]), betaconv(Data@BMSY_B0[x], Data@BMSY_B0[x] *
##         Data@CV_BMSY_B0[x]))
##     Rec <- new("Rec")
##     Rec@TAC <- TACfilter(C_tot/(Data@t[x] + ((1 - Bt_K)/(BMSY_K *
##         FMSY_M * Mdb))))
##     Rec
## }
## <bytecode: 0x000000001fd8ebd0>
## <environment: namespace:DLMtool>
## attr(,"class")
## [1] "MP"
```

"Crikey that looks complicated!" might be your first reaction. However this output MP function is easily demystified.

Like all MPs it has three arguments: `x`, `Data` and `reps`.

The argument `x` is the position in the Data object. When real data are stored in a Data object, there is only one position - there is only one real data set.

However, in MSE we conduct many simulations and x refers to simulated data from simulation number `x`. Any single parameters such as natural mortality rate (`Mort`) are a vector (`nsim` long). See `Data@Mort[x]` in the DCAC code. Any time series such as annual catches or relative abundance indices, are a matrix of `nsim` rows and `nyears` columns.

A range of objects of class Data are available:

```
avail('Data')
```

```
##  [1] "Atlantic_mackerel"  "China_rockfish"     "Cobia"
##  [4] "Example_datafile"   "Gulf_blue_tilefish" "ourReefFish"
##  [7] "Red_snapper"        "SimulatedData"      "Simulation_1"
## [10] "China_rockfish2"    "Data"               "Madeup"
## [13] "Recs"
```

For simplicity lets use a Data object with just one simulation, `Simulation_1` and rename it `Data`

```
Data <- Simulation_1
```

Since there is only one simulation in this data set (1 position) we can now see a single value of natural mortality rate:

```
Data@Mort
```

```
## [1] 0.2244735
```

And a matrix of catches with only 1 row:

```
Data@Cat
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
## [1,] 4.275057 12.43761 14.63192 35.31725 28.69802 30.84651 24.14059
##          [,8]     [,9]    [,10]    [,11]    [,12]    [,13]    [,14]
## [1,] 36.78335 29.27517 38.18088 59.30242 56.08995 37.96849 51.84985
##         [,15]    [,16]    [,17]   [,18]   [,19]    [,20]    [,21]    [,22]
## [1,] 60.76729 41.53713 39.31114 57.9673 57.2248 72.37596 80.69301 76.63558
##         [,23]    [,24]    [,25]    [,26]    [,27]    [,28]    [,29]  [,30]
## [1,] 59.37687 48.82643 50.38803 80.71158 53.35875 74.31955 48.83082 43.348
##         [,31]    [,32]    [,33]    [,34]    [,35]    [,36]    [,37]
## [1,] 65.17864 49.94281 47.38492 45.23197 80.92438 51.91477 29.36491
##         [,38]    [,39]    [,40]    [,41]    [,42]    [,43]    [,44]
## [1,] 43.44834 49.46923 50.33217 49.53639 39.28779 27.31767 38.97092
##        [,45]    [,46]    [,47]    [,48]    [,49]    [,50]
## [1,] 51.1054 37.34677 37.33128 24.24094 23.47756 21.08158
```

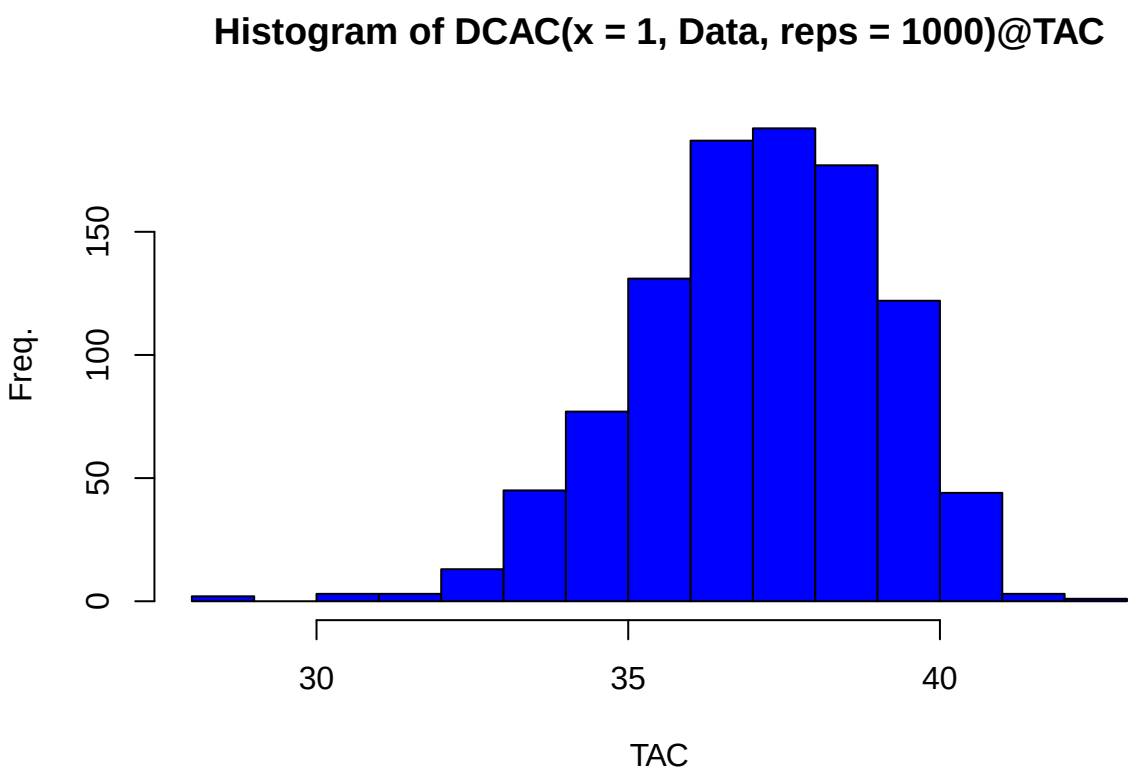
We could generate a single TAC recommendation from these data using DCAC by specifying position 1 (there is only 1 simulation) and by setting reps=1 (we want a single DCAC TAC recommendation)

```
DCAC(x=1,Data,reps=1)
```

```
## TAC (median)
##      37.8473
```

If we wanted a stochastic estimate of the TAC we could increase the number of reps:

```
hist(DCAC(x=1,Data,reps=1000)@TAC,xlab="TAC",ylab="Freq.",col="blue")
```



22.2 A Constant Catch MP

We've now got a better idea of the anatomy of an MP. It is a function that must accept three arguments:

- x: a simulation number
- Data: an object of class `Data`
- reps: the MP can provide a sample of TACs `reps` long.

Let's have a go at designing our own custom MP that can work with DLMtool. We're going to develop an MP that sets the TAC as the '3rd highest catch'.

We decide to call our function `THC`

```
THC<-function(x, Data, reps){

  # Find the position of third highest catch

  THCpos<-order(Data@Cat[x,],decreasing=T)[3]

  # Make this the mean TAC recommendation

  THCmu<-Data@Cat[x,THCpos]

  # A sample of the THC is taken according to a fixed CV of 10%
  TACs <- THCmu * exp(rnorm(reps, -0.1^2/2, 0.1)) # this is a lognormal distribution
```

```
  Rec <- new("Rec") # create a 'Rec'object
  Rec@TAC <- TACs # assign the TACs to the TAC slot
  Rec # return the Rec object
}
```

To recap that's just seven lines of code:

```
THC<-function(x, Data, reps){
  THCpos<-order(Data@Cat[x,],decreasing=T)[3]
  THCmu<-Data@Cat[x,THCpos]
  Rec <- new("Rec")
  Rec@TAC <- THCmu * exp(rnorm(reps, -0.1^2/2, 0.1))
  Rec
}
```

We can quickly test our new MP for the example Data object

```
THC(x=1,Data,reps=10)@TAC
```

```
##  [1] 71.52528 86.93342 93.19337 84.81037 83.08070 71.91512 82.12945
##  [8] 75.41995 84.72235 85.65570
```

Now that we know it works, to make the function compatible with the DLMtool package we have to assign it the class 'MP' so that DLMtool recognizes the function as a management procedure

```
class(THC)<-"MP"
```

If we want to run the MSE in parallel we need to export the newly created function to the cluster:

```
sfExport('THC')
```

22.3 A More Complex MP

The THC MP is simple and frankly not a great performer (depending on depletion, life-history, adherence to TAC recommendations).

Let's innovate and create a brand new MP that could suit a catch-data-only stock like Indian Ocean Longtail tuna!

It may be possible to choose a single fleet and establish a catch rate that is 'reasonable' or 'fairly productive' relative to current catch rates. This could be for example, 40% of the highest catch rate observed for this fleet or, for example, 150% of current cpue levels.

It is straightforward to design an MP that will aim for this target index level by making adjustments to the TAC.

We will call this MP `TCPUE`, short for target catch per unit effort:

```
TCPUE<-function(x,Data,reps){

  mc<-0.05                              # max change in TAC
  frac<-0.3                             # target index is 30% of max
  nyears<-length(Data@Ind[x,])          # number of years of data

  smoothI<-smooth.spline(Data@Ind[x,]) # smoothed index
  targetI<-max(smoothI$y)*frac          # target

  currentI<-mean(Data@Ind[x,(nyears-2):nyears]) # current index
```

```
  ratio<-currentI/targetI              # ratio currentI/targetI

  if(ratio < (1 - mc)) ratio <- 1 - mc # if currentI < targetI
  if(ratio > (1 + mc)) ratio <- 1 + mc # if currentI > targetI

  Rec <- new("Rec")
  Rec@TAC <-  Data@MPrec[x] * ratio * exp(rnorm(reps, -Data@CV_Ind[x]^2/2, Data@CV_Ind[x]))
  Rec
}
```

The TCPUE function simply decreases the past TAC (stored in `Data@MPrec`) if the index is lower than the target and increases the TAC if the index is higher than the target.

All that is left is to make it compatible with DLMtool:

```
class(TCPUE)<-"MP"
sfExport("TCPUE")
```

22.4 Beyond the Catch Limit

All management procedures return an object of class 'Rec' that contains 13 slots:

```
slotNames("Rec")
```

```
##  [1] "TAC"      "Effort"   "Spatial"  "Allocate" "LR5"      "LFR"
##  [7] "HS"       "Rmaxlen"  "L5"       "LFS"      "Vmaxlen"  "Fdisc"
## [13] "Misc"
```

We've already seen the TAC slot in the previous exercise. The remaining slots relate to various forms of input control:

- Effort (total allowable effort (TAE) relative to last historical year)
- Spatial - Fraction of each area that is open
- Allocate - Allocation of effort from closed areas to open areas
- LR5 - Length at 5% retention
- LFR - Length at 100% retention
- HS - Upper slot limit
- Rmaxlen - Retention of the maximum length class
- L5 - Length at 5% selection (e.g a change in gear type)
- LFS - Length at 100% selection (e.g a change in gear type)
- Vmaxlen - Selectivity of the maximum length class
- Fdisc - Update the discard mortality if required
- Misc - An optional slot for storing additional information

The curE MP just keeps effort constant at current levels:

```
curE
```

```
## function (x, Data, ...)
## {
##     rec <- new("Rec")
##     rec@Effort <- 1
##     rec
## }
## <environment: namespace:DLMtool>
## attr(,"class")
```

```
## [1] "MP"
```

Note that only the `Effort` slot in the `Rec` object is populated in this case.

To highlight the differences among Input control MPs examine spatial control MP `MRreal` that closes area 1 to fishing and reallocates fishing to the open area 2:

```
MRreal
```

```
## function (x, Data, ...)
## {
##     rec <- new("Rec")
##     rec@Allocate <- 1
##     rec@Spatial <- c(0, rep(1, Data@nareas - 1))
##     return(rec)
## }
## <environment: namespace:DLMtool>
## attr(,"class")
## [1] "MP"
```

In contrast 'MRnoreal' does not reallocate fishing effort:

```
MRnoreal
```

```
## function (x, Data, ...)
## {
##     rec <- new("Rec")
##     rec@Allocate <- 0
##     rec@Spatial <- c(0, rep(1, Data@nareas - 1))
##     return(rec)
## }
## <environment: namespace:DLMtool>
## attr(,"class")
## [1] "MP"
```

The MP `matlenlim` only specifies the parameters of length retention using an estimate of length at 50% maturity (`Stock@L50`):

```
matlenlim
```

```
## function (x, Data, ...)
## {
##     dependencies = "Data@L50"
##     rec <- new("Rec")
##     rec@LR5 <- Data@L50[x] * 0.95
##     rec@LFR <- Data@L50[x]
##     rec
## }
## <environment: namespace:DLMtool>
## attr(,"class")
## [1] "MP"
```

22.4.1 An Example Effort Control

Here we will copy and modify the MP we developed earlier to specify a new version of the target catch per unit effort MP (`TCPUE`) that provides effort recommendations:

```
TCPUE_e<-function(x,Data,reps){

  mc<-0.05                              # max change in TAC
  frac<-0.3                             # target index is 30% of max
  nyears<-length(Data@Ind[x,])          # number of years of data

  smoothI<-smooth.spline(Data@Ind[x,]) # smoothed index
  targetI<-max(smoothI$y)*frac          # target

  currentI<-mean(Data@Ind[x,(nyears-2):nyears]) # current index

  ratio<-currentI/targetI               # ratio currentI/targetI

  if(ratio < (1 - mc)) ratio <- 1 - mc # if currentI < targetI
  if(ratio > (1 + mc)) ratio <- 1 + mc # if currentI > targetI

  rec <- new("Rec")
  rec@Effort <- Data@MPeff[x] * ratio
  rec

}
```

There have been surprisingly few changes to make TCPUE an input control MP that sets total allowable effort.

1. We have had to use stored recommendations of effort in the `Data@MPeff` slot, and
2. The final line of the MP is our input control recommendatation that only modified the Effort.

That is all. Again, we need to assign our new function to class `MP` and export it to the cluster:

```
class(TCPUE_e)<-"MP"
sfExport('TCPUE_e')
```

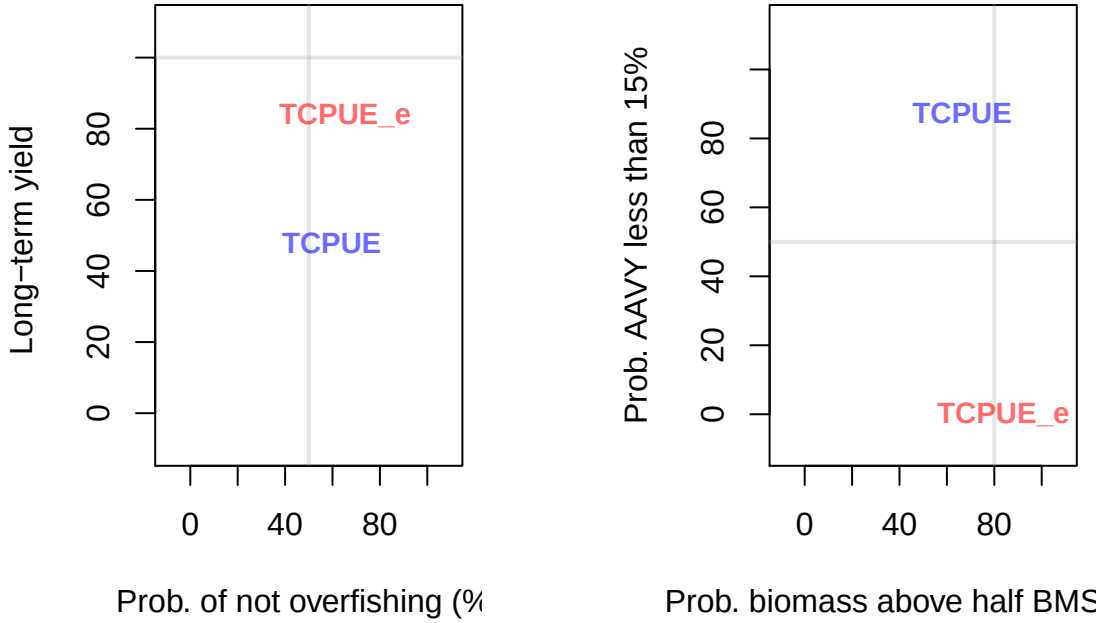
Let's test the two MPs and see how they peform:

```
testMSE<-runMSE(testOM,MPs=c("TCPUE","TCPUE_e"), parallel = TRUE)
```

```
## Running MSE in parallel on 4 processors
```

```
## MSE completed
```

```
NOAA_plot(testMSE)
```




```
##         PNOF  B50  LTY   VY
## TCPUE   59.7 66.5 47.9 87.5
## TCPUE_e 65.3 83.8 83.8  0.0
```

Chapter 23

Custom Parameters

By default, DLMtool samples the operating model parameters from a uniform distribution. Because the parameters are sampled independently, it is not possible to generate correlated samples. However, the `cpars` slot in the `OM` object can be used to pass custom samples into the MSE.

The addition of the `cpars` slot provides a lot of flexibility to the DLMtool, and allows users full control of all parameters used in the model. For example, it is possible to generate operating models directly from the output of common stock assessment packages using functions in DLMtool (e.g `SS2DLM` for Stock Synthesis 3, and `iSCAM2DLM` for a iSCAM model). These functions take the correlated parameter values from the output of the stock assessment and provide them to DLMtool via the `cpars` slot, resulting in an operating model that is conditioned on the stock assessment.

The `cpars` feature is being continually developed as more features are requested for DLMtool.

23.1 Valid cpars names

The `cpars` slot requires a named list containing the custom parameter values. You can see the valid names for `cpars` by typing:

```
validcpars()
```

```
##       [,1]          [,2]           [,3]        [,4]           [,5]
##  [1,] "Abias"       "AC"           "Aerr"      "age95"        "ageM"
##  [2,] "Asize"       "betas"        "Brefbias"  "CAA_ESS"      "CAA_nsamp"
##  [3,] "CAL_bins"    "CAL_binsmid"  "CAL_ESS"   "CAL_nsamp"    "Cbias"
##  [4,] "Crefbias"    "Csd"          "D"         "Dbias"        "Derr"
##  [5,] "dFfinal"     "DR"           "EffLower"  "EffUpper"     "EffYears"
##  [6,] "Esd"         "Fdisc"        "Find"      "FMSY_Mbias"   "Frac_area_1"
##  [7,] "hs"          "initdist"     "Irefbias"  "Isd"          "K"
##  [8,] "Karray"      "Kbias"        "Kgrad"     "Krand"        "Ksd"
##  [9,] "L5"          "L50"          "L50_95"    "L95"          "LatASD"
## [10,] "Len_age"     "LenCV"        "lenMbias"  "LFCbias"      "LFR"
## [11,] "LFS"         "LFSbias"      "Linf"      "Linfarray"    "Linfbias"
## [12,] "Linfgrad"    "Linfrand"     "Linfsd"    "LR5"          "M"
## [13,] "M_ageArray"  "M_at_Length"  "Marray"    "Mat_age"      "maxage"
## [14,] "Mbias"       "Mgrad"        "mov"       "Mrand"        "Msd"
## [15,] "Perr"        "Prob_staying" "procsd"    "qcv"          "qinc"
## [16,] "R0"          "Recsd"        "Rmaxlen"   "Size_area_1"  "Spat_targ"
## [17,] "t0"          "t0bias"       "V"         "Vmaxlen"      "Wt_age"
```

A warning message will alert you if variables appear in the named `cpars` list that are not in `validcpars()`, and these will be ignored in the MSE.

23.2 Correlated samples

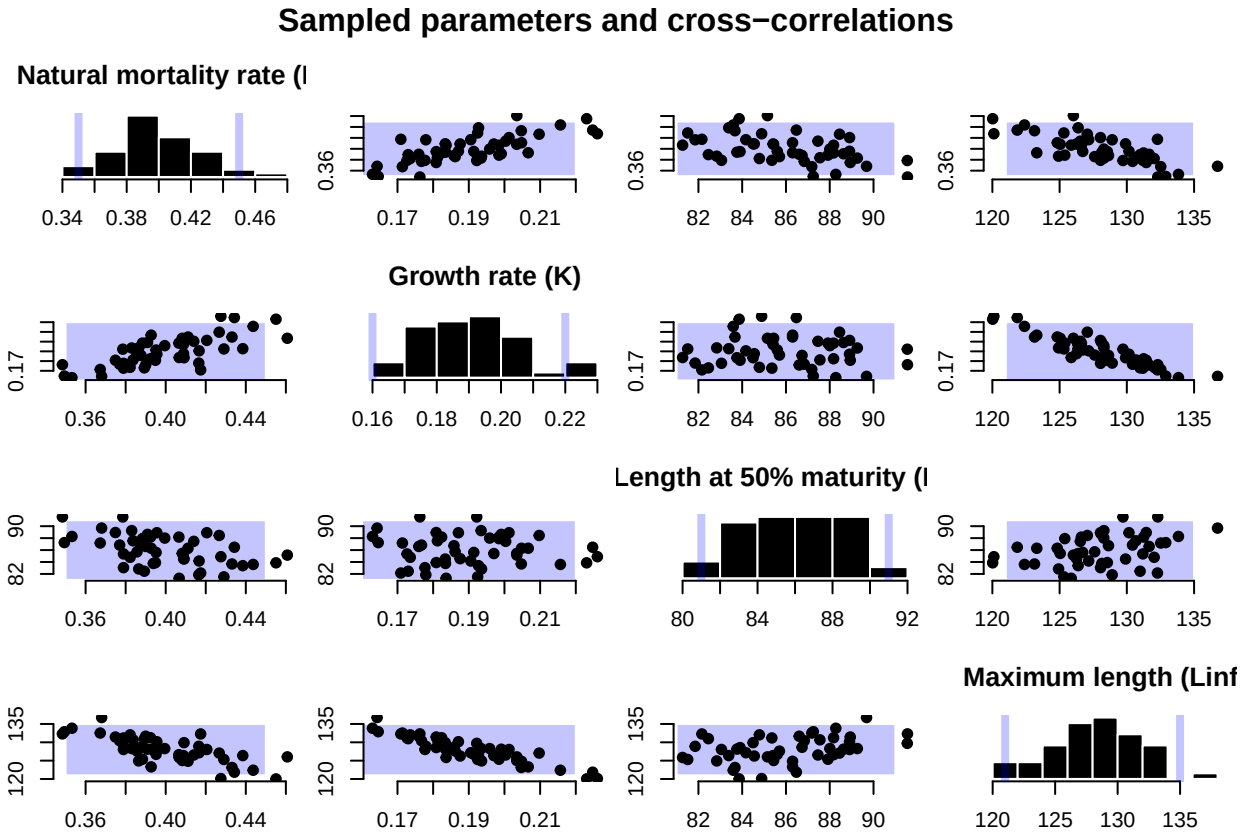
As the `cpars` function is used to provide correlated samples to the MSE, it is important that the same number of custom parameters are provided for each variable. In most cases, this is simply a vector `nsim` long.

For example, if you wish to supply correlated samples of the von Bertalanffy growth parameters, you would create three vectors of length `nsim` containing the samples of Linf, K, and t0.

If the vectors are shorter than `nsim` they will simply be recycled. An error message will alert you if the vectors are not the same length.

As a demonstration, we will use the `ForceCor` function to generate correlated samples of *M*, *K*, *L50*, and *Linf* and examine the `cpars` slot in the resulting OM object:

```
OM <- ForceCor(DLMtool::testOM)
```




```
str(OM@cpars)
```

```
## List of 4
##  $ M   : num [1:48] 0.382 0.39 0.396 0.353 0.4 ...
##  $ K   : num [1:48] 0.174 0.176 0.187 0.163 0.196 ...
##  $ L50 : num [1:48] 84.8 86.6 88.9 88.3 88 ...
##  $ Linf: num [1:48] 132 132 130 134 128 ...
```

You can see that the `OM@cpars` slot is a list of length 4 and contains named vectors with 48 correlated samples of the four parameters.

Because the `OM@cpars` slot contains these values, the *M*, *K*, *L50*, and *Linf* values in the OM, e.g. `OM@M` will be ignored.

Any additional custom parameters can be added to `cpars` using this same approach. For example, to provide custom (in this case uncorrelated) samples of *t0*:

```
OM@cpars$t0 <- runif(OM@cpars, -1, 0)
str(OM@cpars)
```

```
## List of 5
##  $ M   : num [1:48] 0.382 0.39 0.396 0.353 0.4 ...
##  $ K   : num [1:48] 0.174 0.176 0.187 0.163 0.196 ...
##  $ L50 : num [1:48] 84.8 86.6 88.9 88.3 88 ...
##  $ Linf: num [1:48] 132 132 130 134 128 ...
##  $ t0  : num [1:4] -0.768 -0.759 -0.203 -0.168
```

23.3 Custom time-varying parameters

It is also possible to supply custom generated time-varing values for some parameters using the `cpars` slot. For example, time-varying natural mortality or selectivity patterns.

Additional details on using the `cpars` slot for this will be added to the userguide soon. If you find that this is a feature you wish to use but are unclear how to do it, bug us with an email!

Chapter 24

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.

24.1 Subsetting by Performance

For example, we may wish to include only Management Procedures that have greater than 70% probability that the biomass is above $0.5B_{MSY}$:

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

```
stats <- summary(BSharkMSE) # save summary object to `stats`
```

```
## Calculating Performance Metrics
```

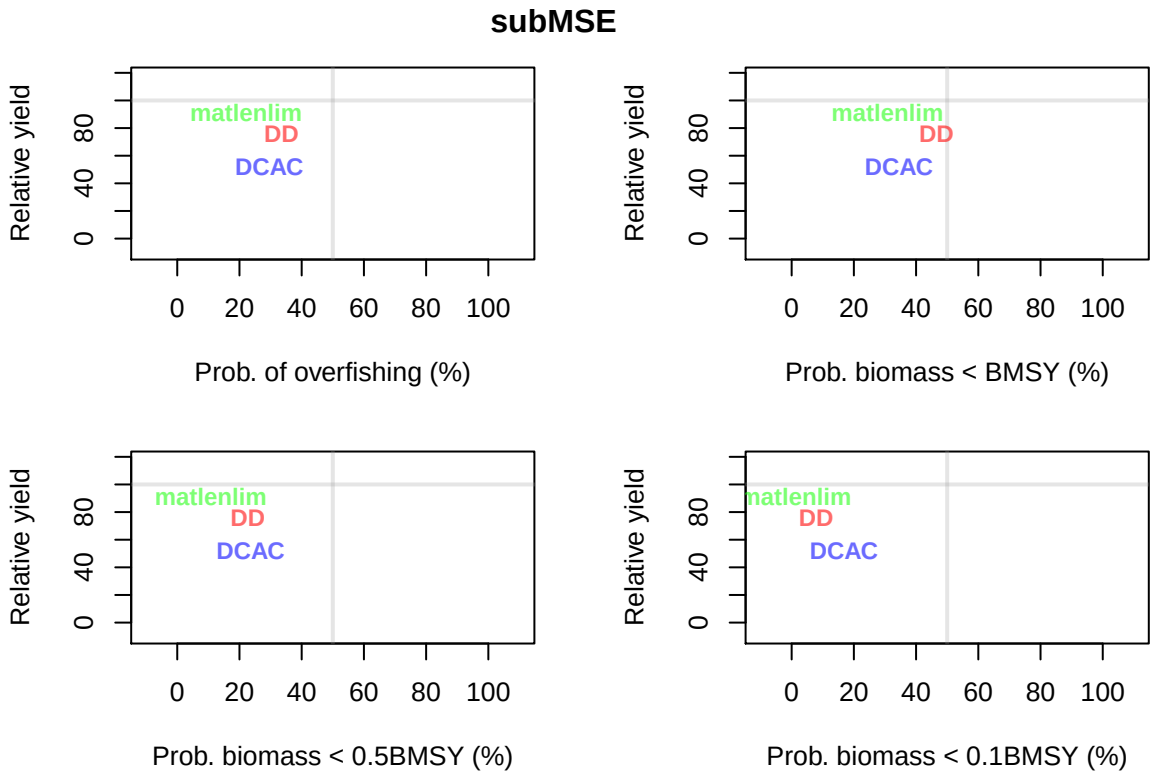
```
##                                    Performance.Metrics
## 1                  Average Annual Variability in Yield
## 2  Average Long-Term Yield relative to Reference Yield
## 3          Probability Spawning Biomass above 10% BMSY
## 4                  Probability Spawning Biomass > BMSY
## 5          Probability Spawning Biomass above 50% BMSY
## 6                                  Probability F < FMSY
## 7 Average Short-Term Yield relative to Reference Yield
## 8                    Yield relative to Reference Yield
##
##
## Probability:
##          MP AAVY  LTY  P10 P100  P50  POF  STY Yield
## 1    Fratio 0.86 0.48 0.82 0.49 0.66 0.59 0.61  0.72
## 2      DCAC 0.96 0.48 0.83 0.65 0.76 0.70 0.67  0.67
## 3      Fdem 0.78 0.47 0.76 0.43 0.59 0.51 0.62  0.71
## 4        DD 0.96 0.80 0.92 0.53 0.77 0.67 0.68  0.86
## 5 matlenlim 0.34 0.82 0.99 0.69 0.89 0.78 0.71  0.95
```

```
accept <- which(stats$P50 > 0.70) # 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, DD, matlenlim methods (3 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)
```



24.2 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] 100
```

```
aboveMSE@nsim
```

```
## [1] 100
```

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.

Chapter 25

Custom Performance Metrics

25.1 The PM Object

> This section is under development. More detail coming soon. Please bug us with emails if it is taking too long!

A newly developed feature of DLMtool is the `PM` (performance metric) object. There are several `PM` objects in DLMtool.

```
avail("PM")
```

```
## [1] "AAVY"  "LTY"   "P10"   "P100"  "P50"   "POF"   "STY"   "Yield"
```

The help documentation for the `PM` objects is still under development:

```
?P100
```

The `PM` objects are designed to be applied to an object of class `MSE`:

```
P100(BSharkMSE)
```

```
## Probability Spawning Biomass > BMSY
##  Prob. SB > SBMSY
##     Fratio DCAC Fdem   DD matlenlim
## 1    0.06 0.88 0.06    1      0.42
## 2     0.9 0.92  0.9  0.9       0.9
## 3    0.14    1 0.08 0.38         1
## 4    0.76    0 0.78 0.32      0.62
## 5    0.84    0 0.84    0      0.84
## 6       0 0.38    0 0.64         0
## 7       0 0.48    0  0.3         0
## 8    0.02 0.22 0.04 0.38       0.6
## 9       0 0.98    0 0.42      0.92
## 10   0.94 0.94 0.64 0.52         0
## 11      .    .    .    .         .
## 12      .    .    .    .         .
## 13      .    .    .    .         .
## 200     1    1    1    1         1
##
## Mean
## [1] 0.49 0.65 0.43 0.53 0.69
```

25.2 Developing Custom PMs

Users can develop their own `PM` objects.

> This section is under development. More detail coming soon. Please bug us with emails if it is taking too long!

Appendix A

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
- Fisheries and Oceans, Canada (DFO)
- United Nations Food & Agriculture Organization (FAO)

Appendix B

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.

Lorenzen, K. (1996), The relationship between body weight and natural mortality in juvenile and adult fish: a comparison of natural ecosystems and aquaculture. Journal of Fish Biology, 49: 627–642

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.

Appendix C

Getting Help

C.1 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.

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.

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.

C.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.

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.

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).

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")
```

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.

C.3 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.

C.4 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
```

You can access this user guide at any time from the R console:

```
userguide()
```

C.5 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 questions that don't demonstrate a proper search for already posted answers to the problem.

Appendix D

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.

D.1 Biology

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.

Density-Dependent Compensation

DLMtool assumes that, with the exception of the stock-recruitment relationship, 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.

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. Since DLMtool V4.4 it is possible to use alternative length-at-age models by using `cpars`. See the Custom Parameters chapter for more information.

Natural Mortality Rate at Age

By default DLMtool assumes that natural mortality (M) is constant with age and size. Since DLMtool V4.4 size or age-specific M can be specified. See the Size-Specific Natural Mortality chapter for more information.

D.2 MSE Model Assumptions

Retention and Selectivity

The OM has slots for both gear selectivity and retention by size. If the retention slots are not populated, it is assumed that retention = selectivity, that is, all fish that are captured by the gear are retained by the fishers.

Most size-regulation MPs (e.g., `matlenlim`) change the retention pattern and leave the selectivity pattern unchanged. For example, if a size limited is regulated well above the current size of selection, fish smaller than the size limit are still caught by the gear but are discarded and may suffer some fishing mortality (Stock@Fdisc).

MPs can be designed to modify gear selectivity instead of, or in addition to, the retention-at-size.

Non-Convergence of Management Procedure

In some cases during the MSE Management Procedure may not be able to successfully calculate a management recommendation from the simulated data. For example, a catch-curve may used to estimate Z, and F is calculated as $F = Z - M$. Because of process and observation error, it is possible that the estimated F is negative, in which case the MP may fail to calculate a recommended catch limit.

The Management Procedures have been designed to return `NA` if they fail to calculate a management recommendation for any reason. In this case, the management recommendations from the previous year are used in the simulation, e.g., $\text{TAC}_y = \text{TAC}_{y-1}$.

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.

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.

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.

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.

D.3 Management Procedures

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.

D.4 Data and Method Application

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.

Appendix E

Changes

Important changes - DLMtool V4.1 and greater

DLMtool V4.1 introduced some important changes to the Operating Model object. The number of simulations (`nsim`) and the number of projection years (`proyears`) are now slots in the `OM` object, rather than arguments to `runMSE` (see Management Strategy Evaluation). This change was required to allow users to specify their own custom futures for parameters like M, growth, etc. The `OM` object also now has a new random seed slot in the operating model, which ensures that the MSE results are now exactly reproducible.

You can modify the number of simulations, the number of projection years, or the value of the random seed by modifiying the relevant slots in the `OM` object:

- `OM@nsim`
- `OM@proyears`
- `OM@seed`

Important changes - DLMtool V4.5 and greater

Since DLMtool V5.0 the following slots have been added to the OM object:

- `OM@interval`
- `OM@pstar`
- `OM@maxF`
- `OM@reps`

This was done so that an OM object is completely self-contained and includes all information used in the MSE.