Data-Limited Methods Toolkit (DLMtool 5.2)

User Guide

Tom Carruthers & Adrian Hordyk

2018-07-09

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 110 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          5.0
## year           2018
## month          04
## day            23
## svn rev        74626
## language       R
## version.string R version 3.5.0 (2018-04-23)
## nickname       Joy in Playing
```

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

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

```
## [1] '5.2'
```

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 is available for download from CRAN.

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          5.0
## year           2018
## month          04
## day            23
## svn rev        74626
## language       R
## version.string R version 3.5.0 (2018-04-23)
## nickname       Joy in Playing
```

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:

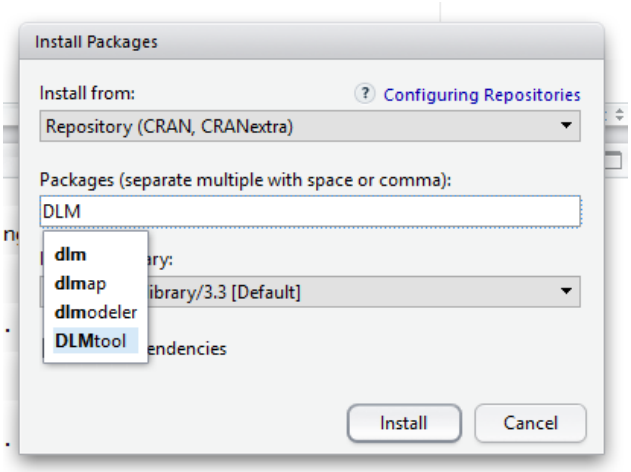
```
install.packages("DLMtool")
```

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] '5.2'
```

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:

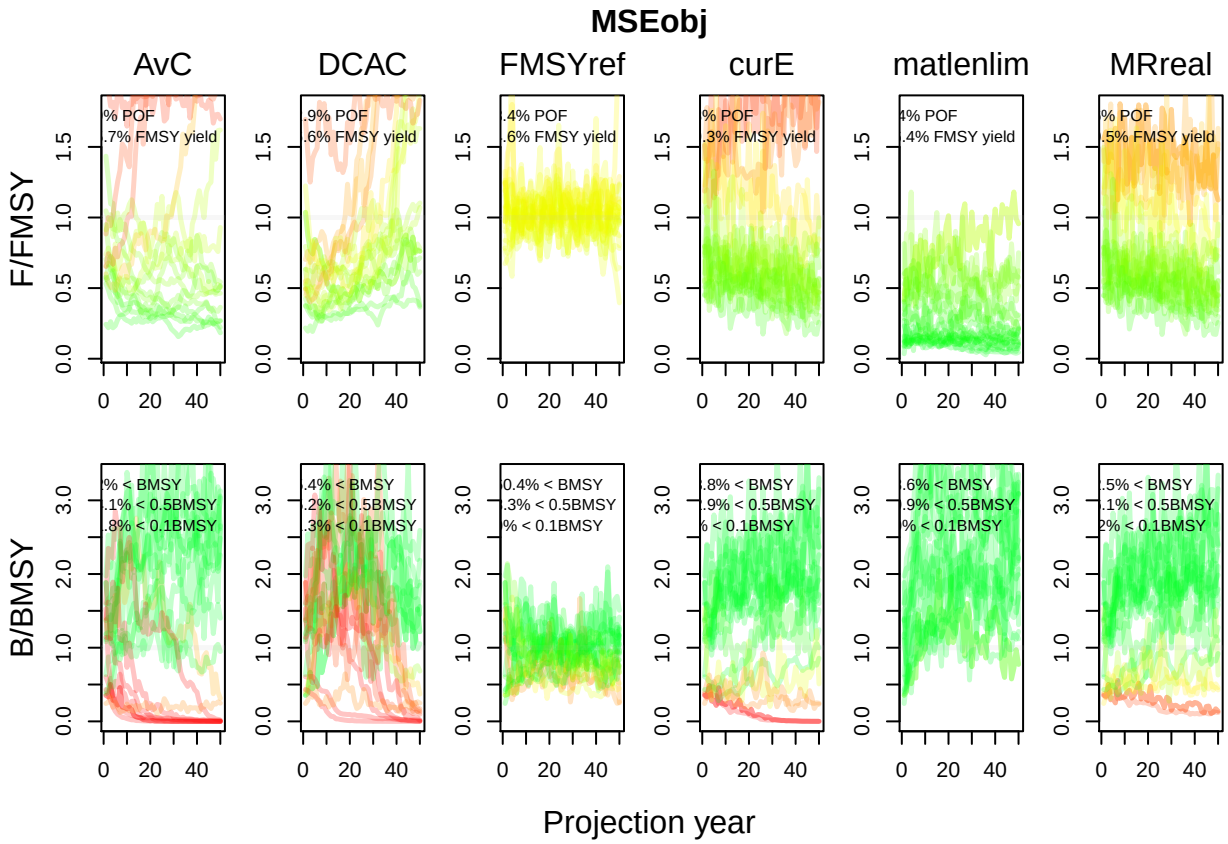
```
myMSE <- runMSE()
```

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                  Probability of not overfishing (F<FMSY)
## 2                       Spawning Biomass relative to SBMSY
## 3         Average Annual Variability in Yield (Years 1-50)
## 4 Average Yield relative to Reference Yield (Years 41-50)
##
## 1               Prob. F < FMSY (Years 1 - 50)
## 2         Prob. SB > 0.5 SBMSY (Years 1 - 50)
## 3               Prob. AAVY < 20% (Years 1-50)
## 4 Prob. Yield > 0.5 Ref. Yield (Years 41-50)
##
##
## Probability:
##          MP PNOF  P50 AAVY  LTY
## 1       AvC 0.60 0.68 0.90 0.39
## 2      DCAC 0.66 0.78 0.96 0.65
## 3   FMSYref 0.42 0.93 1.00 1.00
## 4      curE 0.68 0.80 0.17 0.80
## 5 matlenlim 0.98 0.99 0.19 0.56
## 6    MRreal 0.72 0.85 0.19 0.82
```

Later sections of the user manual will describe more ways to evaluate the outputs of the `runMSE` function.

But first we will look at the most fundamental part of MSE: the *Operating Model*.

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* (Observation) 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* (Implementation) 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 Plotting OM Components

The OM Components *Stock*, *Fleet*, *Obs*, and *Imp* can be plotted to visually examine the contents.

For example, to plot a *Stock* object (note that the figures are not shown here):

```
plot(Albacore)
```

To plot a *Fleet* object you must also provide an object of class *Stock*, for example:

```
plot(FlatE_Dom, Albacore)
```

The *Obs* and *Imp* objects can also be plotted:

```
plot(Generic_Obs)
plot(Overages)
```

4.3 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
```

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.

4.4 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 Procedures

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"          "CC2"          "CC3"
##   [9] "CC4"          "CC5"          "CompSRA"      "CompSRA4010"
##  [13] "curE"         "curE75"       "DAAC"         "DBSRA"
##  [17] "DBSRA_40"     "DBSRA4010"    "DCAC"         "DCAC_40"
##  [21] "DCAC_ML"      "DCAC4010"     "DCACs"        "DD"
##  [25] "DD4010"       "DDe"          "DDe75"        "DDes"
##  [29] "DepF"         "DTe40"        "DTe50"        "DynF"
```

```
##  [33] "EtargetLopt"  "Fadapt"       "Fdem"         "Fdem_CC"
##  [37] "Fdem_ML"      "FMSYref"      "FMSYref50"    "FMSYref75"
##  [41] "Fratio"       "Fratio_CC"    "Fratio_ML"    "Fratio4010"
##  [45] "GB_CC"        "GB_slope"     "GB_target"    "Gcontrol"
##  [49] "HDAAC"        "ICI"          "ICI2"         "Iratio"
##  [53] "Islope1"      "Islope2"      "Islope4"      "IT10"
##  [57] "IT5"          "Itarget1"     "Itarget1_MPA" "Itarget2"
##  [61] "Itarget3"     "Itarget4"     "ItargetE1"    "ItargetE2"
##  [65] "ItargetE3"    "ItargetE4"    "ITe10"        "ITe5"
##  [69] "ITM"          "L95target"    "LBSPR"        "Lratio_BHI"
##  [73] "Lratio_BHI2"  "LstepCC1"     "LstepCC2"     "LstepCC3"
##  [77] "LstepCC4"     "LstepCE1"     "LstepCE2"     "Ltarget1"
##  [81] "Ltarget2"     "Ltarget3"     "Ltarget4"     "LtargetE1"
##  [85] "LtargetE4"    "matlenlim"    "matlenlim2"   "MCD"
##  [89] "MCD4010"      "minlenLopt1"  "MRnoreal"     "MRreal"
##  [93] "NFref"        "Rcontrol"     "Rcontrol2"    "SBT1"
##  [97] "SBT2"         "slotlim"      "SPmod"        "SPMSY"
## [101] "SPslope"      "SPSRA"        "SPSRA_ML"     "YPR"
## [105] "YPR_CC"       "YPR_ML"       "avgMP"        "TCPUE"
## [109] "TCPUE_e"      "THC"
```

As you can see, there are 110 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 Recs
## 1 DCAC Output  TAC
```

This tells us that `DCAC` is an Output control MP and returns a management recommendation in the form of a total allowable catch limit (TAC).

Here we list all available MPs:

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

```
##               MP      Type         Recs
## 1           curE     Input          TAE
## 2         curE75     Input          TAE
```

```
## 3            DDe     Input          TAE
## 4          DDe75     Input          TAE
## 5           DDes     Input          TAE
## 6          DTe40     Input          TAE
## 7          DTe50     Input          TAE
## 8    EtargetLopt     Input          TAE
## 9      ItargetE1     Input          TAE
## 10     ItargetE2     Input          TAE
## 11     ItargetE3     Input          TAE
## 12     ItargetE4     Input          TAE
## 13         ITe10     Input          TAE
## 14          ITe5     Input          TAE
## 15         LBSPR     Input          TAE
## 16      LstepCE1     Input          TAE
## 17      LstepCE2     Input          TAE
## 18     LtargetE1     Input          TAE
## 19     LtargetE4     Input          TAE
## 20     matlenlim     Input           SL
## 21    matlenlim2     Input           SL
## 22   minlenLopt1     Input           SL
## 23      MRnoreal     Input      Spatial
## 24        MRreal     Input      Spatial
## 25       slotlim     Input           SL
## 26       TCPUE_e     Input          TAE
## 27       AvC_MLL     Mixed      TAC, SL
## 28  Itarget1_MPA     Mixed TAC, Spatial
## 29           AvC    Output          TAC
## 30            BK    Output          TAC
## 31         BK_CC    Output          TAC
## 32         BK_ML    Output          TAC
## 33           CC1    Output          TAC
## 34           CC2    Output          TAC
## 35           CC3    Output          TAC
## 36           CC4    Output          TAC
## 37           CC5    Output          TAC
## 38       CompSRA    Output          TAC
## 39   CompSRA4010    Output          TAC
## 40          DAAC    Output          TAC
## 41         DBSRA    Output          TAC
## 42      DBSRA_40    Output          TAC
## 43     DBSRA4010    Output          TAC
## 44          DCAC    Output          TAC
## 45       DCAC_40    Output          TAC
## 46       DCAC_ML    Output          TAC
## 47      DCAC4010    Output          TAC
## 48         DCACs    Output          TAC
## 49            DD    Output          TAC
## 50        DD4010    Output          TAC
## 51          DepF    Output          TAC
## 52          DynF    Output          TAC
## 53        Fadapt    Output          TAC
## 54          Fdem    Output          TAC
## 55       Fdem_CC    Output          TAC
## 56       Fdem_ML    Output          TAC
```

```
## 57        Fratio    Output          TAC
## 58     Fratio_CC    Output          TAC
## 59     Fratio_ML    Output          TAC
## 60    Fratio4010    Output          TAC
## 61         GB_CC    Output          TAC
## 62      GB_slope    Output          TAC
## 63     GB_target    Output          TAC
## 64      Gcontrol    Output          TAC
## 65         HDAAC    Output          TAC
## 66           ICI    Output          TAC
## 67          ICI2    Output          TAC
## 68        Iratio    Output          TAC
## 69       Islope1    Output          TAC
## 70       Islope2    Output          TAC
## 71       Islope4    Output          TAC
## 72          IT10    Output          TAC
## 73           IT5    Output          TAC
## 74      Itarget1    Output          TAC
## 75      Itarget2    Output          TAC
## 76      Itarget3    Output          TAC
## 77      Itarget4    Output          TAC
## 78           ITM    Output          TAC
## 79    L95target    Output          TAC
## 80    Lratio_BHI    Output          TAC
## 81   Lratio_BHI2    Output          TAC
## 82     LstepCC1    Output          TAC
## 83     LstepCC2    Output          TAC
## 84     LstepCC3    Output          TAC
## 85     LstepCC4    Output          TAC
## 86     Ltarget1    Output          TAC
## 87     Ltarget2    Output          TAC
## 88     Ltarget3    Output          TAC
## 89     Ltarget4    Output          TAC
## 90          MCD    Output          TAC
## 91       MCD4010    Output          TAC
## 92      Rcontrol    Output          TAC
## 93     Rcontrol2    Output          TAC
## 94          SBT1    Output          TAC
## 95          SBT2    Output          TAC
## 96         SPmod    Output          TAC
## 97         SPMSY    Output          TAC
## 98       SPslope    Output          TAC
## 99         SPSRA    Output          TAC
## 100     SPSRA_ML    Output          TAC
## 101          YPR    Output          TAC
## 102       YPR_CC    Output          TAC
## 103       YPR_ML    Output          TAC
## 104        avgMP    Output          TAC
## 105        TCPUE    Output          TAC
## 106          THC    Output          TAC
## 107      FMSYref Reference          TAC
## 108    FMSYref50 Reference          TAC
## 109    FMSYref75 Reference          TAC
## 110        NFref Reference          TAC
```

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"    "ItargetE2"
## [11] "ItargetE3"    "ItargetE4"    "ITe10"        "ITe5"         "LBSPR"
## [16] "LstepCE1"     "LstepCE2"     "LtargetE1"    "LtargetE4"    "matlenlim"
## [21] "matlenlim2"   "minlenLopt1"  "MRnoreal"     "MRreal"       "slotlim"
## [26] "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] "CC2"          "CC3"          "CC4"          "CC5"          "CompSRA"
## [11] "CompSRA4010"  "DAAC"         "DBSRA"        "DBSRA_40"     "DBSRA4010"
## [16] "DCAC"         "DCAC_40"      "DCAC_ML"      "DCAC4010"     "DCACs"
## [21] "DD"           "DD4010"       "DepF"         "DynF"         "Fadapt"
## [26] "Fdem"         "Fdem_CC"      "Fdem_ML"      "Fratio"       "Fratio_CC"
## [31] "Fratio_ML"    "Fratio4010"   "GB_CC"        "GB_slope"     "GB_target"
## [36] "Gcontrol"     "HDAAC"        "ICI"          "ICI2"         "Iratio"
## [41] "Islope1"      "Islope2"      "Islope4"      "IT10"         "IT5"
## [46] "Itarget1"     "Itarget2"     "Itarget3"     "Itarget4"     "ITM"
## [51] "L95target"    "Lratio_BHI"   "Lratio_BHI2"  "LstepCC1"     "LstepCC2"
## [56] "LstepCC3"     "LstepCC4"     "Ltarget1"     "Ltarget2"     "Ltarget3"
## [61] "Ltarget4"     "MCD"          "MCD4010"      "Rcontrol"     "Rcontrol2"
## [66] "SBT1"         "SBT2"         "SPmod"        "SPMSY"        "SPslope"
## [71] "SPSRA"        "SPSRA_ML"     "YPR"          "YPR_CC"       "YPR_ML"
## [76] "avgMP"        "TCPUE"        "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         Recs
## 1    matlenlim     Input           SL
## 2        ITe10     Input          TAE
## 3      AvC_MLL     Mixed      TAC, SL
## 4 Itarget1_MPA     Mixed TAC, Spatial
## 5          AvC    Output          TAC
## 6     Itarget1    Output          TAC
## 7      FMSYref Reference          TAC
## 8        NFref Reference          TAC
```

See Determining Feasible and Available Management Procedures for information on how to identify management procedures that are potentially suitable for your fishery.

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)
```

This may take a minute or two to run. We have now conducted a Management Strategy Evaluation for our fishery described in the Operating Model with 8 Management Procedures. Next we will evaluate whether the model has converged and then look at the MSE results.

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%?

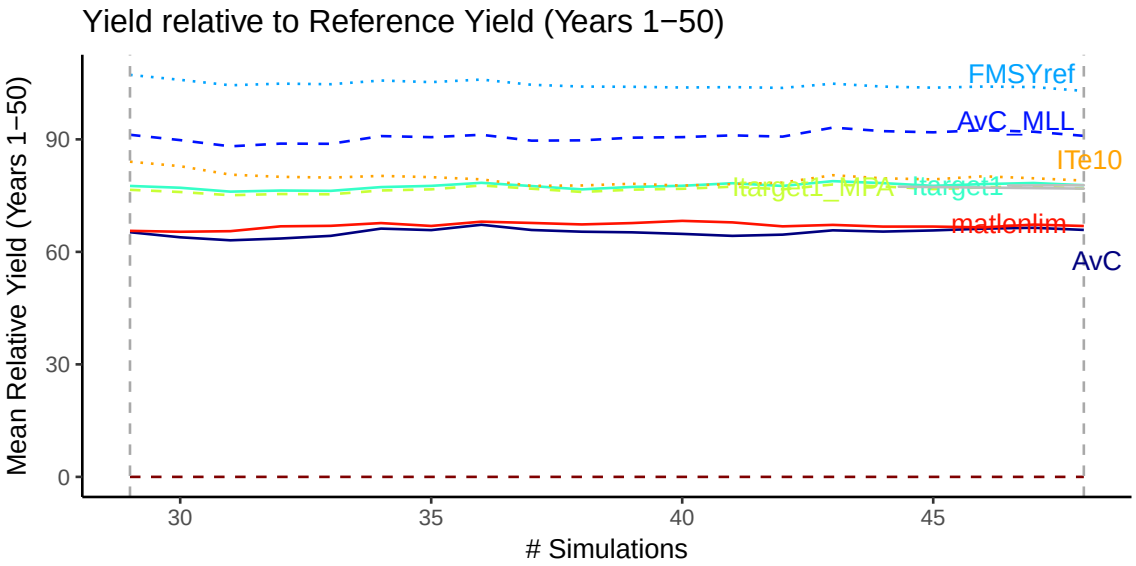
The number of simulations to calculate the convergence statistics, the minimum change threshold, and the performance metrics to use can be specified as arguments to the function. See the help documentation for more details (`?Converge`).

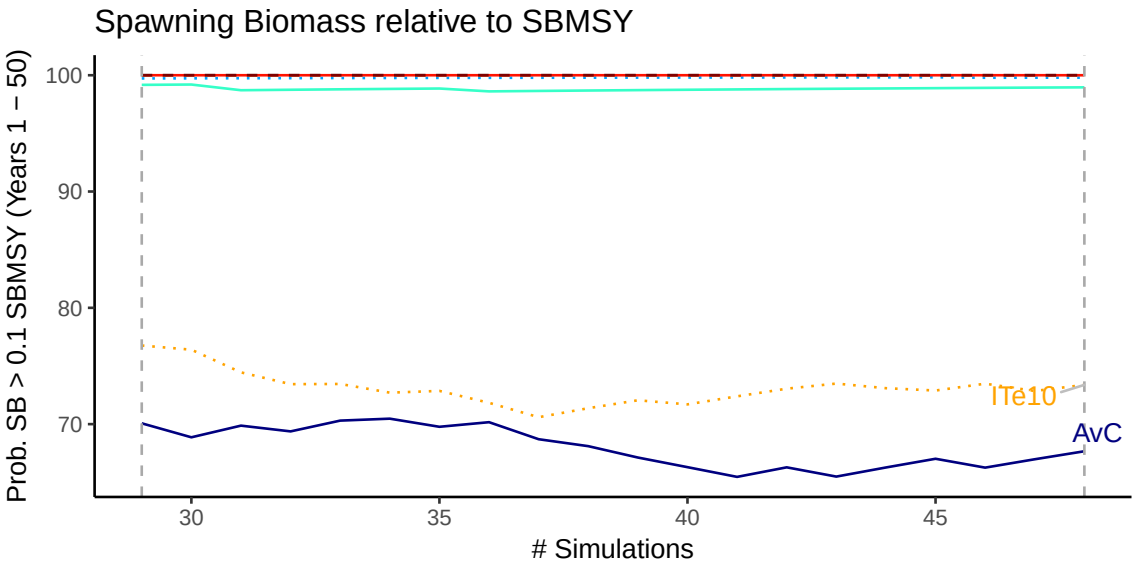
```
Converge(myMSE)
```

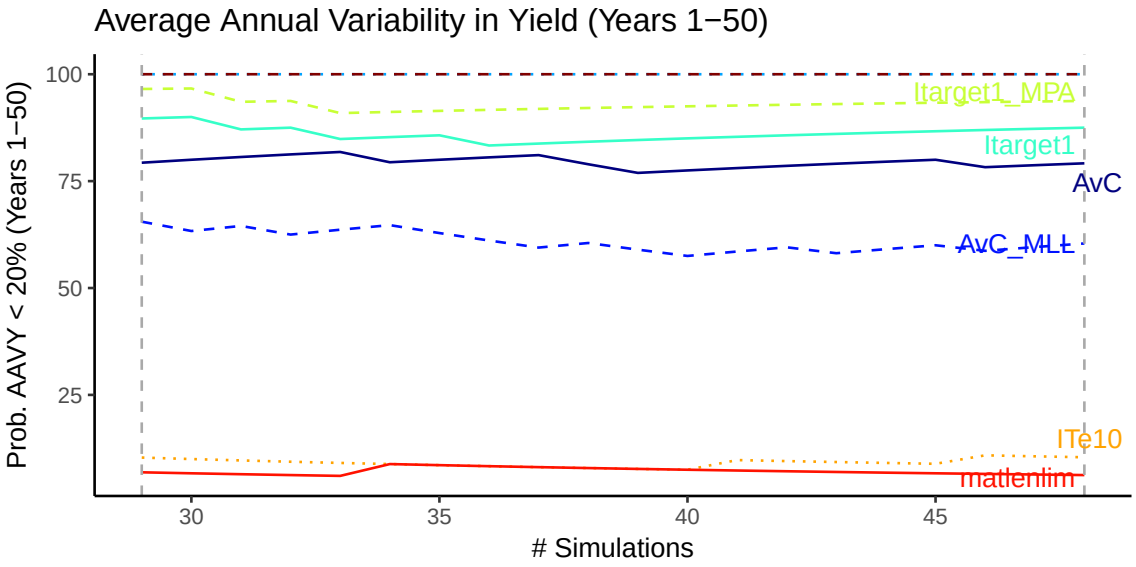
```
## Checking if order of MPs is changing in last 20 iterations

## Checking average difference in PM over last 20 iterations is > 0.5

## Plotting MPs 1 - 8

##
## Yield relative to Reference Yield (Years 1-50)

## Order over last 20 iterations is not consistent for:
##  Itarget1 ITe10

## Mean difference over last 20 iterations is > 0.5 for:
## AvC Itarget1 matlenlim ITe10 AvC_MLL Itarget1_MPA FMSYref

##
## Spawning Biomass relative to SBMSY

## Mean difference over last 20 iterations is > 0.5 for:
## AvC ITe10
```

```
##
## Average Annual Variability in Yield (Years 1-50)
```

```
## Mean difference over last 20 iterations is > 0.5 for:
## AvC Itarget1 matlenlim ITe10 AvC_MLL Itarget1_MPA
```


Yield relative to Reference Yield (Years 1−50)
Mean Relative Yield (Years 1−50)
90
60
30
0
FMSYref
AvC_MLL
ITe10
Itarget1_MPA
Itarget1
matlenlim
AvC
30
35
40
45
Simulations


Spawning Biomass relative to SBMSY
Prob. SB > 0.1 SBMSY (Years 1 − 50)
100
90
80
70
ITe10
AvC
30
35
40
45
Simulations


Average Annual Variability in Yield (Years 1−50)
Prob. AAVY < 20% (Years 1−50)
100
75
50
25
Itarget1_MPA
Itarget1
AvC
AvC_MLL
ITe10
matlenlim
30
35
40
45
Simulations

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)
```

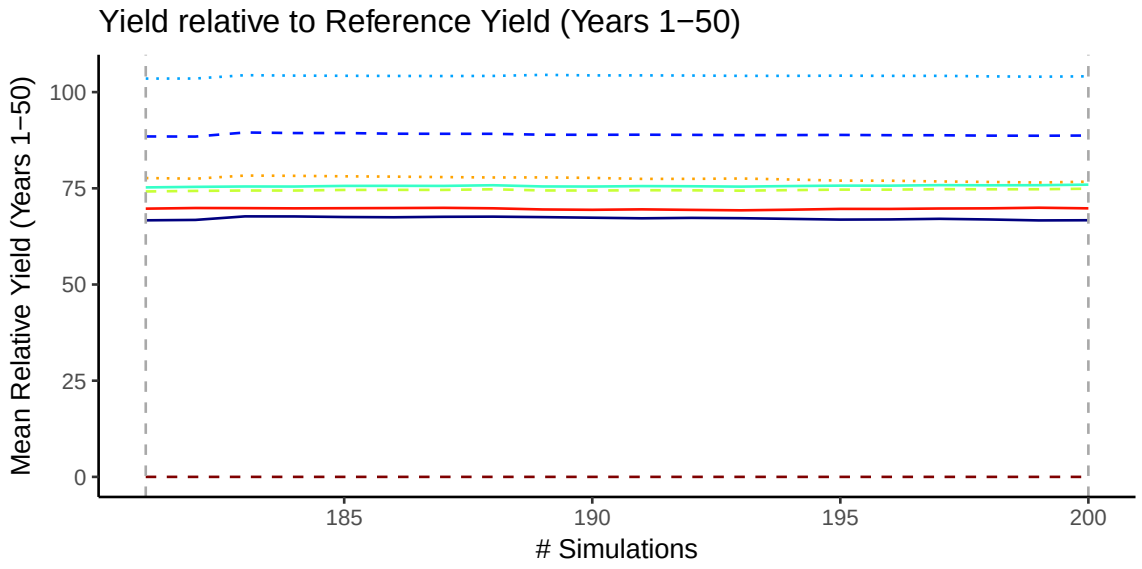
Is 200 simulations enough?

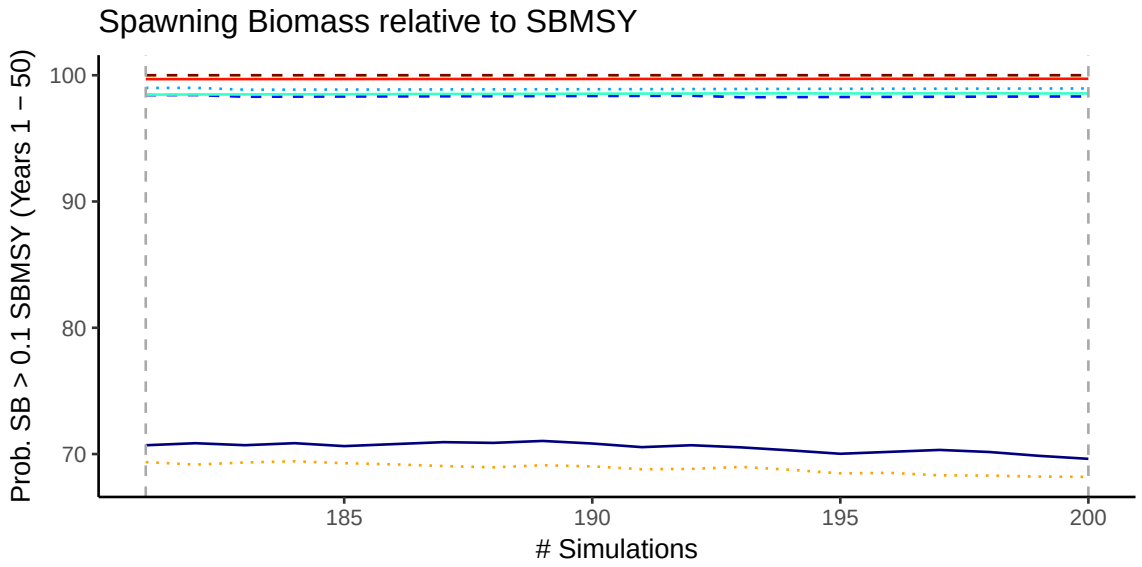
```
Converge(myMSE_200)
```

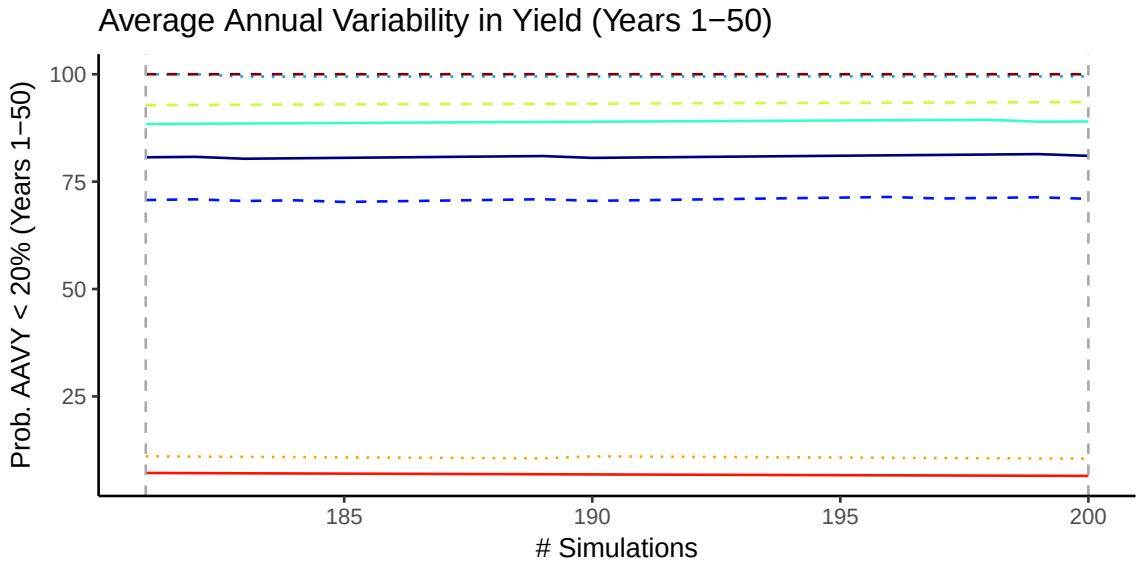
```
## Checking if order of MPs is changing in last 20 iterations
```

```
## Checking average difference in PM over last 20 iterations is > 0.5
```

```
## Plotting MPs 1 - 8
```


Yield relative to Reference Yield (Years 1−50)
Mean Relative Yield (Years 1−50)
100
75
50
25
0
185
190
195
200
Simulations


Spawning Biomass relative to SBMSY
Prob. SB > 0.1 SBMSY (Years 1 − 50)
100
90
80
70
185
190
195
200
Simulations


Average Annual Variability in Yield (Years 1−50)
Prob. AAVY < 20% (Years 1−50)
100
75
50
25
185
190
195
200
Simulations

Chapter 8

Examining the MSE Results

Arguably the most important part of the MSE is interpreting the results and identifying a management procedure (MP) that is most suitable for the fishery.

This involves asking several questions:

1. Which MPs can be excluded from the list of candidates because they perform worst than all other options?
2. Which MPs are most likely to meet our management objectives?
3. How do we identify the MP most suited to our fishery?
4. Which data sources are most critical to the performance of the best performing MP?

8.1 Introducing Performance Metrics

To interpret the MSE results it is important that a clear set of performance metrics have been defined. Fisheries managers often have broadly defined policy goals. These conceptual objectives must be translated to quantitative operational objectives so that the MSE results can be used to evaluate performance against the specified management objectives.

For example, suppose that the fishery managers had stated broad goals to maximize yield from the fishery while minimizing the risk of the stock collapsing to unacceptably low levels. In order to use MSE to determine which MPs are most likely to meet these objectives it is neccessary to be more specific:

- What are *unacceptable low stock levels*? Some fraction of unfished biomass? The lowest observed historical biomass?
- What is an *acceptable level of risk*? What chance are we willing to tolerate that the stock will fall below that limit?
- How much yield are we willing to give up in order to increase the probability of the stock staying above unacceptably low limit?

It is important to recognize that performance metrics can vary considerably between different fisheries and management structures, but are a crucial component of the MSE and must be carefully defined before the analysis is carried out. The Performance Metrics chapter discusses this topic in more detail.

The DLMtool includes a number of commonly used performance metrics and a series of functions to summarize MP performance. The MSE results can be examined either graphically in a plot or summarized in a table. Advanced users can also develop their own plotting and summary functions (see the Custom Performance Metrics chapter for more details).

Here we briefly demonstrate some of the plotting and summary functions in DLMtool. The Examining the MSE object chapter and other chapters in that section describe the process of evaluating MSE results in more detail.

8.2 Summary Table

The `summary` function can be used to generate a table of MP performance with respect to a set of performance metrics:

```
summary(myMSE_200)
```

```
## Calculating Performance Metrics

##                                       Performance.Metrics
## 1                 Probability of not overfishing (F<FMSY)
## 2                      Spawning Biomass relative to SBMSY
## 3        Average Annual Variability in Yield (Years 1-50)
## 4 Average Yield relative to Reference Yield (Years 41-50)
##
## 1              Prob. F < FMSY (Years 1 - 50)
## 2        Prob. SB > 0.5 SBMSY (Years 1 - 50)
## 3              Prob. AAVY < 20% (Years 1-50)
## 4 Prob. Yield > 0.5 Ref. Yield (Years 41-50)
##
##
## Probability:
##             MP PNOF  P50  AAVY  LTY
## 1          AvC 0.53 0.60 0.810 0.43
## 2     Itarget1 0.81 0.89 0.890 0.75
## 3    matlenlim 0.91 0.96 0.065 0.62
## 4        ITe10 0.26 0.42 0.100 0.31
## 5      AvC_MLL 0.80 0.86 0.710 0.76
## 6 Itarget1_MPA 0.80 0.90 0.940 0.75
## 7      FMSYref 0.12 0.71 1.000 0.91
## 8        NFref 1.00 0.99 1.000 0.00
```

By default the `summary` function includes four performance metrics, and displays the probability that:

1. fishing mortality (F) is below F_{MSY}, i.e *Not Overfishing* (`PNOF`)
2. spawning biomass (SB) is above half of biomass at maximum sustainable yield (SB_{MSY}) (`P50`)
3. average interannual variability in yield is less than 20% (`AAVY`)
4. long-term yield (last 10 years of projection period) is above half of the maximum yield obtainable at a constant fishing rate (`LTY`)

In this example we can see that probability of $\text{SB} > 0.5\text{SB}_{\text{MSY}}$ for AvC is 0.6.

The performance metrics have been defined in such a way that a higher number is always better (e.g, probability of *Not Overfishing* rather than *Overfishing* where a lower probability would be more desirable).

Help documentation for the peformance metrics can be found in the usual way, for example:

```
?PNOF
```

The performance metrics in the `summary` function are completely customizable. See the Performance Metrics and Custom Performance Metrics chapters for more details.

8.3 Plotting MSE Results

DLMtool includes several functions for plotting the MSE results. 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_Dplot COSEWIC_Hplot COSEWIC_Pplot Cplot
## DFO_plot DFO_plot2 DFO_proj IOTC_plot Kplot
## NOAA_plot NOAA_plot2 Pplot Pplot2 PWhisker
## Tplot Tplot_old Tplot2 Tplot2_old Tplot3
## TradePlot TradePlot_old VOI VOI2 VOIplot
## VOIplot2 wormplot
```

Here we demonstrate a few of the plotting functions for the MSE results.

8.3.1 Trade-Off Plots

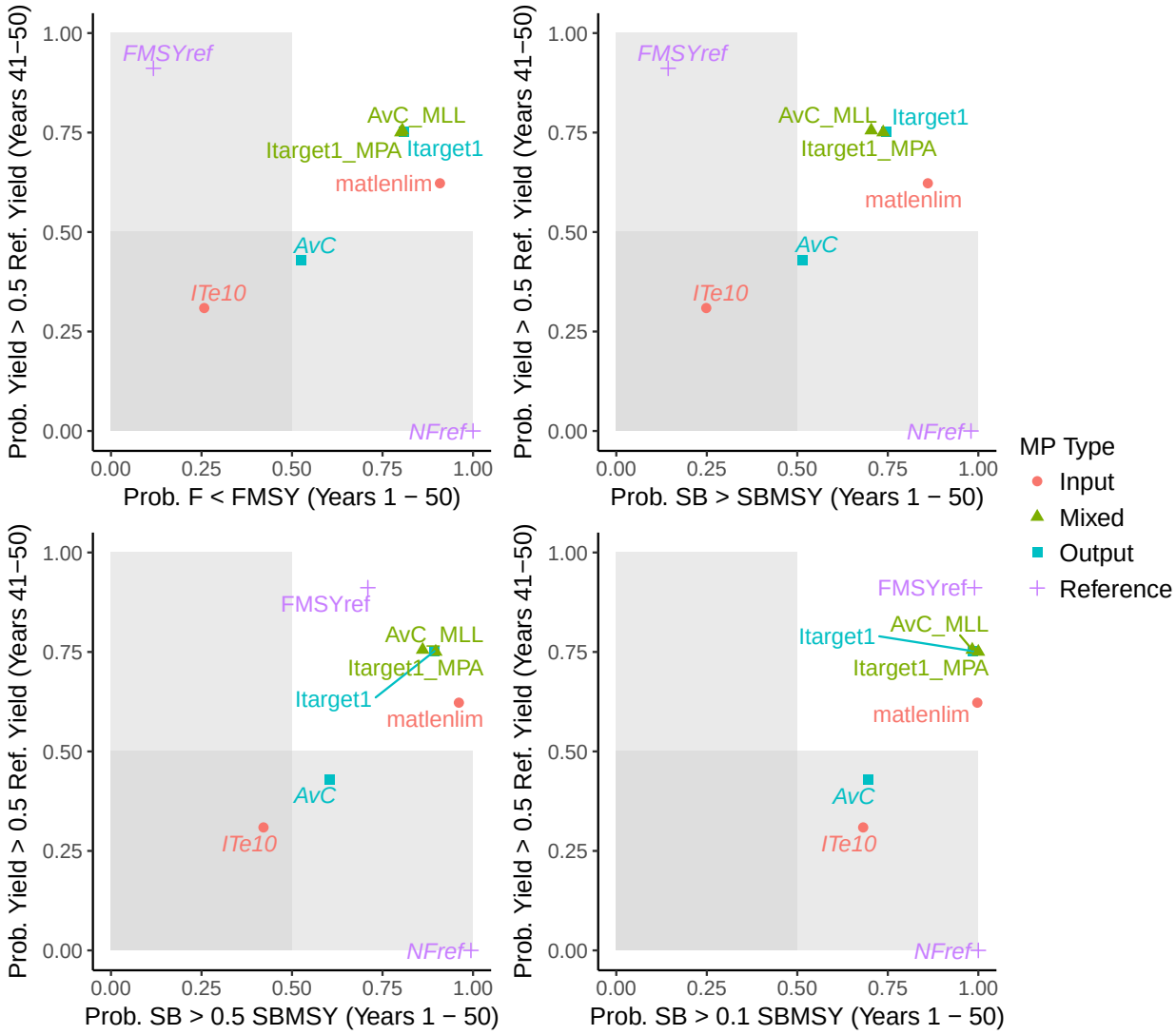
The `Tplot` function creates four plots that show the trade-off between the probability that the long-term expected yield is greater than half of the highest obtainable yield at a fixed F (reference yield) against the probability of:

1. Not overfishing in all projection years ($F/F_{\text{MSY}} < 1$)
2. Spawning biomass (SB) above SB_{MSY} in all projection years ($\text{SB} > \text{SB}_{\text{MSY}}$)
3. Spawning biomass above $0.5\text{SB}_{\text{MSY}}$ ($\text{SB} > 0.5\text{SB}_{\text{MSY}}$)
4. Spawning biomass above $0.1\text{SB}_{\text{MSY}}$ ($\text{SB} > 0.1\text{SB}_{\text{MSY}}$)

The `Tplot` function includes minimum acceptable risk thresholds indicated by the horizontal and vertical gray shading. These thresholds can be adjusted be the `Lims` argument to the `Tplot` function. See `?Tplot` for more information on adjusting the risk thresholds.

MPs that fail to meet one or both of the risk thresholds for each axis are shown in *italics* text. The `Tplot` function returns a data frame showing the performance of each MP with respect to the 5 performance metrics, and whether the MP is *Satisificed*, i.e., if it meets the minimum performance criteria for **all** performance metrics.

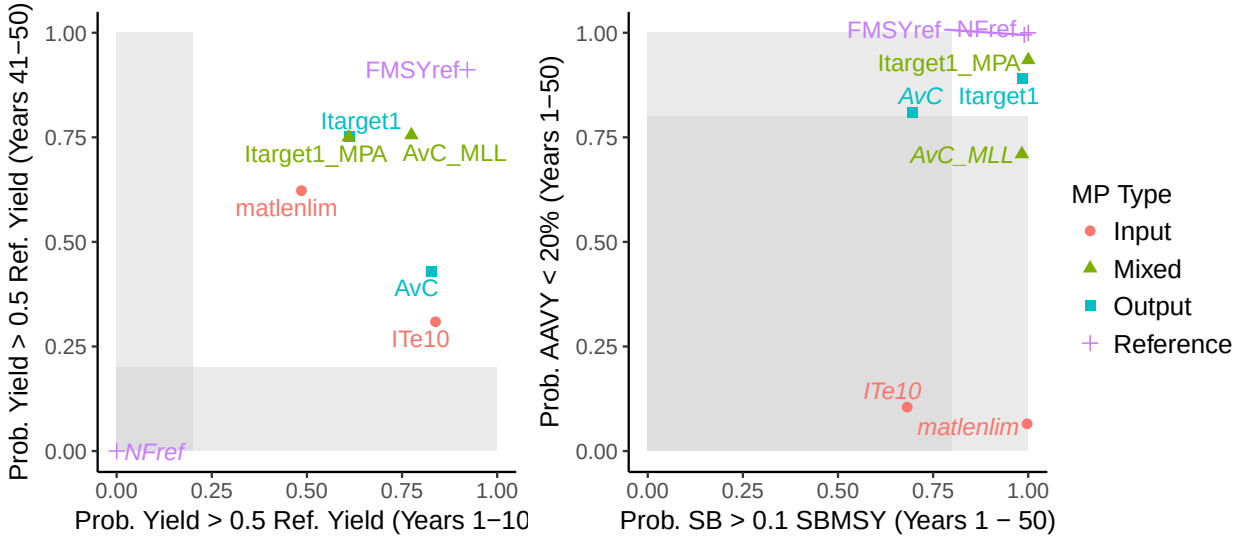
```
Tplot(myMSE_200)
```




```
##             MP PNOF  LTY P100  P50  P10 Satisificed
## 1          AvC 0.53 0.43 0.51 0.60 0.70       FALSE
## 2     Itarget1 0.81 0.75 0.75 0.89 0.99        TRUE
## 3    matlenlim 0.91 0.62 0.86 0.96 1.00        TRUE
## 4        ITe10 0.26 0.31 0.25 0.42 0.68       FALSE
## 5      AvC_MLL 0.80 0.76 0.70 0.86 0.98        TRUE
## 6 Itarget1_MPA 0.80 0.75 0.74 0.90 1.00        TRUE
## 7      FMSYref 0.12 0.91 0.14 0.71 0.99       FALSE
## 8        NFref 1.00 0.00 0.98 0.99 1.00       FALSE
```

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)
```



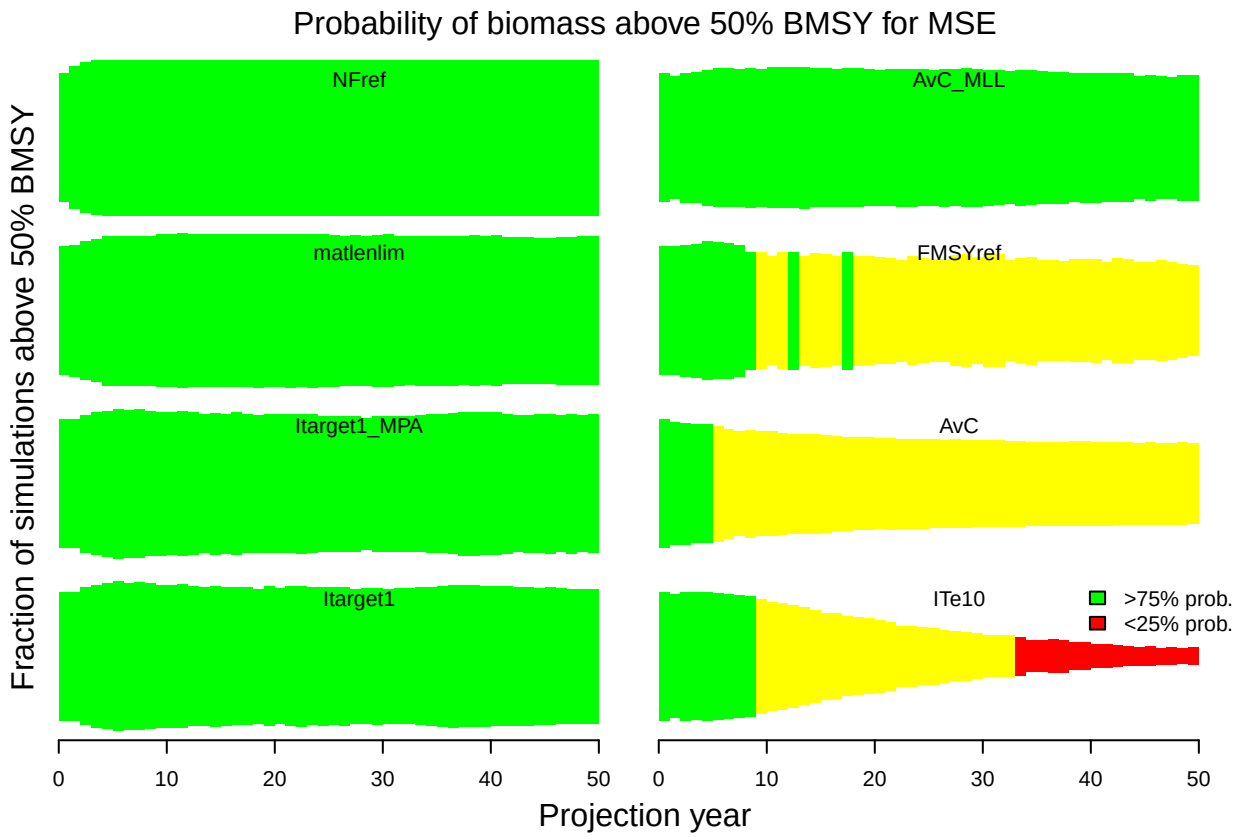

```
##             MP  STY  LTY  P10  AAVY Satisificed
## 1          AvC 0.83 0.43 0.70 0.810       FALSE
## 2     Itarget1 0.61 0.75 0.99 0.890        TRUE
## 3    matlenlim 0.49 0.62 1.00 0.065       FALSE
## 4        ITe10 0.84 0.31 0.68 0.100       FALSE
## 5      AvC_MLL 0.77 0.76 0.98 0.710       FALSE
## 6 Itarget1_MPA 0.61 0.75 1.00 0.940        TRUE
## 7      FMSYref 0.92 0.91 0.99 1.000        TRUE
## 8        NFref 0.00 0.00 1.00 1.000       FALSE
```

The `Tplot`, `Tplot2` and `Tplot3` functions are part of a family of plotting functions that are fully customizable, and designed to work with all Performance Metrics objects. See `?Tplot` and the Performance Metrics chapter for more information.

8.3.2 Wormplot

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

```
wormplot(myMSE_200)
```



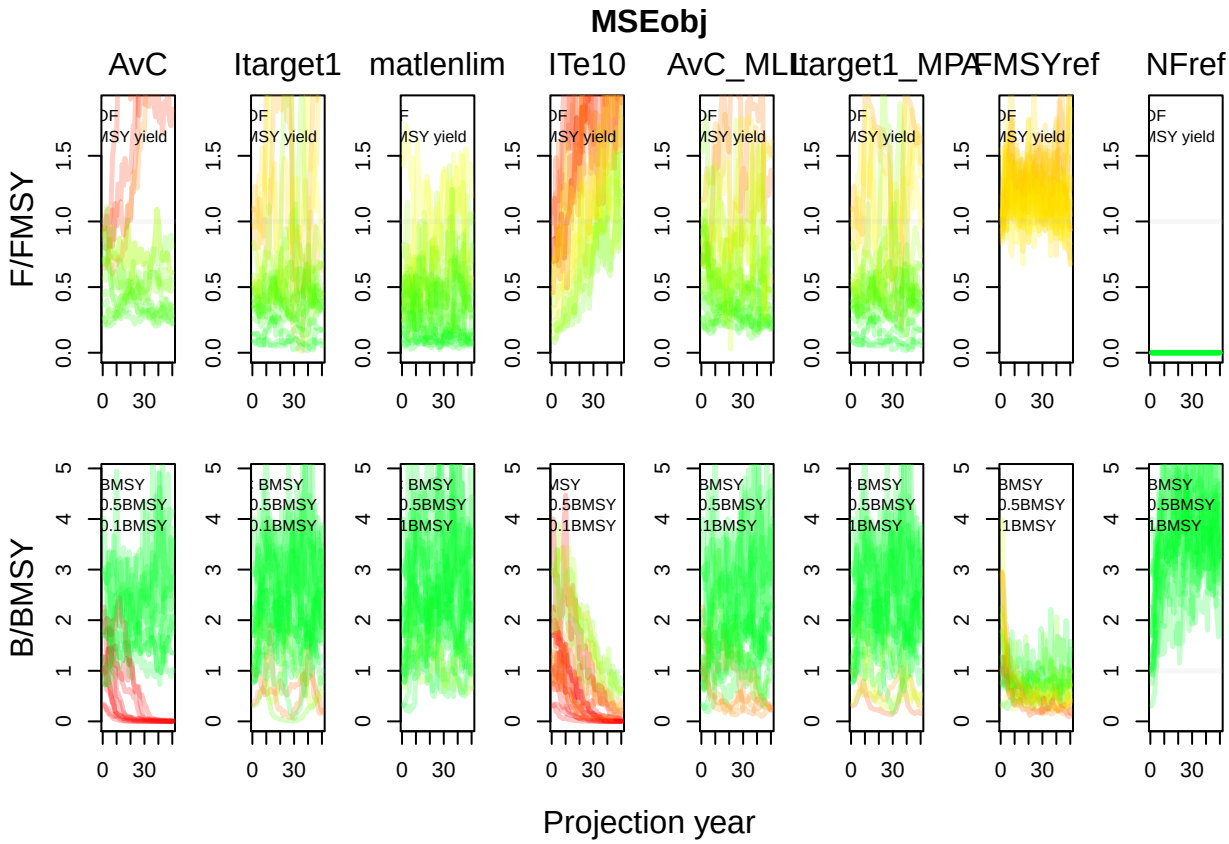
The arguments to the `wormplot` function allow you to choose the reference level for the biomass relative to B_{MSY}, as well as the upper and lower bounds of the colored bands.

8.3.3 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:

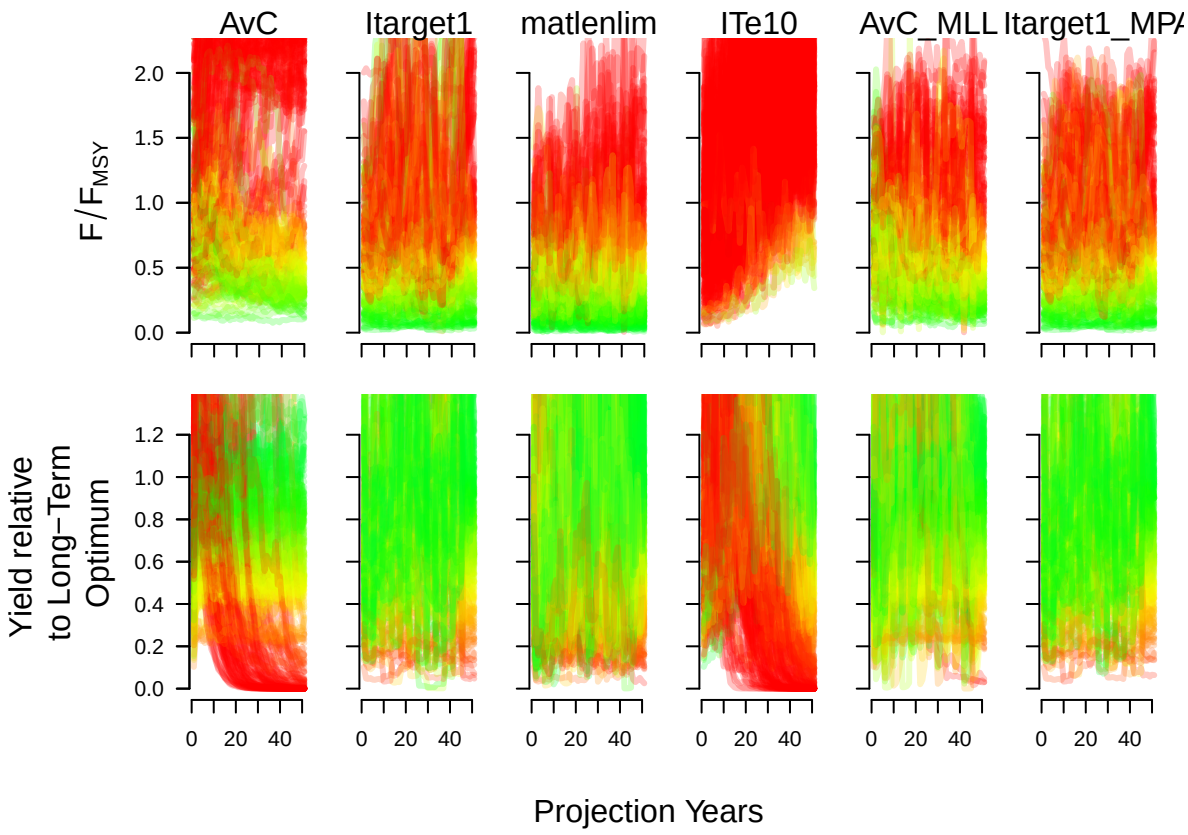
```
Pplot(myMSE_200)
```



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"))
```

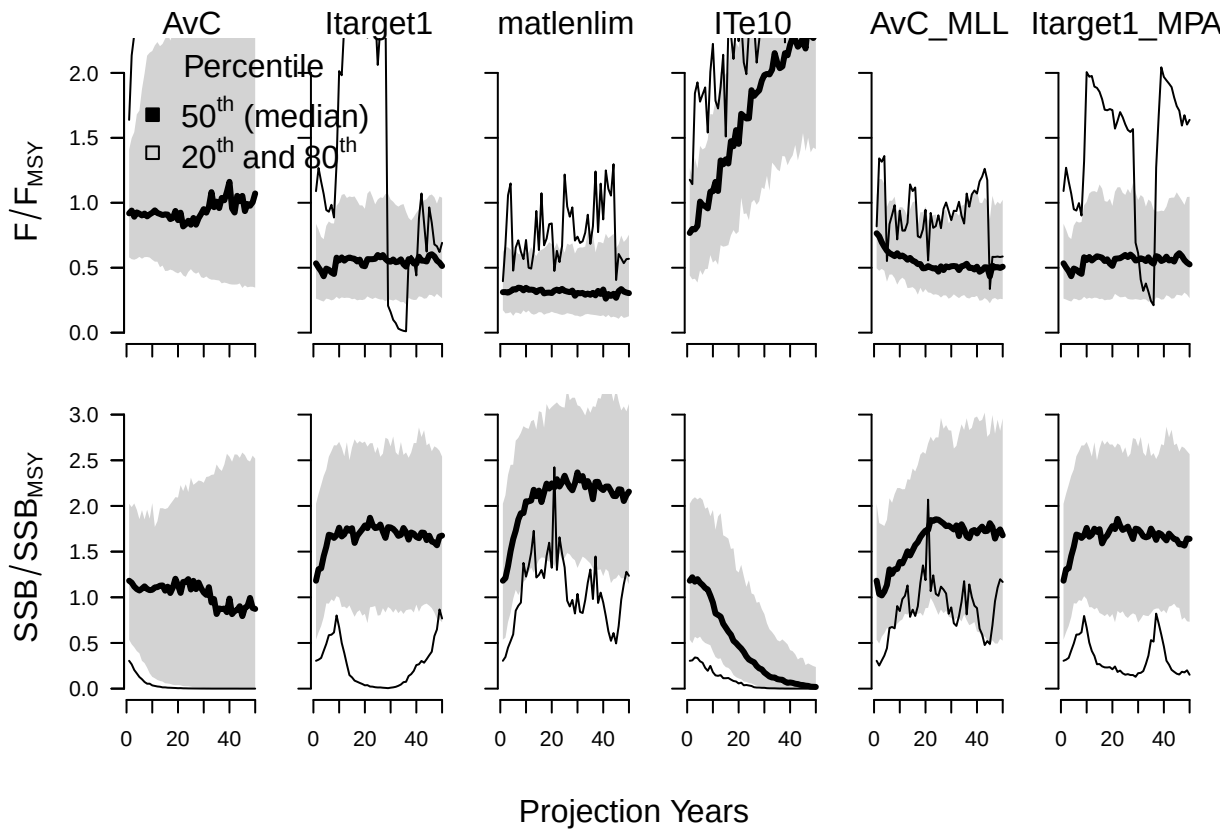
```
## MSE object has more than 6 MPs. Plotting the first 6
```



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
```

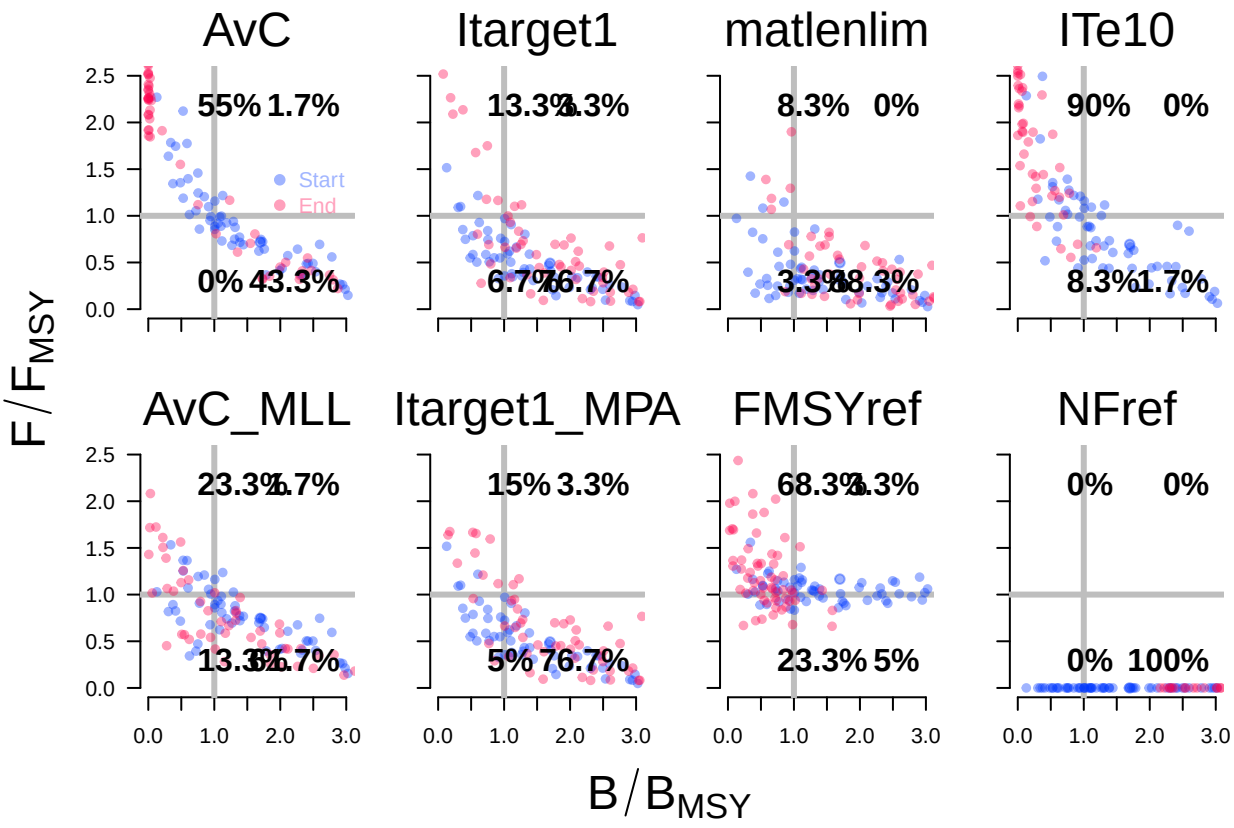


Details on additional controls for the `Pplot` and `Pplot2` functions can be found in the help documentation associated with this function.

8.3.4 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:

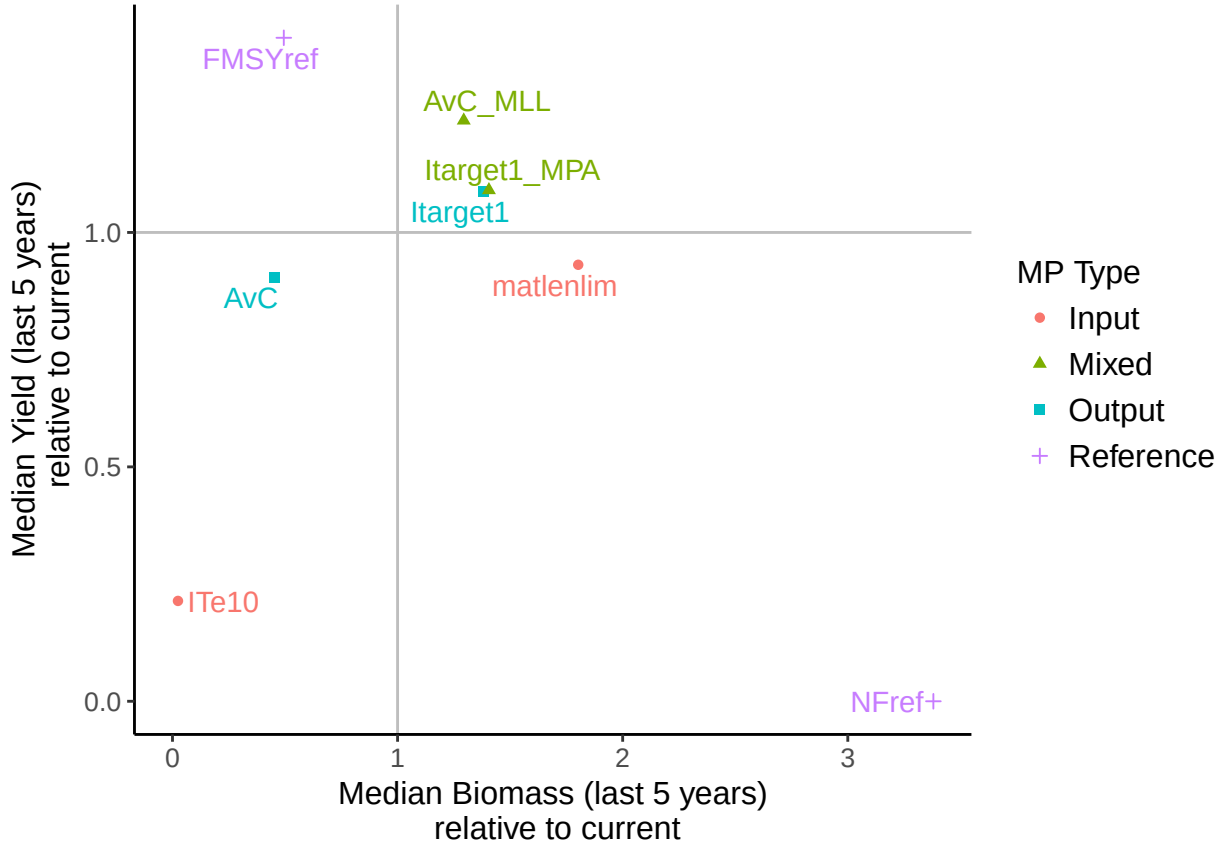
```
Kplot(myMSE_200)
```



8.3.5 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)
```


FMSYref
AvC_MLL
Itarget1_MPA
Itarget1
matlenlim
AvC
ITe10
NFref+
1.0
0.5
0.0
0
1
2
3
Median Yield (last 5 years) relative to current
Median Biomass (last 5 years) relative to current
MP Type
Input
Mixed
Output
Reference

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 10 CPUs.
```

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

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

By default the `setup` function initializes half of the available 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 10 processors
```

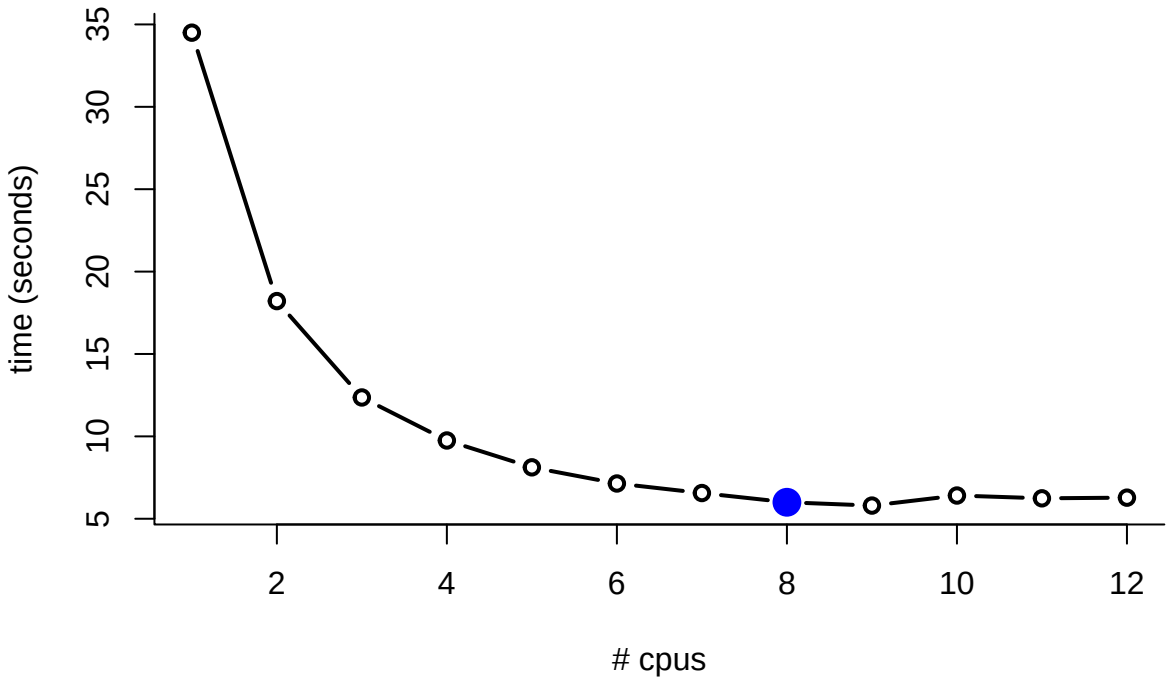
```
## MSE completed
```

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, run time decreased from 1 to 10 seconds when using parallel processing on 10 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()
```




```
##    ncpu  time
## 1     1 34.50
## 2     2 18.21
## 3     3 12.36
## 4     4  9.75
## 5     5  8.12
## 6     6  7.14
## 7     7  6.56
## 8     8  6.00
## 9     9  5.80
```

```
## 10   10  6.41
## 11   11  6.24
## 12   12  6.28
```

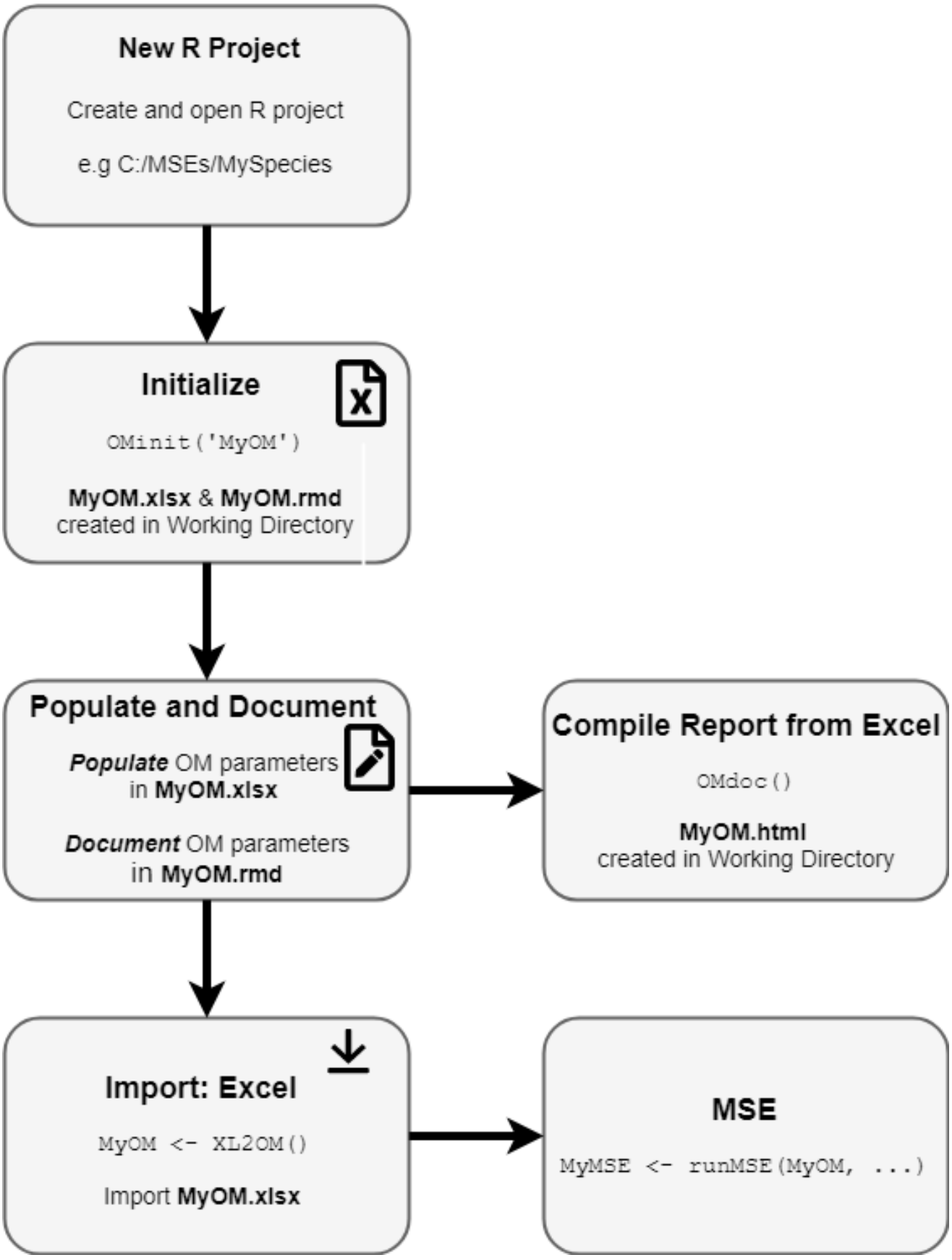
Creating an Operating Model

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

Generating Correlated Life-History Parameters

By default DLMtool independently samples the life-history parameters from uniform distributions. The `LH2OM` function can be used to force correlation between the life-history parameters, or predict values for missing life-history parameters.

Thorson et al. (2017) developed a hierarchical model, based on records available in FishBase, to predict life-history parameters for all 33,000+ fish species listed within FishBase. The `LH2OM` function uses the Thorson et al. (2017) model to generate correlated samples of the life-history parameters based on the relevant taxonomic information (Class, Family, Genus, and Species) for the species being modelled in the MSE.

The approach uses taxonomic information and any available information on the life-history parameters of the species in the OM to generate predictions of the missing parameters. For example, if no knowledge exists on the four life-history parameters (*Linf*, *L50*, *M*, and *K*), the model uses the taxonomic information (first Genus and Species, and if records don't exist for this species in FishBase, then up a taxonomic level to Family, and so on) to generate predicted values for all four parameters. If information is available for some parameters, e.g., reasonable bounds on *Linf* and *M* for the species, the hierarchical model is used to generate predictions of the corresponding *L50* and *K* values. That is, the observed ratios of $L50/Linf$ and M/K are maintained. In this way the simulated life-history strategies are biologically realistic, and are appropriate for the species being modelled in the MSE.

11.1 Predicting all life-history parameters

The `LH2OM` function can be used to predict the four correlated life-history parameters (*Linf*, *L50*, *M*, and *K*) using only the available taxonomic information (only for fish species).

For example, here we create an empty `OM` object, populate the `Species` slot and use the `LH2OM` function to predict the life-history parameters using hierarchical model (Thorson et al. 2017):

```
OM <- new("OM")
```

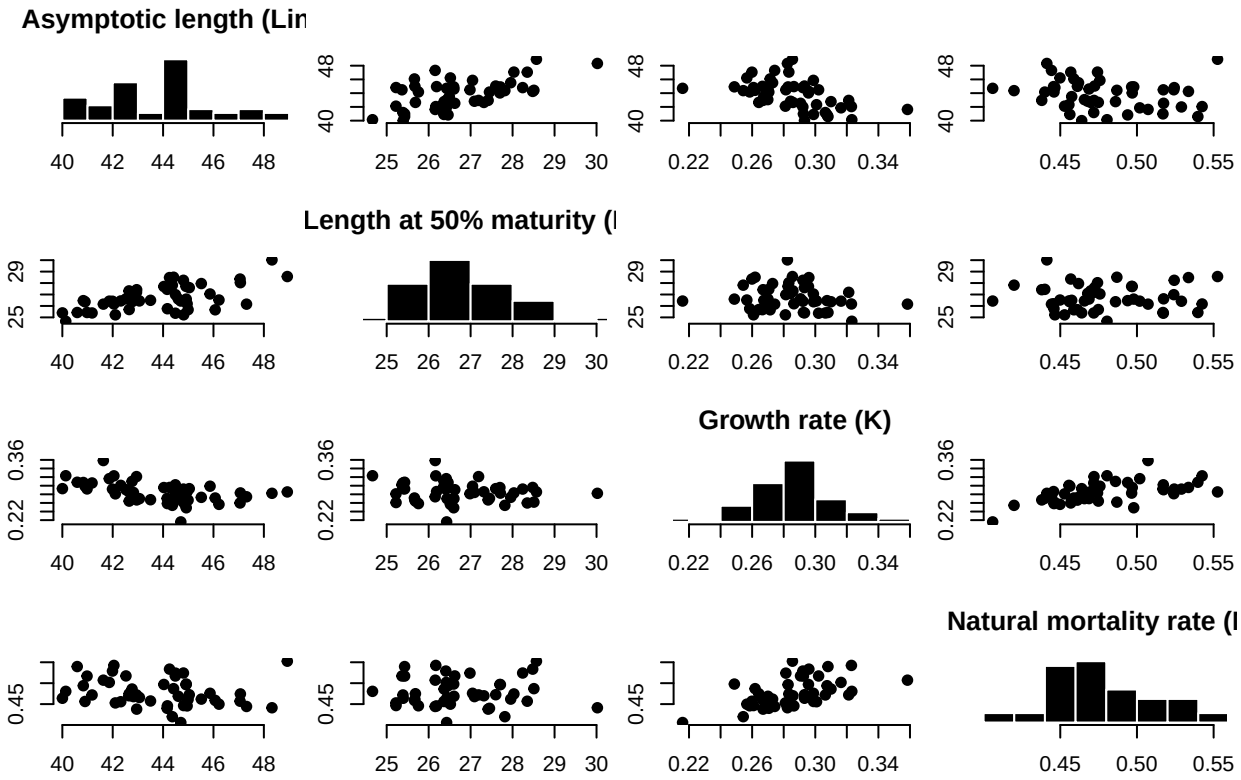
```
## No Stock object found. Returning a blank OM object
```

```
OM@Species <- "Scomber japonicus"
OM <- LH2OM(OM)
```

```
## Predicting Linf
```

```
## Predicting L50
```

```
## Predicting K

## Predicting M

## Species match: Actinopterygii Perciformes Scombridae Scomber japonicus
```



11.2 Predicting some life-history parameters

In some cases local estimates of life-history parameters may be available which are more reliable and less variable than those predicted from the FishBase database. For example, suppose that we had estimates of the natural mortality rate (M) for our stock that ranged between 0.3 and 0.4. We populate the `OM@M` slot with these values and use `LH2OM` to predict the correlated K parameter values:

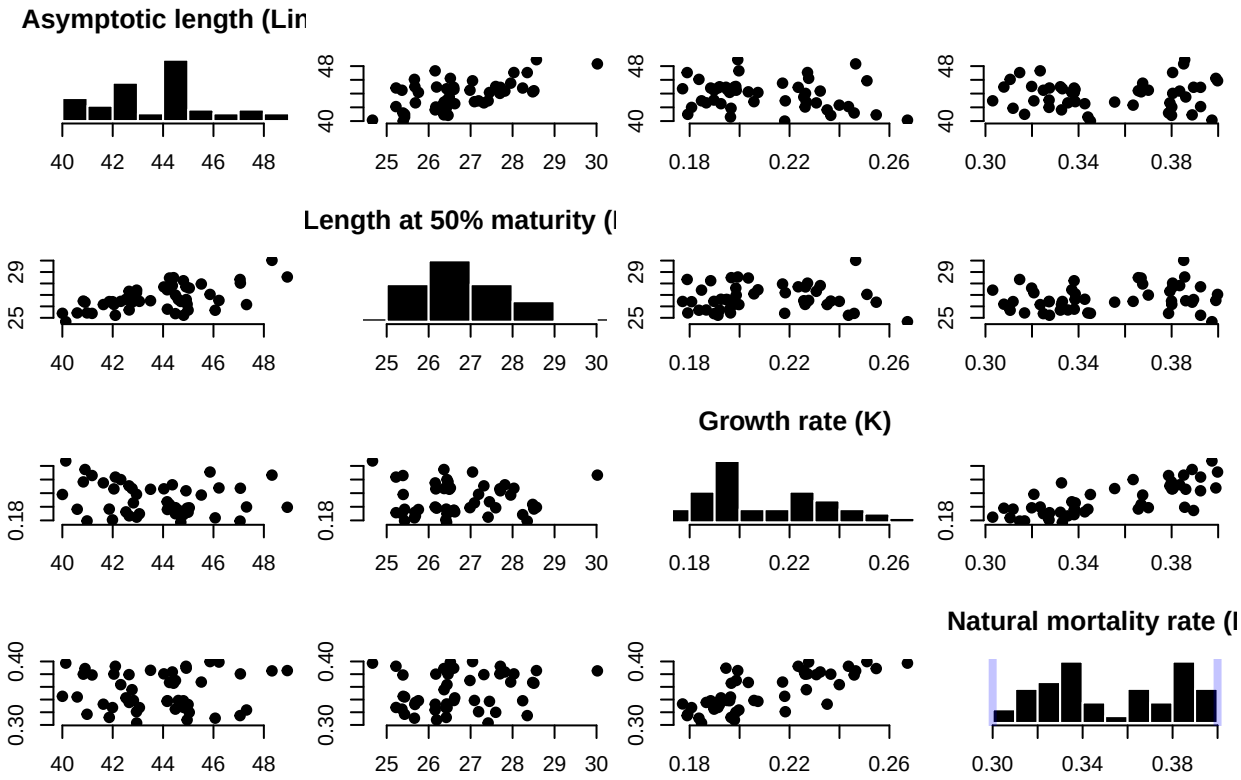
```
OM <- new("OM")
```

```
## No Stock object found. Returning a blank OM object
```

```
OM@Species <- "Scomber japonicus"
OM@M <- c(0.3, 0.4)
OM <- LH2OM(OM)
```

```
## Predicting Linf

## Predicting L50

## Predicting K

## Predicting K from M
```

```
## Species match: Actinopterygii Perciformes Scombridae Scomber japonicus
```



Notice that the sampled M values are within the bounds specified in `OM@M` (vertical lines).

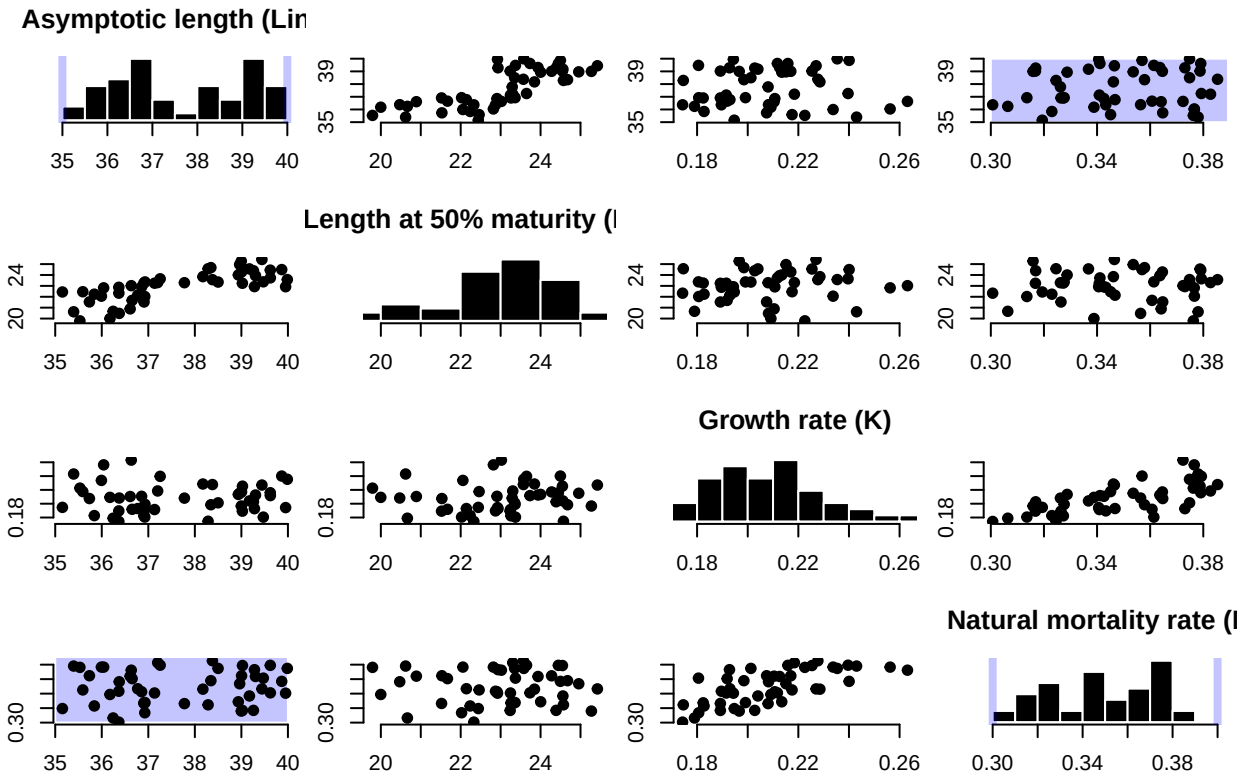
Similiarly, if information is also available for asymptotic length *Linf*:

```
OM <- new("OM")
```

```
## No Stock object found. Returning a blank OM object
```

```
OM@Species <- "Scomber japonicus"
OM@M <- c(0.3, 0.4)
OM@Linf <- c(35, 40)
OM <- LH2OM(OM)
```

```
## Predicting L50

## Predicting K

## Predicting L50 from Linf

## Predicting K from M

## Species match: Actinopterygii Perciformes Scombridae Scomber japonicus
```



11.3 Predicting correlated parameters

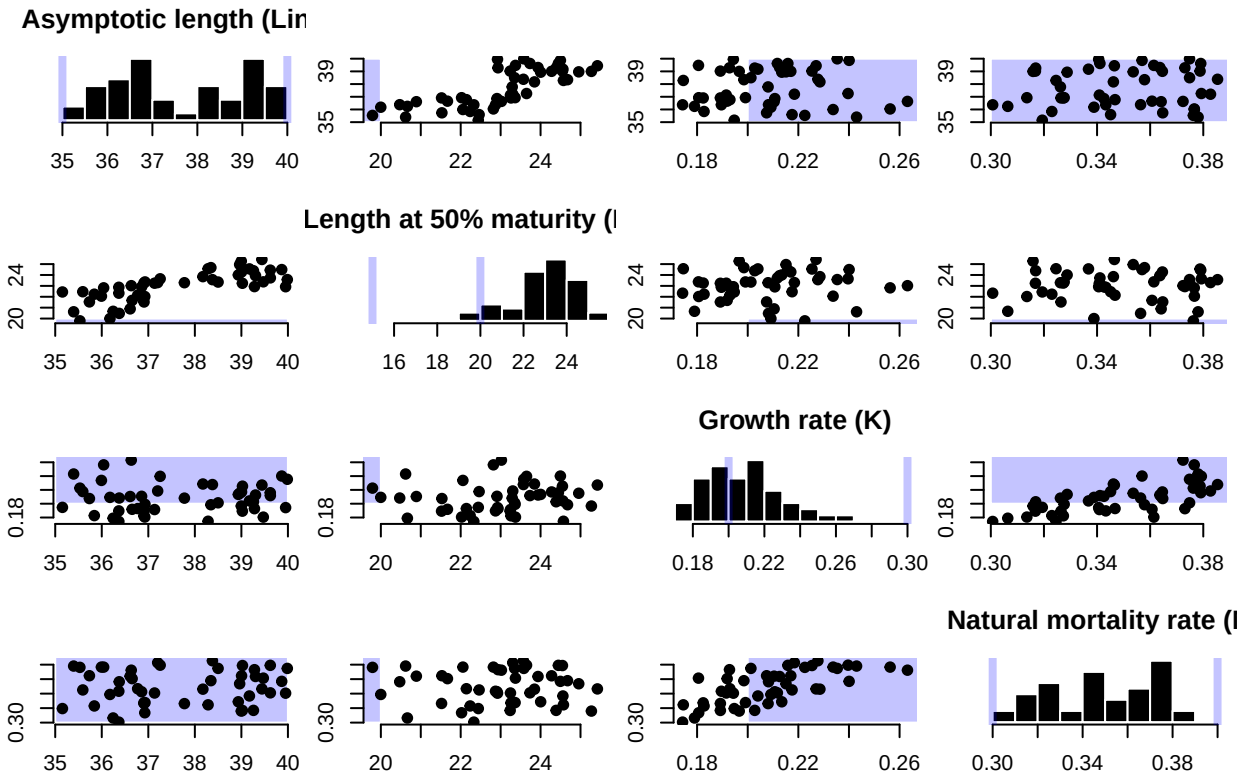
If bounds for all life-history parameters are specified in the OM, the `LH2OM` function will predict values of $L50$ and K which may fall outside of the bounds specified in the OM. For example, here we specify bounds for all life-history parameters and see that the predicted values for $L50$ and K are mostly above and below the bounds we specied in the OM slots (vertical lines and shading). This is because the predictions of the $L50/Linf$ and M/K ratios from the FishBase database were outside the ranges specied in the OM; in other words, the ranges specified in the OM have rarely been observed in nature.

```
OM <- new("OM")
```

```
## No Stock object found. Returning a blank OM object
```

```
OM@Species <- "Scomber japonicus"
OM@M <- c(0.3, 0.4)
OM@K <- c(0.2, 0.3)
OM@Linf <- c(35, 40)
OM@L50 <- c(15, 20)
OM <- LH2OM(OM)
```

```
## Predicting L50 from Linf

## Predicting K from M

## Species match: Actinopterygii Perciformes Scombridae Scomber japonicus
```



We can force the `LH2OM` function to only return values within the M and K bounds by using the `filterMK` argument:

```
OM <- new("OM")
```

```
## No Stock object found. Returning a blank OM object
```

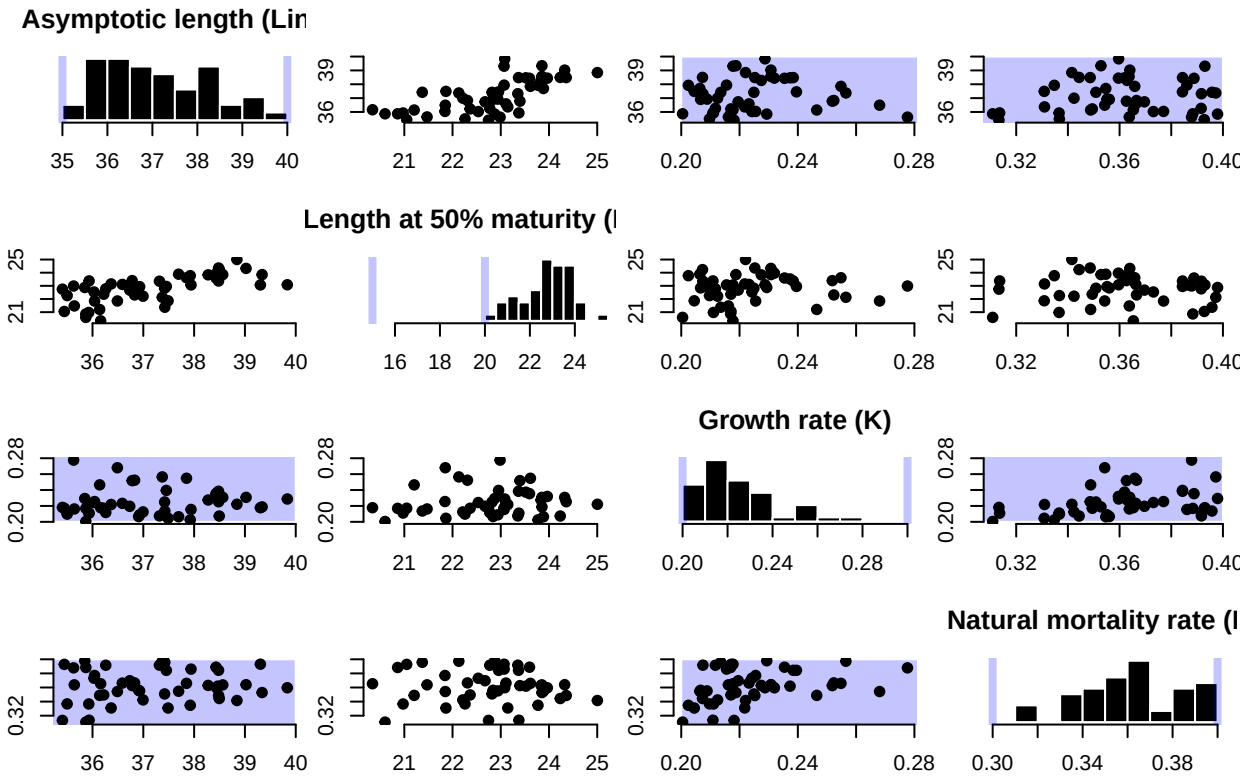
```
OM@Species <- "Scomber japonicus"
OM@M <- c(0.3, 0.4)
OM@K <- c(0.2, 0.3)
OM@Linf <- c(35, 40)
OM@L50 <- c(15, 20)
OM <- LH2OM(OM, filterMK=TRUE)
```

```
## Predicting L50 from Linf
```

```
## Predicting K from M
```

```
## Filtering predicted K within bounds: 0.2 0.3
```

```
## Species match: Actinopterygii Perciformes Scombridae Scomber japonicus
```



11.4 Introducing Custom Parameters

The `LH2OM` function uses a feature of DLMtool called *Custom Parameters*, which are stored in the `OM@cpars` slot.

By default the `OM@cpars` is an empty list:

```
OM <- new("OM")
```

```
## No Stock object found. Returning a blank OM object
```

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

```
##  list()
```

After using the `LH2OM` function, the `cpars` slot is populated with `OM@nsim` correlated samples of the life-history parameters:

```
OM <- new("OM")
```

```
## No Stock object found. Returning a blank OM object
```

```
OM@Species <- "Scomber japonicus"
OM@M <- c(0.3, 0.4)
OM <- LH2OM(OM, plot=FALSE, msg=FALSE)
str(OM@cpars)
```

```
## List of 4
```

```
##  $ Linf: num [1:48] 42 44.4 42.8 48.3 42.7 ...
##  $ M   : num [1:48] 0.38 0.383 0.339 0.385 0.335 ...
##  $ K   : num [1:48] 0.226 0.232 0.206 0.246 0.187 ...
##  $ L50 : num [1:48] 26.2 27.8 27.1 30 25.7 ...
```

Notice also that the `OM@M` slot is no longer used *after* the `LH2OM` function has been run on the OM object:

```
OM@M
```

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

Custom Parameters are a very powerful way to customize the DLMtool, and allow users full control over all sampled and most internal parameters in the DLMtool Operating Model. See the Custom Parameters chapter for more information on this feature.

Chapter 12

Modifying the OM

It is often desirable to created modified versions of an OM for testing purposes such as series of robustness tests or to evaluate whether an MP is performing as expected under ideal conditions.

DLMtool includes several functions for this purpose.

12.1 The `tinyErr` function

The `tinyErr` function can be used to remove observation, implementation, and process error, as well as any gradients in life-history parameters. For example, we first create an Operating Model using built-in OM Components:

```
OM <- new("OM", Albacore, Generic_IncE, Imprecise_Biased, Overages)
```

Notice that our OM includes process error and gradients in life-history parameters, as well as observation and implementation error:

```
OM@Perr # recruitment process error
```

```
## [1] 0.15 0.30
```

```
OM@Linfgrad # gradient in Linf
```

```
## [1] -0.25  0.25
```

```
OM@Cobs # error in observations
```

```
## [1] 0.2 0.6
```

```
OM@TACFrac # implementation error in TAC
```

```
## [1] 1.1 1.2
```

By default the `tinyErr` function will remove all sources of uncertainty and variability:

```
OM2 <- tinyErr(OM)
```

```
## Removing all Observation Error
## Removing all Implementation Error
## Removing all Process Error
## Removing all Gradients
```

```
OM2@Perr # no recruitment process error
```

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

```
OM2@Linfgrad # no gradient in Linf
```

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

```
OM2@Cobs # very low observation error
```

```
## [1] 0.00 0.05
```

```
OM2@TACFrac # no implementation error
```

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

The `obs`, `imp`, `proc`, and `grad` arguments to the `tinyErr` function can be used to control which sources of error and variability to remove from the OM. See `?tinyErr` for more details.

12.2 The `Replace` function

The can be used to replace individual `Stock`, `Fleet`, `Obs`, or `Imp` components in an Operating Model.

For example, to replace the `Stock` object in an OM we provide `Replace` with a new Stock object:

```
OM1 <- new("OM", Albacore, Generic_DecE, Generic_Obs, Overages)
OM2 <- Replace(OM1, Blue_shark, Name="Blue_shark OM based on OM1")
```

```
## Replacing sub-model: Stock
```

Likewise, to replace any of the other OM components:

```
OM1 <- new("OM", Albacore, Generic_DecE, Generic_Obs, Overages)
OM2 <- Replace(OM1, Generic_IncE, Name="OM1 with new Fleet")
```

```
## Replacing sub-model: Fleet
```

```
OM3 <- Replace(OM2, Perfect_Info, Name="OM2 with new Obs")
```

```
## Replacing sub-model: Obs
```

```
OM4 <- Replace(OM3, Perfect_Imp, Name="OM2 with new Imp")
```

```
## Replacing sub-model: Imp
```

Chapter 13

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 14

Examining the MSE object

The MSE object contains all of the output from the MSE. 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. We will then briefly examine some of the contents of the MSE object. The chapters Performance Metrics and Custom Performance Metrics contain more information on the MSE object.

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 10 processors
```

```
## MSE completed
```

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.

14.1 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`).

14.2 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] "A"            "AC"           "ageM"         "B0"
##  [5] "Blow"         "BMSY"         "BMSY_B0"      "Depletion"
##  [9] "dFfinal"      "DR"           "Esd"          "Fdisc"
## [13] "FMSY"         "FMSY_M"       "Frac_area_1"  "hs"
## [17] "K"            "Kgrad"        "Ksd"          "L5"
## [21] "L50"          "L95"          "LFC"          "LFR"
## [25] "LFS"          "Linf"         "Linfgrad"     "Linfsd"
## [29] "LR5"          "M"            "Mexp"         "Mgrad"
## [33] "MGT"          "Msd"          "MSY"          "N0"
## [37] "OFLreal"      "Prob_staying" "procsd"       "qcv"
## [41] "qinc"         "RefY"         "Rmaxlen"      "Size_area_1"
## [45] "SizeLimFrac"  "SizeLimSD"    "Spat_targ"    "SSB0"
## [49] "SSBMSY"       "SSBMSY_SSB0"  "t0"           "TACFrac"
## [53] "TACSD"        "TAEFrac"      "TAESD"        "Vmaxlen"
```

If you use the `dim` function to report the dimensions of the `OM` data frame, you'll notice that there are 56 columns, corresponding to the 56 parameters in the Operating Model, and 200 rows, each corresponding to a single simulation of the MSE.

More information on the `MSE@OM` slot can be found in the help documentation: `class?MSE`

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

The Obs data frame also has 200 rows, each corresponding to a single simulation. More information on the MSE@Obs slot can be found in the help documentation: class?MSE

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.

14.4 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):

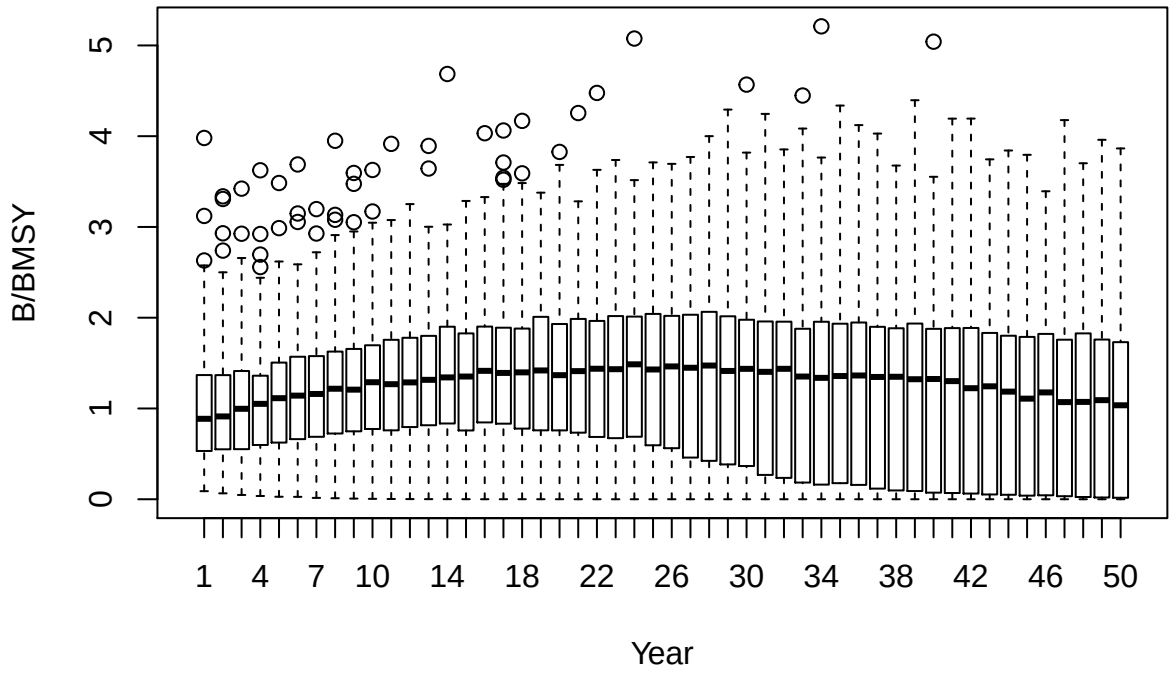
```
dim(BSharkMSE@B_BMSY)
```

```
## [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 0.89 in the first year to 1.04 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.

14.5 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.

14.6 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.

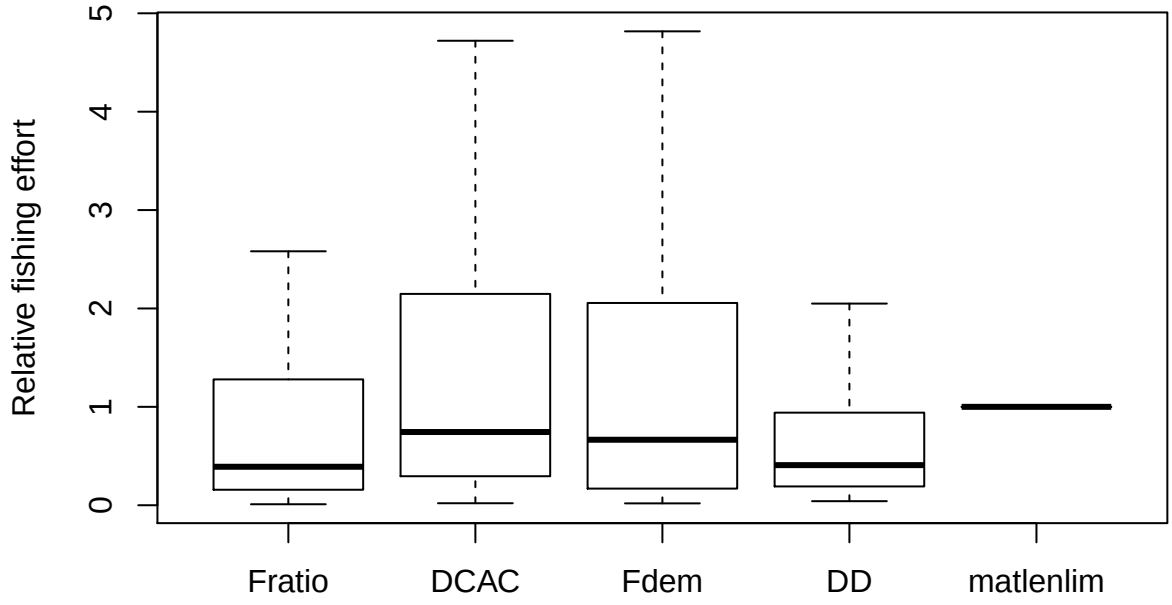
The `apply` function can be used to aggregate these data over the age-classes or spatial areas.

14.7 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 0.74 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 15

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.

15.1 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.

15.2 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).

15.2.1 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).

15.3 Commonly used Performance Metrics

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.

15.3.1 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).

15.3.2 Stock Biomass

Another common performance metric 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.

15.3.3 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.

15.4 Performance Metrics Methods

DLMtool includes a set of functions, of class `PM`, for calculating Performance Metrics. The available `PM` functions (referred to as PMs) can be found using the `avail` function:

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

```
##  [1] "AAVY"  "LTY"   "P10"   "P100"  "P50"   "PNOF"  "STY"   "Yield"
##  [9] "MeanB" "MeanF"
```

The PMs are used for summarizing the performance of the management procedures and plotting the results in trade-off plots.

Here we briefly describe the built-in Performance Metrics functions and demonstrate their use. Advanced DLMtool users can develop their own `PM` methods, see the Custom Performance Metrics chapter for details.

Functions of class `PM` are used on an object of class `MSE` (i.e the object returned by `runMSE`), and return an object of class `PMobj`. Most of the time the `PM` functions are used internally in the `summary` or plotting functions, and it will not be neccessary to acess the `PMobj` directly.

To demonstrate the `PM` functions we first run a quick example MSE:

```
MSE <- runMSE()
```

15.4.1 Overview of the `PM` Functions

We will use the `P50` function to demonstrate the `PM` methods. Help documentation on the PM methods can be accessed in the usual way: `?P50`.

The `P50` PM method calculates the probability that spawning biomass is above half of the spawning biomass that results in maximum sustainable yield ($SB > 0.5SB_{MSY}$).

Applying the `P50` function to our `MSE` object results in the following output:

```
P50(MSE)
```

```
## Spawning Biomass relative to SBMSY
##  Prob. SB > 0.5 SBMSY (Years 1 - 50)
##     AvC DCAC FMSYref curE matlenlim MRreal
## 1  0.02 0.92     0.9    0      0.96   0.06
## 2     1  0.6       1    1         1      1
## 3  0.94  0.8    0.94    1         1      1
## 4     1    1       1    1         1      1
```

```
## 5  0.96 0.96    0.54    0      0.96      0
## 6     1    1       1    1         1      1
## 7  0.04 0.12    0.98 0.68         1   0.82
## 8     1    1       1    1         1      1
## 9     1    1       1    1         1      1
## 10 0.94 0.92    0.96 0.08      0.98   0.52
## 11    .    .       .    .         .      .
## 12    .    .       .    .         .      .
## 13    .    .       .    .         .      .
## 48    1    1       1    1         1      1
##
## Mean
## [1] 0.68 0.78 0.93 0.80 0.99 0.85
```

We can see that the `PM` function calculated, for the 6 MPs in the `MSE` object, the probability SB $> 0.5\text{SB}_{\text{MSY}}$ for all 50 projection years.

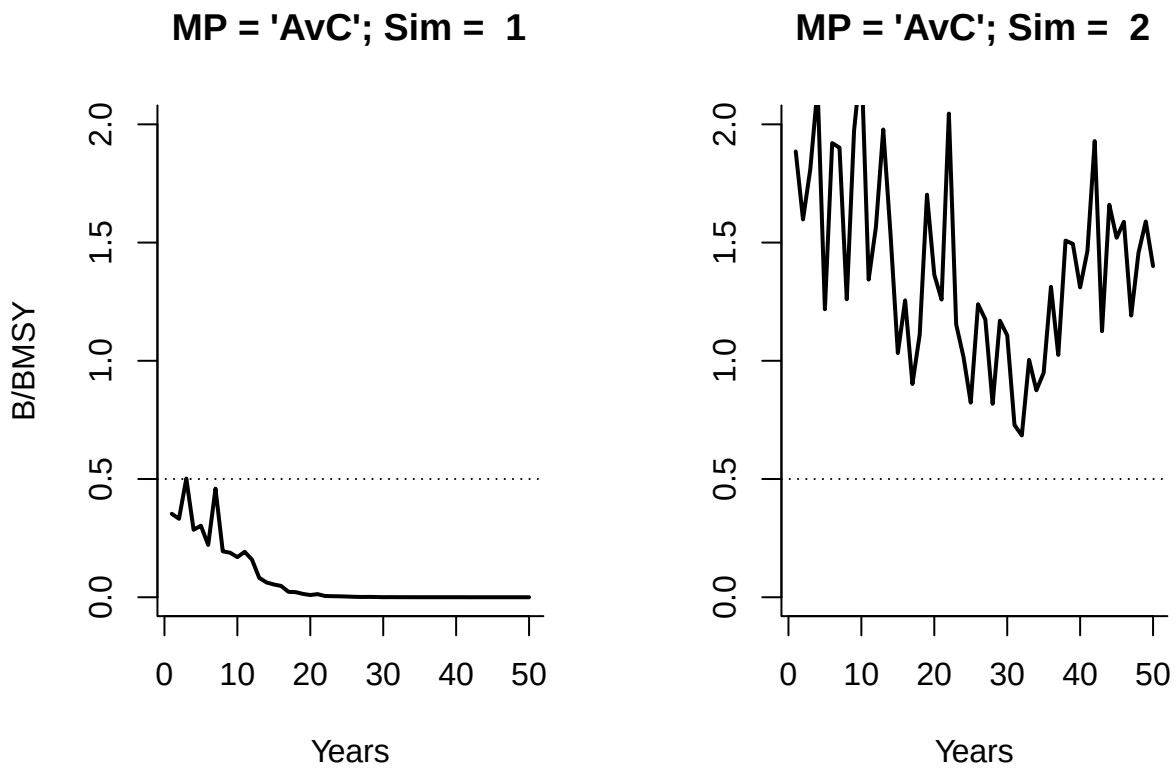
The `PM` function prints out a summary table of the performance metrics statistics for the first 10 simulations and the last simulation (48 in this case) for each MP. The final line shows the overall probability of the performance metric, i.e the average performance across all simulations.

We will look into this output in a little more detail.

We can see that the first MP is AvC and the performance statistics for the first and second simulations are 0.02 and 1. How have these values been calculated and what do they mean?

Let's first plot the spawning biomass relative to BMSY for the first two simulations of the `AvC` MP:

```
par(mfrow=c(1,2))
plot(1:MSE@proyears, MSE@B_BMSY[1,1,], type='l',
     xlab="Years", ylab="B/BMSY", lwd=2, bty="l", ylim=c(0,2),
     main="MP = 'AvC'; Sim =  1")
abline(h=0.5, lty=3)

plot(1:MSE@proyears, MSE@B_BMSY[2,1,], type='l',
     xlab="Years", ylab='', lwd=2, bty="l", ylim=c(0,2),
     main="MP = 'AvC'; Sim =  2")
abline(h=0.5, lty=3)
```



Now we will calculate fraction of years where spawning biomass is above 0.5 SB_{MSY} for the first and second simulations:

```
mean(MSE@B_BMSY[1,1,] > 0.5) # first simulation
```

```
## [1] 0.02
```

```
mean(MSE@B_BMSY[2,1,] > 0.5) # second simulation
```

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

```
# identical to:
# sum(MSE@B_BMSY[1,1,] > 0.5)/MSE@proyears
# sum(MSE@B_BMSY[2,1,] > 0.5)/MSE@proyears
```

Notice how the performance statistics for each simulation correspond with the plot shown above?

The overall performance is then calculated by the probability over all simulations, i.e for the first MP AvC:

```
mean(MSE@B_BMSY[,1,]>0.5)
```

```
## [1] 0.6829167
```

And for 6 MPs:

```
round(apply(MSE@B_BMSY >0.5, 2, mean),2)
```

```
## [1] 0.68 0.78 0.93 0.80 0.99 0.85
```

which, reassuringly, is the same as the output of the `P50` function.

15.4.2 Customizing the PM Functions

The PM functions allow for very quick calculation of performance metrics. For example, suppose that instead of calculating performance over all projection years, we are only interested in the long-term performance, say over the last 10 years. This can be easily achieved using the Yrs argument in the PM function:

```
P50(MSE, Yrs=c(41,50))
```

```
## Spawning Biomass relative to SBMSY
##  Prob. SB > 0.5 SBMSY (Years 41 - 50)
##    AvC DCAC FMSYref curE matlenlim MRreal
## 1    0    1       1    0         1      0
## 2    1    0       1    1         1      1
## 3  0.7    0     0.8    1         1      1
## 4    1    1       1    1         1      1
## 5    1    1     0.3    0         1      0
## 6    1    1       1    1         1      1
## 7    0    0       1    1         1      1
## 8    1    1       1    1         1      1
## 9    1    1       1    1         1      1
## 10   1  0.7       1    0         1    0.4
## 11   .    .       .    .         .      .
## 12   .    .       .    .         .      .
## 13   .    .       .    .         .      .
## 48   1    1       1    1         1      1
##
## Mean
## [1] 0.59 0.55 0.88 0.79 1.00 0.85
```

Or the first 10 years:

```
P50(MSE, Yrs=c(1,10))
```

```
## Spawning Biomass relative to SBMSY
##  Prob. SB > 0.5 SBMSY (Years 1 - 10)
##    AvC DCAC FMSYref curE matlenlim MRreal
## 1  0.1  0.6     0.5    0       0.8    0.2
## 2    1    1       1    1         1      1
## 3    1    1       1    1         1      1
## 4    1    1       1    1         1      1
## 5  0.8  0.8     0.7    0       0.8      0
## 6    1    1       1    1         1      1
## 7  0.2  0.6     0.9  0.4         1    0.4
## 8    1    1       1    1         1      1
## 9    1    1       1    1         1      1
## 10 0.7  0.9     0.8  0.4       0.9    0.7
## 11   .    .       .    .         .      .
## 12   .    .       .    .         .      .
## 13   .    .       .    .         .      .
## 48   1    1       1    1         1      1
##
## Mean
## [1] 0.83 0.92 0.92 0.80 0.97 0.83
```

The other biomass Performance Metric functions work in the same way:

```
P10(MSE) # probability SB > 0.1SB_MSY for all years
```

```
## Spawning Biomass relative to SBMSY
##  Prob. SB > 0.1 SBMSY (Years 1 - 50)
##     AvC DCAC FMSYref curE matlenlim MRreal
## 1  0.24    1       1 0.42         1      1
## 2     1 0.98       1    1         1      1
## 3     1    1       1    1         1      1
## 4     1    1       1    1         1      1
## 5     1    1       1 0.38         1   0.96
## 6     1    1       1    1         1      1
## 7   0.2 0.32       1    1         1      1
## 8     1    1       1    1         1      1
## 9     1    1       1    1         1      1
## 10    1    1       1    1         1      1
## 11    .    .       .    .         .      .
## 12    .    .       .    .         .      .
## 13    .    .       .    .         .      .
## 48    1    1       1    1         1      1
##
## Mean
## [1] 0.79 0.89 1.00 0.96 1.00 1.00
```

```
P100(MSE) # probability SB > SB_MSY for all years
```

```
## Spawning Biomass relative to SBMSY
##  Prob. SB > SBMSY (Years 1 - 50)
##     AvC DCAC FMSYref curE matlenlim MRreal
## 1     0  0.8    0.24    0      0.62      0
## 2  0.86 0.46    0.82    1         1      1
## 3  0.78 0.76     0.2 0.98         1      1
## 4  0.92 0.86    0.84 0.96         1   0.96
## 5   0.6 0.52       0    0      0.56      0
## 6     1    1    0.84    1         1      1
## 7     0    0    0.04 0.02      0.84   0.06
## 8     1    1    0.58    1         1      1
## 9  0.86 0.92    0.72 0.96      0.98   0.96
## 10 0.86 0.64     0.5    0      0.94      0
## 11    .    .       .    .         .      .
## 12    .    .       .    .         .      .
## 13    .    .       .    .         .      .
## 48    1    1    0.76 0.98      0.98   0.98
##
## Mean
## [1] 0.59 0.65 0.37 0.65 0.93 0.68
```

Long-term, short-term and overall average yield are calculated using `LTY`, `STY` and `Yield` respectively:

```
LTY(MSE)
```

```
## Average Yield relative to Reference Yield (Years 41-50)
##  Prob. Yield > 0.5 Ref. Yield (Years 41-50)
##    AvC DCAC FMSYref curE matlenlim MRreal
## 1    0    1       1    0         1      0
## 2    1  0.6       1  0.9       0.2    0.9
## 3    1  0.9       1    1         1      1
```

```
## 4    1    1       1    1         0      1
## 5    1    1       1    0         1    0.1
## 6    0  0.6       1  0.1       0.1    0.1
## 7    0    0       1    1       0.5      1
## 8    0    1       1  0.8         0    0.8
## 9    1    1       1    1       0.2      1
## 10   1    1       1    1         1      1
## 11   .    .       .    .         .      .
## 12   .    .       .    .         .      .
## 13   .    .       .    .         .      .
## 48   0    1       1  0.8         0    0.7
##
## Mean
## [1] 0.39 0.65 1.00 0.80 0.56 0.82
```

```
STY(MSE)
```

```
## Average Yield relative to Reference Yield (Years 1-10)
##  Prob. Yield > 0.5 Ref. Yield (Years 1-10)
##    AvC DCAC FMSYref curE matlenlim MRreal
## 1    1    1       1    1       0.4    0.9
## 2    1    1       1    1       0.2      1
## 3    1    1       1    1       0.9      1
## 4    1    1       1  0.7         0    0.7
## 5    1  0.6       1    1       0.7      1
## 6    0    0       1  0.1         0    0.1
## 7  0.8    1     0.6  0.8       0.1    0.8
## 8    0  0.2       1  0.9       0.1    0.9
## 9    1  0.6     0.6  0.1         0    0.1
## 10   1  0.6       1  0.9       0.7    0.9
## 11   .    .       .    .         .      .
## 12   .    .       .    .         .      .
## 13   .    .       .    .         .      .
## 48   0    0     0.6  0.6         0    0.6
##
## Mean
## [1] 0.73 0.67 0.90 0.82 0.44 0.80
```

```
Yield(MSE)
```

```
## Yield relative to Reference Yield (Years 1-50)
##  Mean Relative Yield (Years 1-50)
##     AvC DCAC FMSYref curE matlenlim MRreal
## 1  0.26 0.77    0.95  0.4      0.94   0.51
## 2  0.97 0.89       1 0.76      0.39   0.74
## 3   1.2  1.1     1.2  1.1       0.9    1.1
## 4  0.69 0.81    0.91 0.66      0.29   0.65
## 5  0.78 0.77    0.95 0.44      0.92   0.62
## 6  0.37 0.41    0.77 0.37      0.31   0.37
## 7  0.26 0.36    0.79 0.75      0.52   0.73
## 8  0.47 0.64    0.86 0.69      0.31   0.67
## 9  0.64 0.79    0.87 0.73      0.35   0.71
## 10 0.59 0.85    0.86 0.72      0.99   0.71
## 11    .    .       .    .         .      .
## 12    .    .       .    .         .      .
## 13    .    .       .    .         .      .
```

```
## 48  0.4 0.58    0.84 0.64      0.17   0.62
##
## Mean
## [1] 0.67 0.81 1.10 0.83 0.61 0.82
```

The `PNOF` PM function calculates the probability of not overfishing:

```
PNOF(MSE)
```

```
## Probability of not overfishing (F<FMSY)
##  Prob. F < FMSY (Years 1 - 50)
##     AvC DCAC FMSYref curE matlenlim MRreal
## 1     0 0.98    0.32    0      0.78   0.02
## 2  0.86 0.42    0.74    1         1      1
## 3  0.78 0.74     0.4 0.96         1   0.96
## 4     1 0.82    0.66    1         1      1
## 5     1 0.96    0.56    0         1   0.02
## 6     1    1    0.36    1         1      1
## 7     0    0    0.16  0.2         1    0.4
## 8     1    1    0.56 0.98         1   0.98
## 9  0.96 0.88    0.38    1         1      1
## 10 0.92 0.58    0.12    0         1   0.02
## 11    .    .       .    .         .      .
## 12    .    .       .    .         .      .
## 13    .    .       .    .         .      .
## 48    1    1    0.34 0.98         1   0.98
##
## Mean
## [1] 0.60 0.66 0.42 0.68 0.98 0.72
```

Finally, the average annual variability in yield (AAVY) can be calculated with the `AAVY` function:

```
AAVY(MSE)
```

```
## Average Annual Variability in Yield (Years 1-50)
##  Prob. AAVY < 20% (Years 1-50)
##    AvC DCAC FMSYref curE matlenlim MRreal
## 1    0    1       1    0         0      0
## 2    1    1       1    0         0      0
## 3    1    1       1    0         0      0
## 4    1    1       1    0         0      0
## 5    1    1       1    0         0      0
## 6    1    1       1    0         0      0
## 7    1    1       1    0         0      0
## 8    1    1       1    0         0      0
## 9    1    1       1    1         1      1
## 10   1    1       1    0         0      0
## 11   .    .       .    .         .      .
## 12   .    .       .    .         .      .
## 13   .    .       .    .         .      .
## 48   1    1       1    0         0      0
##
## Mean
## [1] 0.90 0.96 1.00 0.17 0.19 0.19
```

By default the `AAVY` PM function calculates the probability that AAVY is less than 20%. This reference level can easily be modified using the `Ref` argument:

```
AAVY(MSE, Ref=0.15) # prob. AAVY < 15%
```

```
## Average Annual Variability in Yield (Years 1-50)
##  Prob. AAVY < 15% (Years 1-50)
##    AvC DCAC FMSYref curE matlenlim MRreal
## 1    0    1       1    0         0      0
## 2    1    1       1    0         0      0
## 3    1    1       1    0         0      0
## 4    1    1       1    0         0      0
## 5    1    1       1    0         0      0
## 6    1    1       1    0         0      0
## 7    1    1       1    0         0      0
## 8    1    1       1    0         0      0
## 9    1    1       1    0         0      0
## 10   1    1       1    0         0      0
## 11   .    .       .    .         .      .
## 12   .    .       .    .         .      .
## 13   .    .       .    .         .      .
## 48   1    1       1    0         0      0
##
## Mean
## [1] 0.830 0.960 1.000 0.062 0.083 0.062
```

```
AAVY(MSE, Ref=0.30) # prob. AAVY < 30%
```

```
## Average Annual Variability in Yield (Years 1-50)
##  Prob. AAVY < 30% (Years 1-50)
##    AvC DCAC FMSYref curE matlenlim MRreal
## 1    1    1       1    0         1      1
## 2    1    1       1    0         0      0
## 3    1    1       1    0         0      0
## 4    1    1       1    0         0      0
## 5    1    1       1    1         1      1
## 6    1    1       1    0         0      0
## 7    1    1       1    1         1      1
## 8    1    1       1    1         1      1
## 9    1    1       1    1         1      1
## 10   1    1       1    0         1      0
## 11   .    .       .    .         .      .
## 12   .    .       .    .         .      .
## 13   .    .       .    .         .      .
## 48   1    1       1    0         0      0
##
## Mean
## [1] 0.98 1.00 1.00 0.56 0.60 0.58
```

The other `PM` functions also have the `Ref` argument which can be used in the same way. For example, you may notice that the `P50` and `P100` functions are identical except for the value of the `Ref` argument:

```
args(P50)
```

```
## function (MSEobj = NULL, Ref = 0.5, Yrs = NULL)
## NULL
```

```
args(P100)
```

```
## function (MSEobj = NULL, Ref = 1, Yrs = NULL)
## NULL
```

It follows then that it is very simple to calculate a custom performance metric based on the built-in `PM` functions. For example, suppose we wanted to calculate the probability that spawning biomass was above 5% of BMSY. This can be achieved by using any of the biomass-based `PM` functions and modifying the `Ref` argument:

```
P50(MSE, Ref=0.05)
```

```
## Spawning Biomass relative to SBMSY
##  Prob. SB > 0.05 SBMSY (Years 1 - 50)
##    AvC DCAC FMSYref curE matlenlim MRreal
## 1  0.3    1       1 0.56         1      1
## 2    1    1       1    1         1      1
## 3    1    1       1    1         1      1
## 4    1    1       1    1         1      1
## 5    1    1       1 0.48         1      1
## 6    1    1       1    1         1      1
## 7  0.3 0.58       1    1         1      1
## 8    1    1       1    1         1      1
## 9    1    1       1    1         1      1
## 10   1    1       1    1         1      1
## 11   .    .       .    .         .      .
## 12   .    .       .    .         .      .
## 13   .    .       .    .         .      .
## 48   1    1       1    1         1      1
##
## Mean
## [1] 0.84 0.93 1.00 0.97 1.00 1.00
```

```
P100(MSE, Ref=0.05)
```

```
## Spawning Biomass relative to SBMSY
##  Prob. SB > 0.05 SBMSY (Years 1 - 50)
##    AvC DCAC FMSYref curE matlenlim MRreal
## 1  0.3    1       1 0.56         1      1
## 2    1    1       1    1         1      1
## 3    1    1       1    1         1      1
## 4    1    1       1    1         1      1
## 5    1    1       1 0.48         1      1
## 6    1    1       1    1         1      1
## 7  0.3 0.58       1    1         1      1
## 8    1    1       1    1         1      1
## 9    1    1       1    1         1      1
## 10   1    1       1    1         1      1
## 11   .    .       .    .         .      .
## 12   .    .       .    .         .      .
## 13   .    .       .    .         .      .
## 48   1    1       1    1         1      1
##
## Mean
## [1] 0.84 0.93 1.00 0.97 1.00 1.00
```

More information on customizing `PM` functions can be found in the Custom Performance Metrics chapter.

In the next section we will demonstrate using `PM` functions in summarizing and plotting functions.

15.5 Summarizing Management Procedure Performance

The Examining the MSE Results chapter introduced the `summary` function for `MSE` objects and some of the plotting functions for visualizing the results. Here we demonstrate how the `PM` functions can be used in the `summary` function and the trade-off plots:

15.5.1 `summary` Table

The `summary` function provides information on the performance of the Management Procedures with respect to the performance metrics. By default, `summary` includes the `PNOF`, `P50`, `AAVY` and `LTY` performance metrics:

```
summary(MSE)
```

```
## Calculating Performance Metrics
```

```
##                                       Performance.Metrics
## 1                 Probability of not overfishing (F<FMSY)
## 2                      Spawning Biomass relative to SBMSY
## 3        Average Annual Variability in Yield (Years 1-50)
## 4 Average Yield relative to Reference Yield (Years 41-50)
##
## 1              Prob. F < FMSY (Years 1 - 50)
## 2        Prob. SB > 0.5 SBMSY (Years 1 - 50)
## 3              Prob. AAVY < 20% (Years 1-50)
## 4 Prob. Yield > 0.5 Ref. Yield (Years 41-50)
##
##
## Probability:
##          MP PNOF  P50 AAVY  LTY
## 1       AvC 0.60 0.68 0.90 0.39
## 2      DCAC 0.66 0.78 0.96 0.65
## 3   FMSYref 0.42 0.93 1.00 1.00
## 4      curE 0.68 0.80 0.17 0.80
## 5 matlenlim 0.98 0.99 0.19 0.56
## 6    MRreal 0.72 0.85 0.19 0.82
```

It is straightforward to include other `PM` functions by adding the names of the `PM` functions, for example:

```
summary(MSE, 'P100', 'Yield')
```

```
## Calculating Performance Metrics
```

```
##                               Performance.Metrics
## 1             Spawning Biomass relative to SBMSY
## 2 Yield relative to Reference Yield (Years 1-50)
##
## 1  Prob. SB > SBMSY (Years 1 - 50)
## 2 Mean Relative Yield (Years 1-50)
##
##
## Probability:
##          MP P100 Yield
## 1       AvC 0.59  0.67
## 2      DCAC 0.65  0.81
## 3   FMSYref 0.37  1.10
## 4      curE 0.65  0.83
```

```
## 5 matlenlim 0.93  0.61
## 6    MRreal 0.68  0.82
```

or all available PM functions:

```
summary(MSE, avail('PM'))
```

```
## Calculating Performance Metrics

##                                        Performance.Metrics
## 1         Average Annual Variability in Yield (Years 1-50)
## 2  Average Yield relative to Reference Yield (Years 41-50)
## 3                       Spawning Biomass relative to SBMSY
## 4                       Spawning Biomass relative to SBMSY
## 5                       Spawning Biomass relative to SBMSY
## 6                 Probability of not overfishing (F<FMSY)
## 7   Average Yield relative to Reference Yield (Years 1-10)
## 8           Yield relative to Reference Yield (Years 1-50)
## 9                       Spawning Biomass relative to SBMSY
## 10                      Fishing Mortality relative to FMSY
##
## 1               Prob. AAVY < 20% (Years 1-50)
## 2  Prob. Yield > 0.5 Ref. Yield (Years 41-50)
## 3         Prob. SB > 0.1 SBMSY (Years 1 - 50)
## 4             Prob. SB > SBMSY (Years 1 - 50)
## 5         Prob. SB > 0.5 SBMSY (Years 1 - 50)
## 6               Prob. F < FMSY (Years 1 - 50)
## 7   Prob. Yield > 0.5 Ref. Yield (Years 1-10)
## 8            Mean Relative Yield (Years 1-50)
## 9               Mean SB/SBMSY (Years 46 - 50)
## 10                Mean F/FMSY (Years 46 - 50)
##
##
## Probability:
##          MP AAVY  LTY  P10 P100  P50 PNOF  STY Yield MeanB MeanF
## 1       AvC 0.90 0.39 0.79 0.59 0.68 0.60 0.73  0.67  1.40  1.20
## 2      DCAC 0.96 0.65 0.89 0.65 0.78 0.66 0.67  0.81  0.88  1.40
## 3   FMSYref 1.00 1.00 1.00 0.37 0.93 0.42 0.90  1.10  0.91  1.00
## 4      curE 0.17 0.80 0.96 0.65 0.80 0.68 0.82  0.83  1.40  0.93
## 5 matlenlim 0.19 0.56 1.00 0.93 0.99 0.98 0.44  0.61  2.30  0.34
## 6    MRreal 0.19 0.82 1.00 0.68 0.85 0.72 0.80  0.82  1.50  0.78
```

The summary function returns a data frame which can be useful for referring to the PM results elsewhere in the analysis. For example,

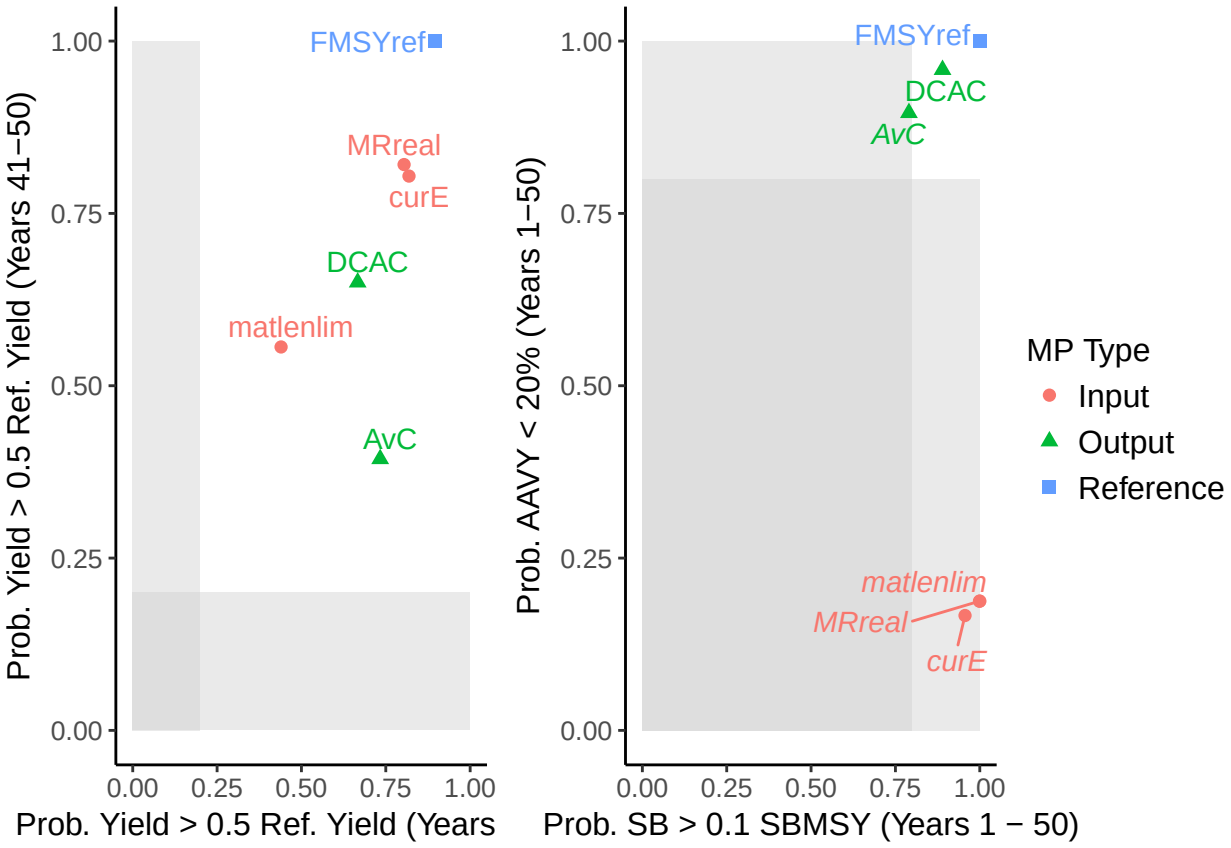
```
Results <- summary(MSE, avail('PM'), silent=TRUE) # silent=TRUE to hide print-out to console
Results$Yield # access the PM results
```

```
## [1] 0.67 0.81 1.10 0.83 0.61 0.82
```

15.5.2 Trade-Off Plots

The TradePlot function takes an object of class MSE and the names of PM functions (at least 2) to produce a trade-off plot. For example:

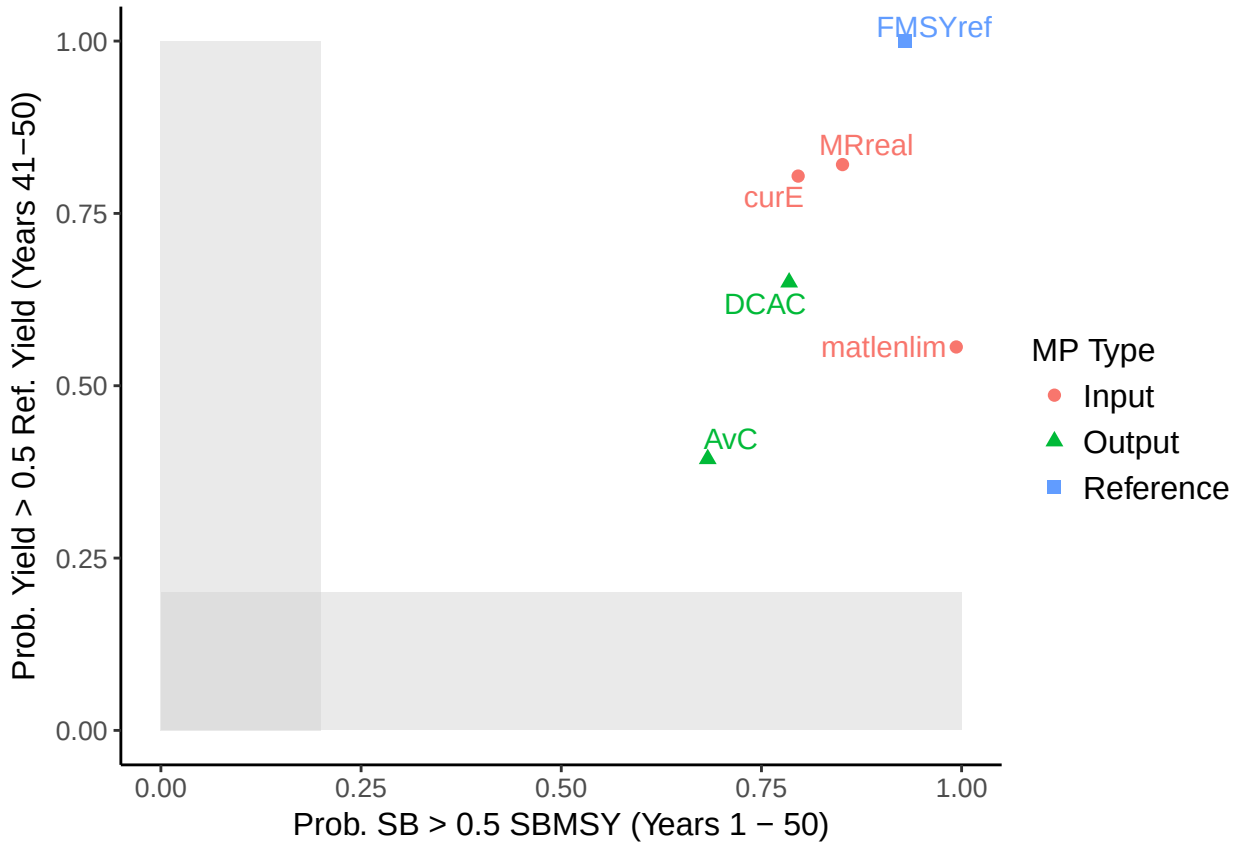
```
TradePlot(MSE) # default plot
```




```
##          MP  STY  LTY  P10 AAVY Satisificed
## 1       AvC 0.73 0.39 0.79 0.90       FALSE
## 2      DCAC 0.67 0.65 0.89 0.96        TRUE
## 3   FMSYref 0.90 1.00 1.00 1.00        TRUE
## 4      curE 0.82 0.80 0.96 0.17       FALSE
## 5 matlenlim 0.44 0.56 1.00 0.19       FALSE
## 6    MRreal 0.80 0.82 1.00 0.19       FALSE
```

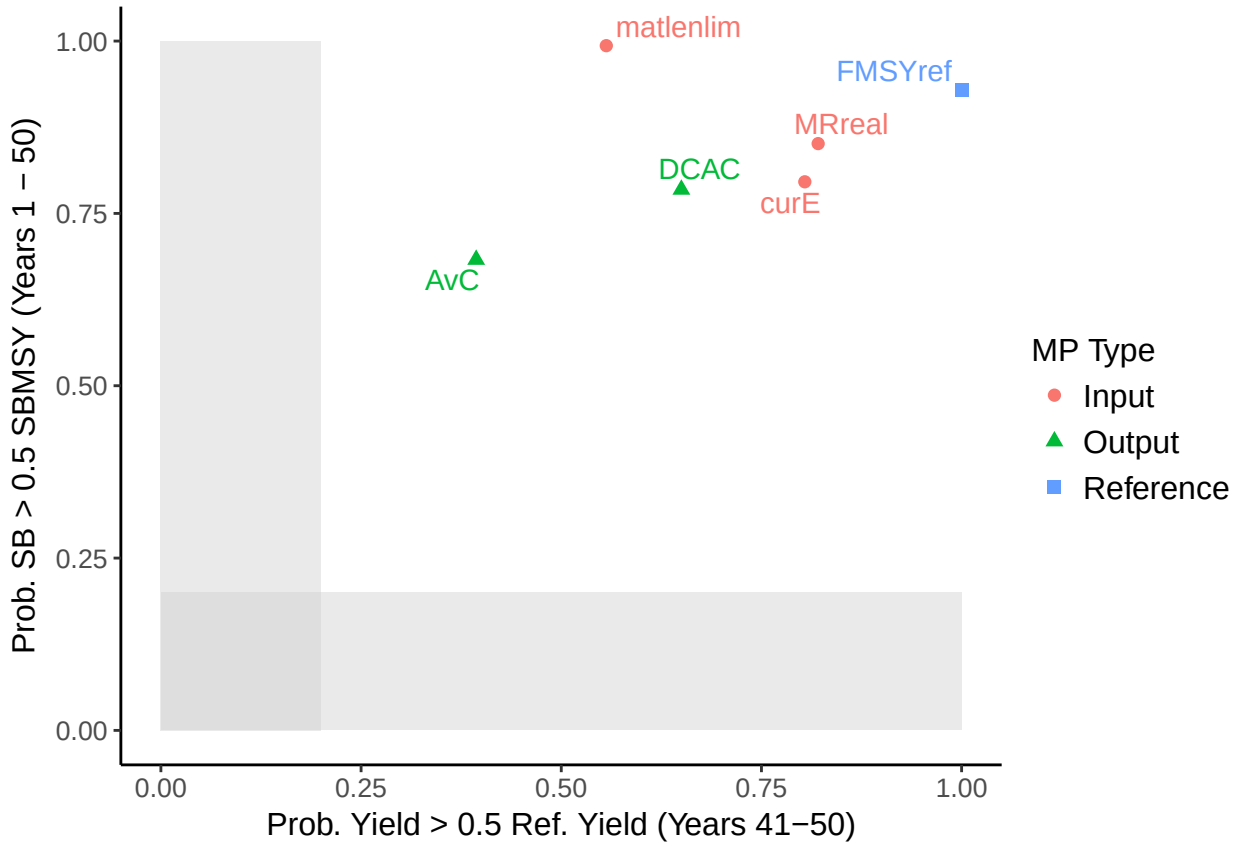
The order of the `PM` function names determines plotting on the x and y axes. For example:

```
TradePlot(MSE, 'P50' ,'LTY')  # x = P50, y  = LTY
```




```
##          MP  P50  LTY Satisificed
## 1       AvC 0.68 0.39        TRUE
## 2      DCAC 0.78 0.65        TRUE
## 3   FMSYref 0.93 1.00        TRUE
## 4      curE 0.80 0.80        TRUE
## 5 matlenlim 0.99 0.56        TRUE
## 6    MRreal 0.85 0.82        TRUE
```

```
TradePlot(MSE, 'LTY' ,'P50')  # x = LTY, y = P50
```

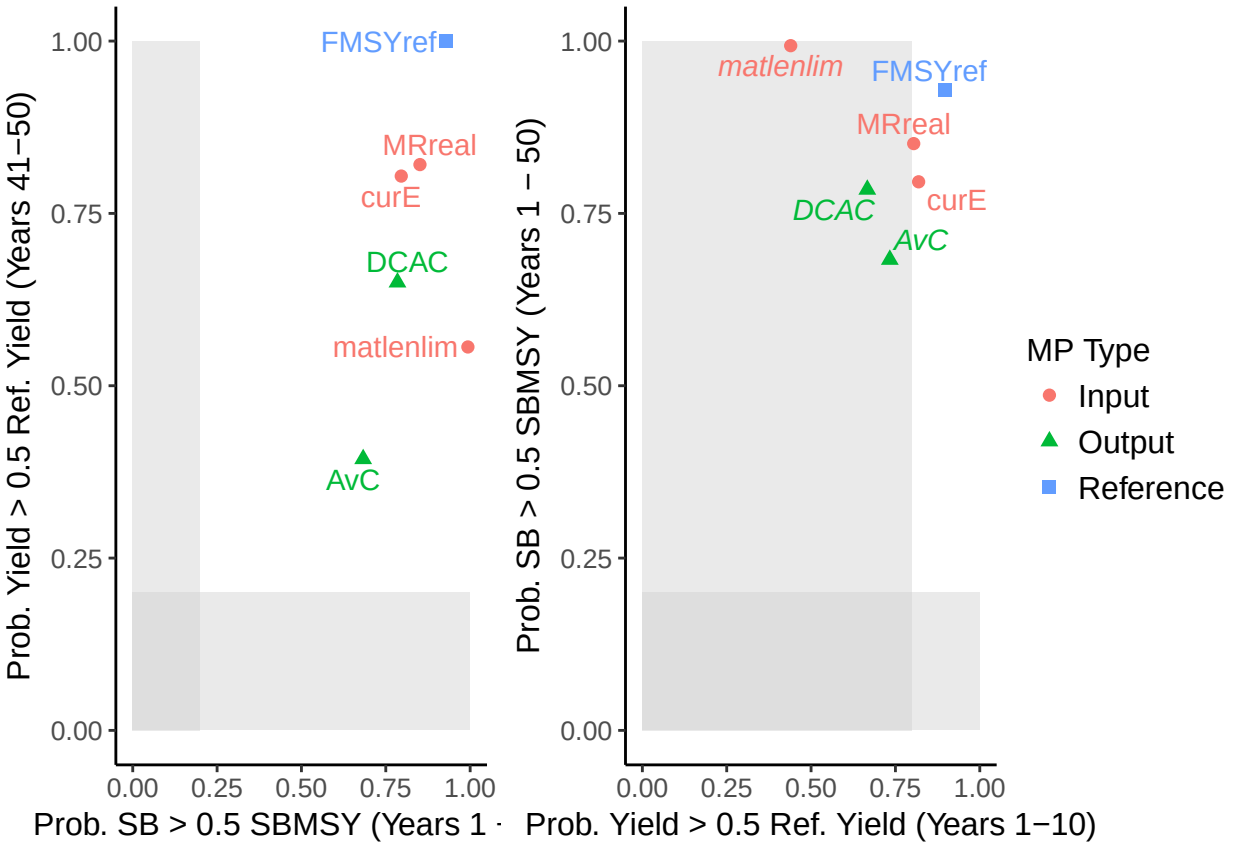



```
##          MP  LTY  P50 Satisificed
## 1       AvC 0.39 0.68        TRUE
## 2      DCAC 0.65 0.78        TRUE
## 3   FMSYref 1.00 0.93        TRUE
## 4      curE 0.80 0.80        TRUE
## 5 matlenlim 0.56 0.99        TRUE
## 6    MRreal 0.82 0.85        TRUE
```

The PMs are recycled if an odd number are provided:

```
TradePlot(MSE, 'P50' ,'LTY', 'STY')
```

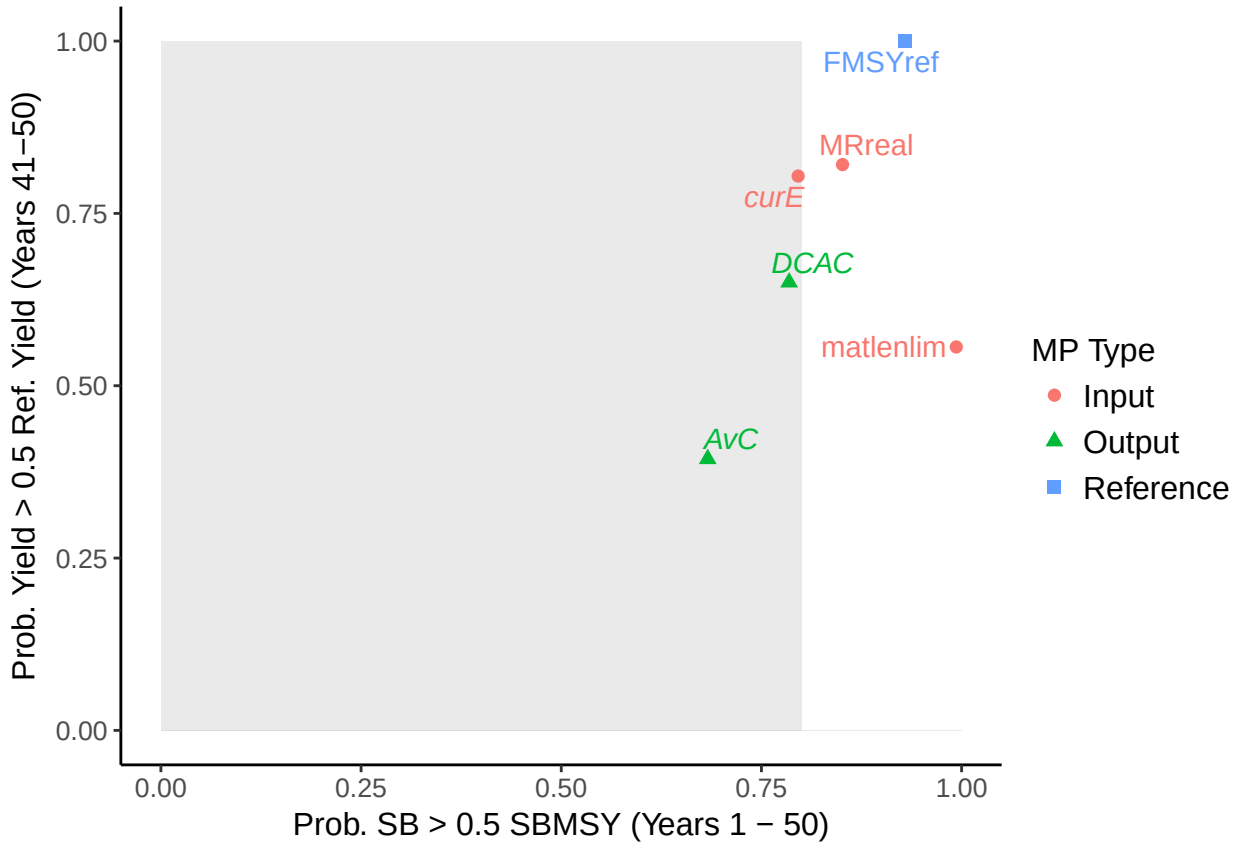
```
## Odd number of PMs. Recycling first PM
```




```
##          MP  P50  LTY  STY Satisificed
## 1       AvC 0.68 0.39 0.73       FALSE
## 2      DCAC 0.78 0.65 0.67       FALSE
## 3   FMSYref 0.93 1.00 0.90        TRUE
## 4      curE 0.80 0.80 0.82        TRUE
## 5 matlenlim 0.99 0.56 0.44       FALSE
## 6    MRreal 0.85 0.82 0.80        TRUE
```

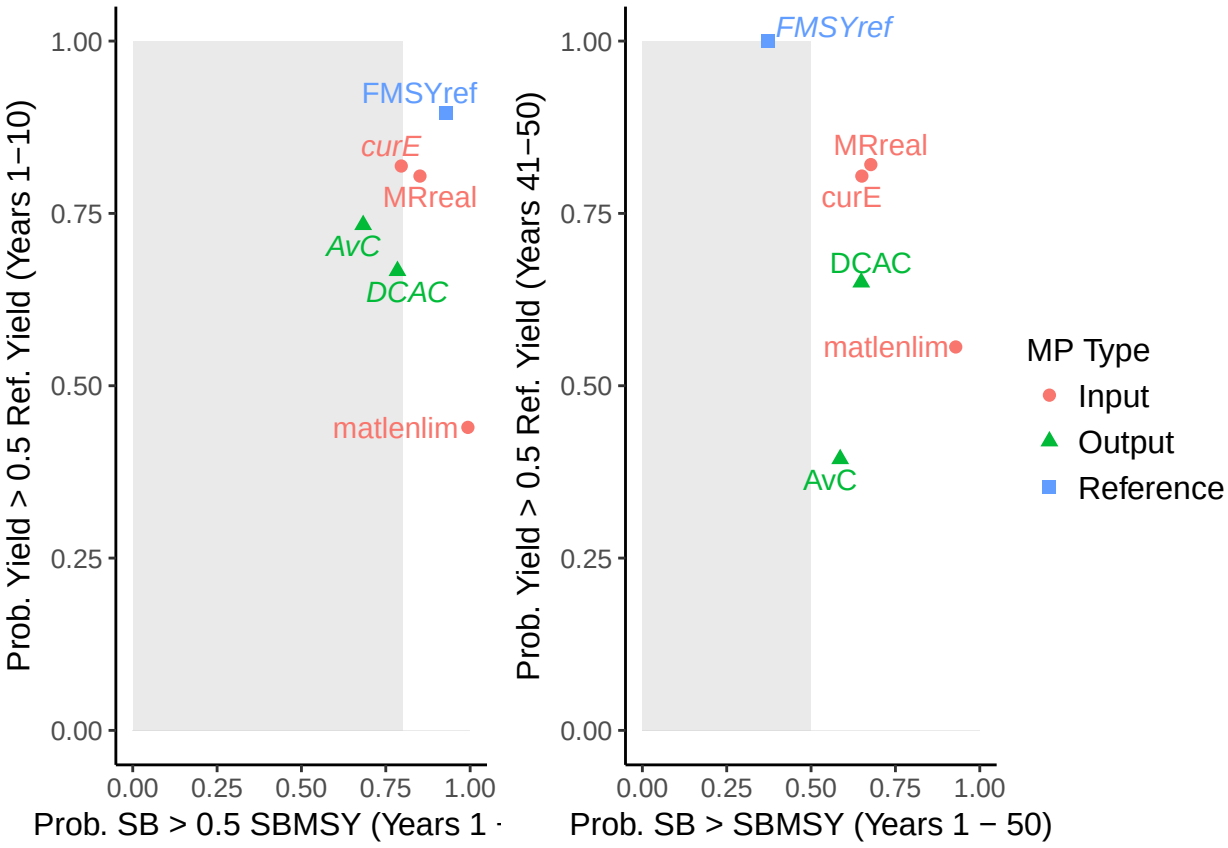
The `Lims` argument is used to set the vertical and horizontal acceptable risk thresholds and are interpreted in the same order as the names of the `PM` functions. For example:

```
TradePlot(MSE, 'P50' ,'LTY', Lims=c(0.8, 0))  # 80% minimum acceptable risk for P50, no minimum for LTY
```




```
##          MP  P50  LTY Satisificed
## 1       AvC 0.68 0.39       FALSE
## 2      DCAC 0.78 0.65       FALSE
## 3   FMSYref 0.93 1.00        TRUE
## 4      curE 0.80 0.80       FALSE
## 5 matlenlim 0.99 0.56        TRUE
## 6    MRreal 0.85 0.82        TRUE
```

```
TradePlot(MSE, 'P50' ,'STY', 'P100', 'LTY', Lims=c(0.8, 0, 0.5, 0))  # 80% minimum acceptable risk for
```




```
##          MP  P50  STY P100  LTY Satisificed
## 1       AvC 0.68 0.73 0.59 0.39       FALSE
## 2      DCAC 0.78 0.67 0.65 0.65       FALSE
## 3   FMSYref 0.93 0.90 0.37 1.00       FALSE
## 4      curE 0.80 0.82 0.65 0.80       FALSE
## 5 matlenlim 0.99 0.44 0.93 0.56        TRUE
## 6    MRreal 0.85 0.80 0.68 0.82        TRUE
```

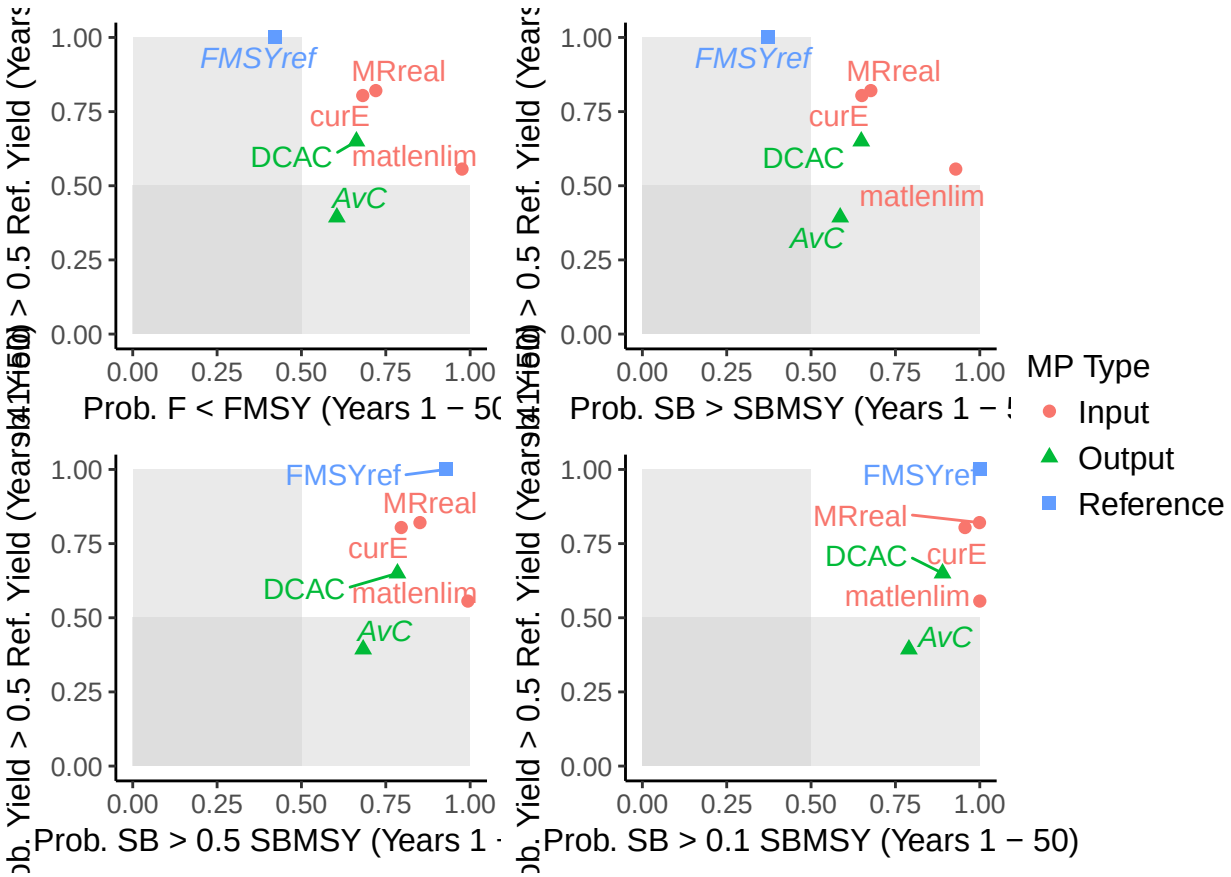
The `TradePlot` function returns a data frame with the results of the performance metrics, and a column indicating if an MP has met minimum performance criteria for **all** performance metrics. In the previous example, 2 MPs (matlenlim, MRreal) met the minimum performance criteria for all four performance metrics.

The `TradePlot` function can be used to make a variety of custom trade-off plots. For example, the `Tplot`, `Tplot2`, and `Tplot3` functions all use this function to produce different trade-off plots:

```
Tplot
```

```
## function(MSEobj, Lims=c(0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5), ...) {
##   if (class(Lims)!="numeric") stop("Second argument must be numeric")
##   TradePlot(MSEobj, Lims=Lims, PMlist=list("PNOF", "LTY", "P100", "LTY", "P50", "LTY", "P10", "LTY")
## }
## <bytecode: 0x0000000021881928>
## <environment: namespace:DLMtool>
```

```
Tplot(MSE)
```



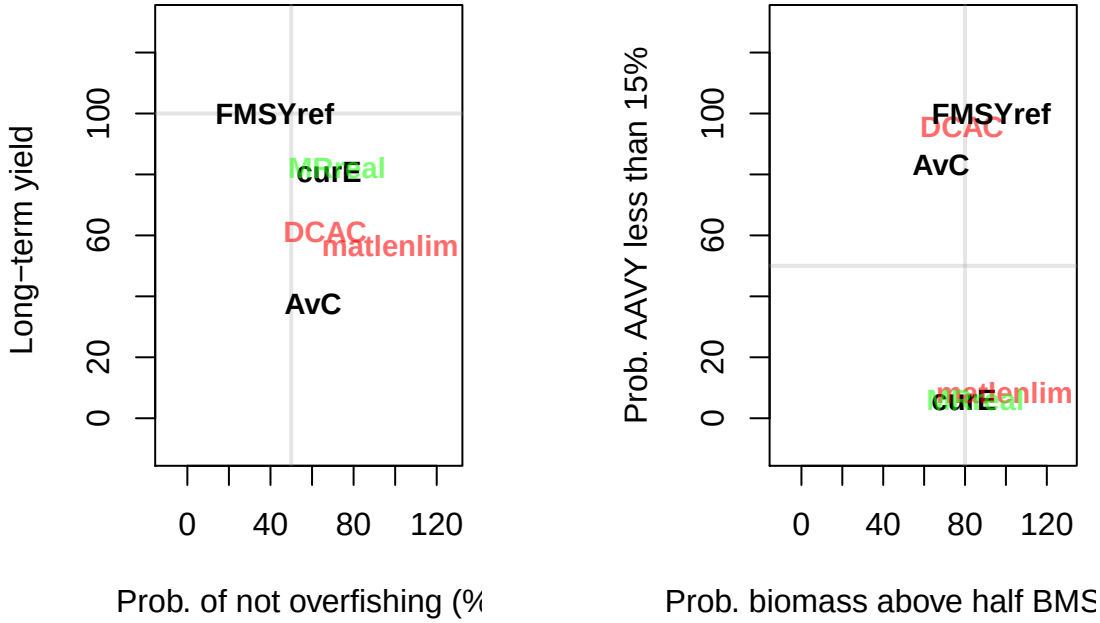

```
##          MP PNOF  LTY P100  P50  P10 Satisificed
## 1       AvC 0.60 0.39 0.59 0.68 0.79       FALSE
## 2      DCAC 0.66 0.65 0.65 0.78 0.89        TRUE
## 3   FMSYref 0.42 1.00 0.37 0.93 1.00       FALSE
## 4      curE 0.68 0.80 0.65 0.80 0.96        TRUE
## 5 matlenlim 0.98 0.56 0.93 0.99 1.00        TRUE
## 6    MRreal 0.72 0.82 0.68 0.85 1.00        TRUE
```

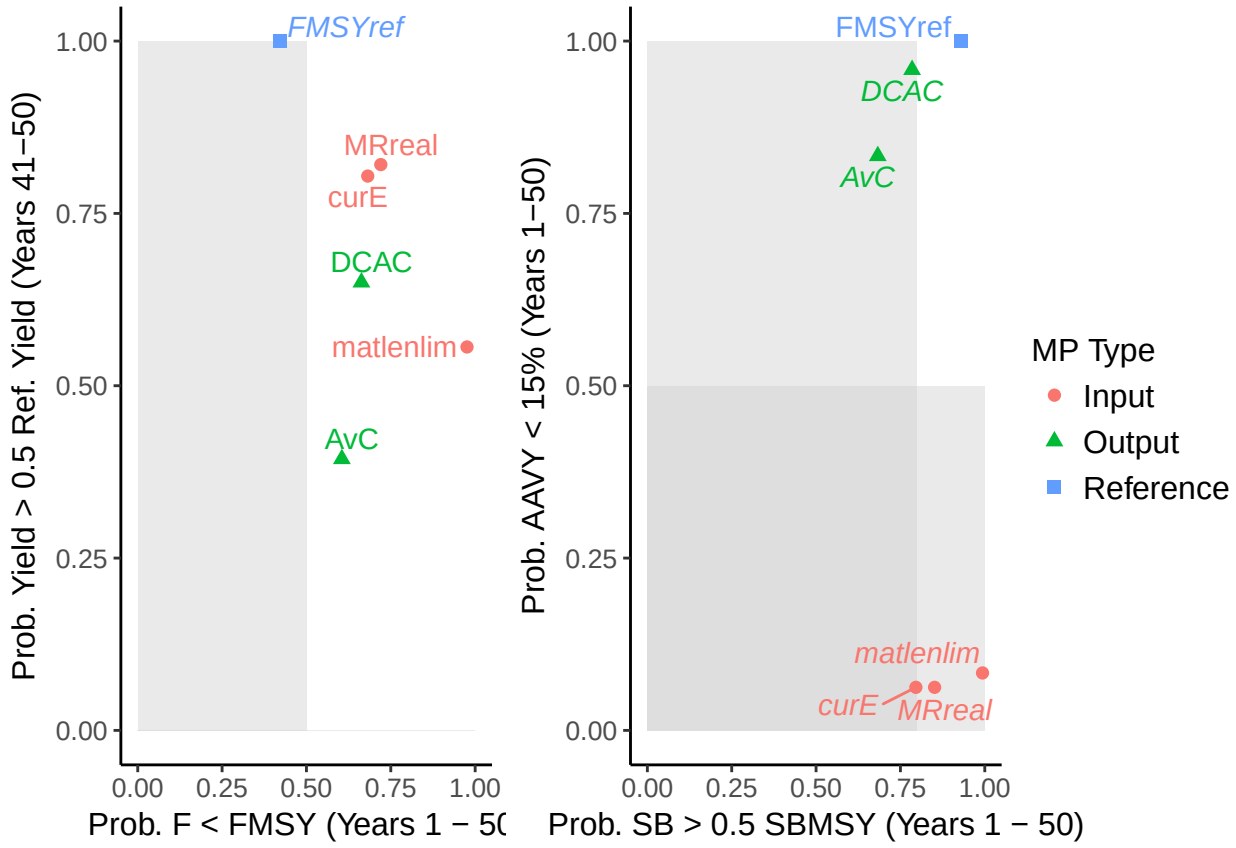
Similarly, we can easily reproduce `NOAA_plot` using the `Tradeplot` function:

```
NOAA_plot(MSE)
```

```
##           PNOF  B50   LTY    VY
## AvC       60.5 68.3  37.5  83.3
## DCAC      66.3 78.5  61.2  95.8
## FMSYref   42.1 92.9 100.0 100.0
## curE      68.2 79.6  80.8   6.2
## matlenlim 97.6 99.3  56.7   8.3
## MRreal    72.0 85.1  82.1   6.2
```

```
TradePlot(MSE, Lims=c(0.5, 0, 0.8, 0.5),
          PMlist=list("PNOF", "LTY", "P50", "AAVY"), Refs=list(AAVY=0.15))
```


Long−term yield
0
20
60
100
FMSYref
curE
DCAC
matlenlim
AvC
0
40
80
120
Prob. of not overfishing (%
Prob. AAVY less than 15%
0
20
60
100
FMSYref
DCAC
AvC
matlenlim
curE
0
40
80
120
Prob. biomass above half BMS




```
##          MP PNOF  LTY  P50  AAVY Satisificed
## 1       AvC 0.60 0.39 0.68 0.830       FALSE
## 2      DCAC 0.66 0.65 0.78 0.960       FALSE
## 3   FMSYref 0.42 1.00 0.93 1.000       FALSE
## 4      curE 0.68 0.80 0.80 0.062       FALSE
## 5 matlenlim 0.98 0.56 0.99 0.083       FALSE
## 6    MRreal 0.72 0.82 0.85 0.062       FALSE
```

See the Plotting 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 16

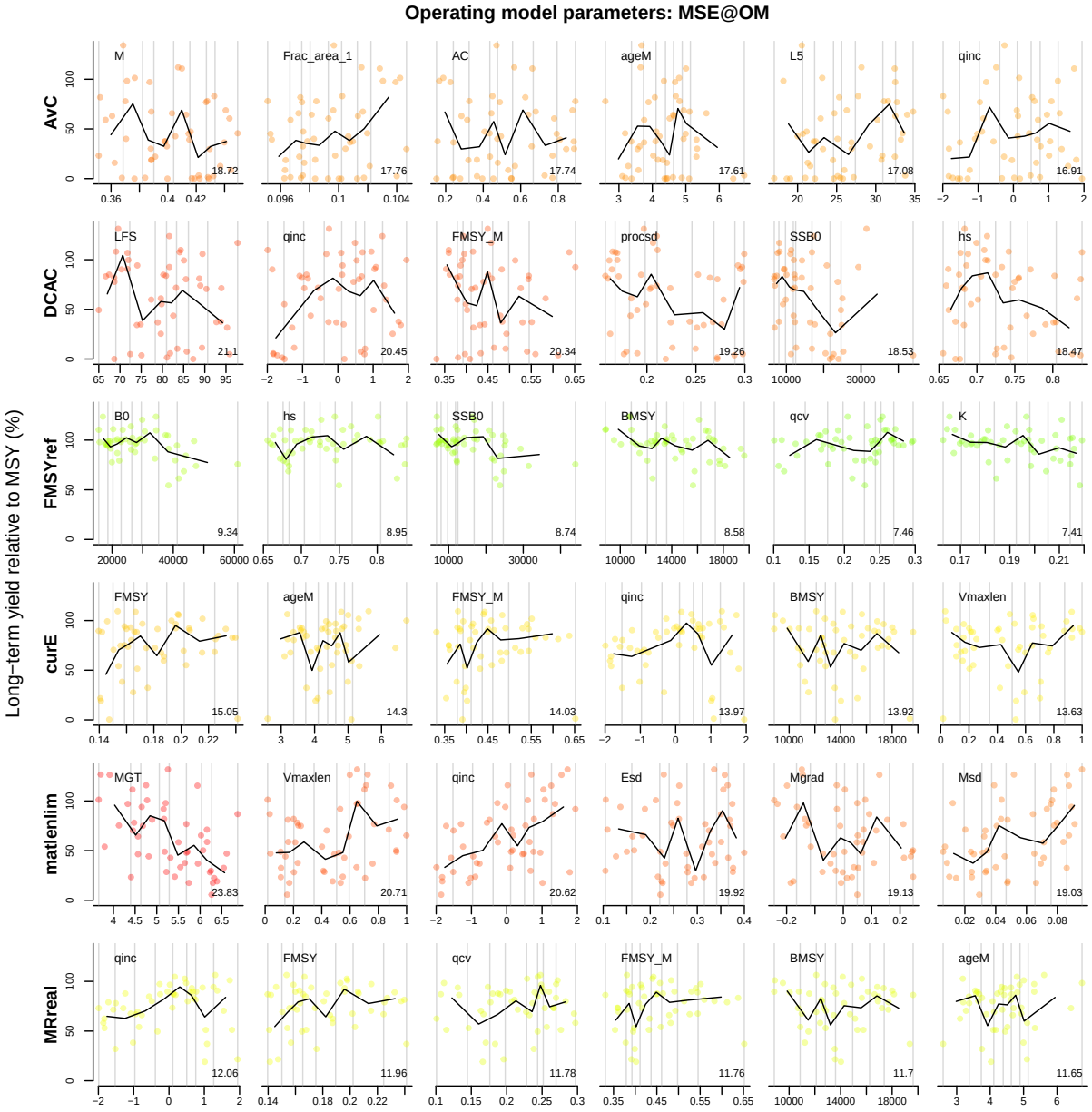
Value of Information

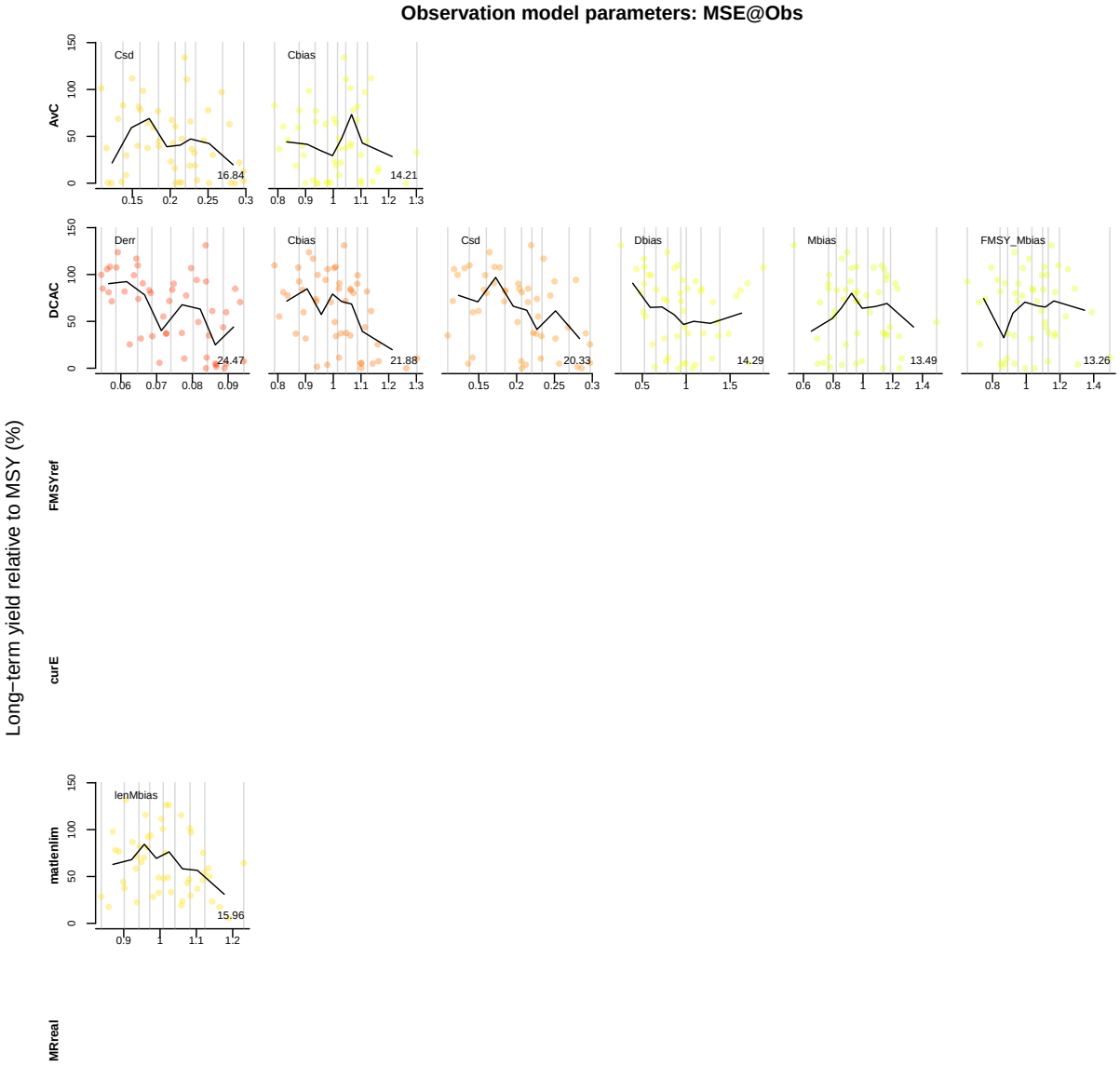
The Value of Information (VOI) functions have been designed to explore the sensitivity of the performance of the Management Procedures to variability in the observation processes and operating model parameters.

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(MSE)
```


Operating model parameters: MSE@OM
Long−term yield relative to MSY (%)
AvC
M
Frac_area_1
AC
ageM
L5
qinc
18.72
17.76
17.74
17.61
17.08
16.91
DCAC
LFS
qinc
FMSY_M
procsd
SSB0
hs
21.1
20.45
20.34
19.26
18.53
18.47
FMSYref
B0
hs
SSB0
BMSY
qcv
K
9.34
8.95
8.74
8.58
7.46
7.41
curE
FMSY
ageM
FMSY_M
qinc
BMSY
Vmaxlen
15.05
14.3
14.03
13.97
13.92
13.63
matlenlim
MGT
Vmaxlen
qinc
Esd
Mgrad
Msd
23.83
20.71
20.62
19.92
19.13
19.03
MRreal
qinc
FMSY
qcv
FMSY_M
BMSY
ageM
12.06
11.96
11.78
11.76
11.7
11.65

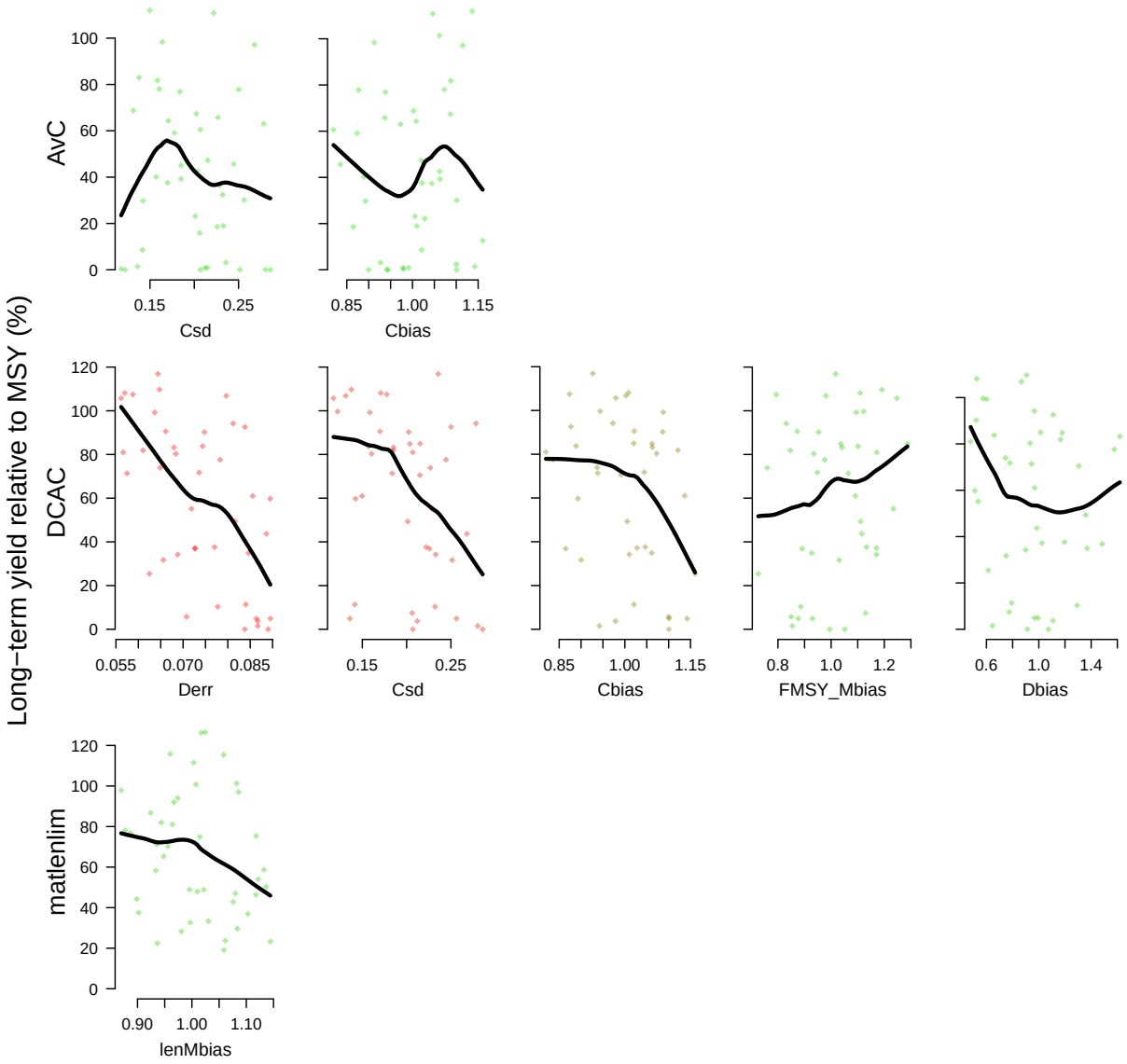



```
## [[1]]
##           MP     1           2      3      4     5       6
## 1        AvC     M Frac_area_1     AC   ageM    L5    qinc
## 2          18.72       17.76  17.74  17.61 17.08   16.91
## 3       DCAC   LFS        qinc FMSY_M procsd  SSB0      hs
## 4          21.1        20.45  20.34  19.26 18.53   18.47
## 5    FMSYref    B0          hs   SSB0   BMSY   qcv       K
## 6          9.34        8.95   8.74   8.58  7.46    7.41
## 7       curE  FMSY        ageM FMSY_M   qinc  BMSY Vmaxlen
## 8          15.05        14.3  14.03  13.97 13.92   13.63
## 9  matlenlim   MGT     Vmaxlen   qinc    Esd Mgrad     Msd
## 10         23.83       20.71  20.62  19.92 19.13   19.03
## 11    MRreal  qinc        FMSY    qcv FMSY_M  BMSY    ageM
## 12         12.06       11.96  11.78  11.76  11.7   11.65
```

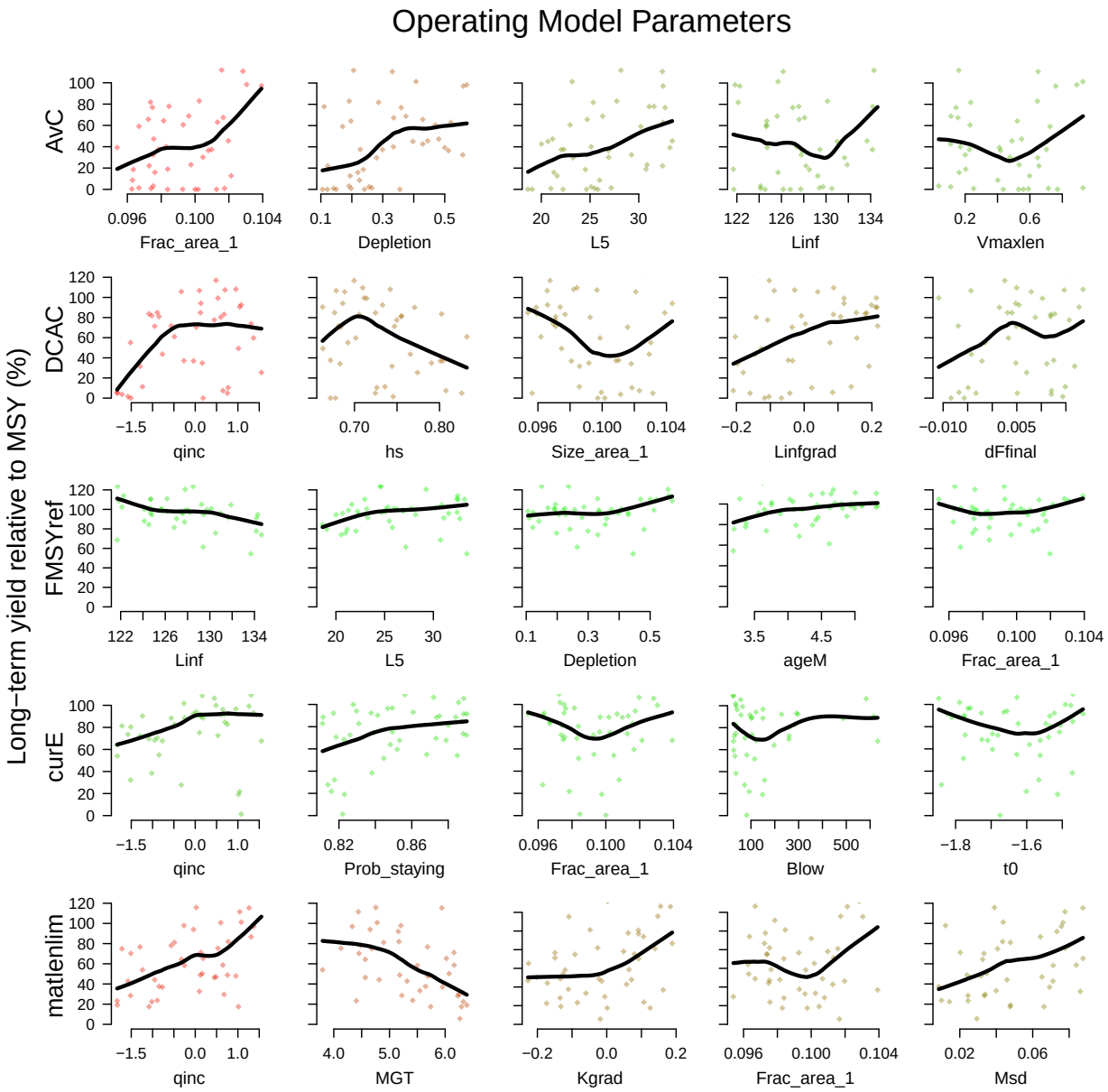
```
##
## [[2]]
##           MP        1     2     3     4     5          6
## 1        AvC      Csd Cbias
## 2               16.84 14.21
## 3       DCAC     Derr Cbias   Csd Dbias Mbias FMSY_Mbias
## 4               24.47 21.88 20.33 14.29 13.49      13.26
## 5    FMSYref     <NA>
## 6                <NA>
## 7       curE     <NA>
## 8                <NA>
## 9  matlenlim lenMbias
## 10              15.96
## 11    MRreal     <NA>
## 12               <NA>
```

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

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




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

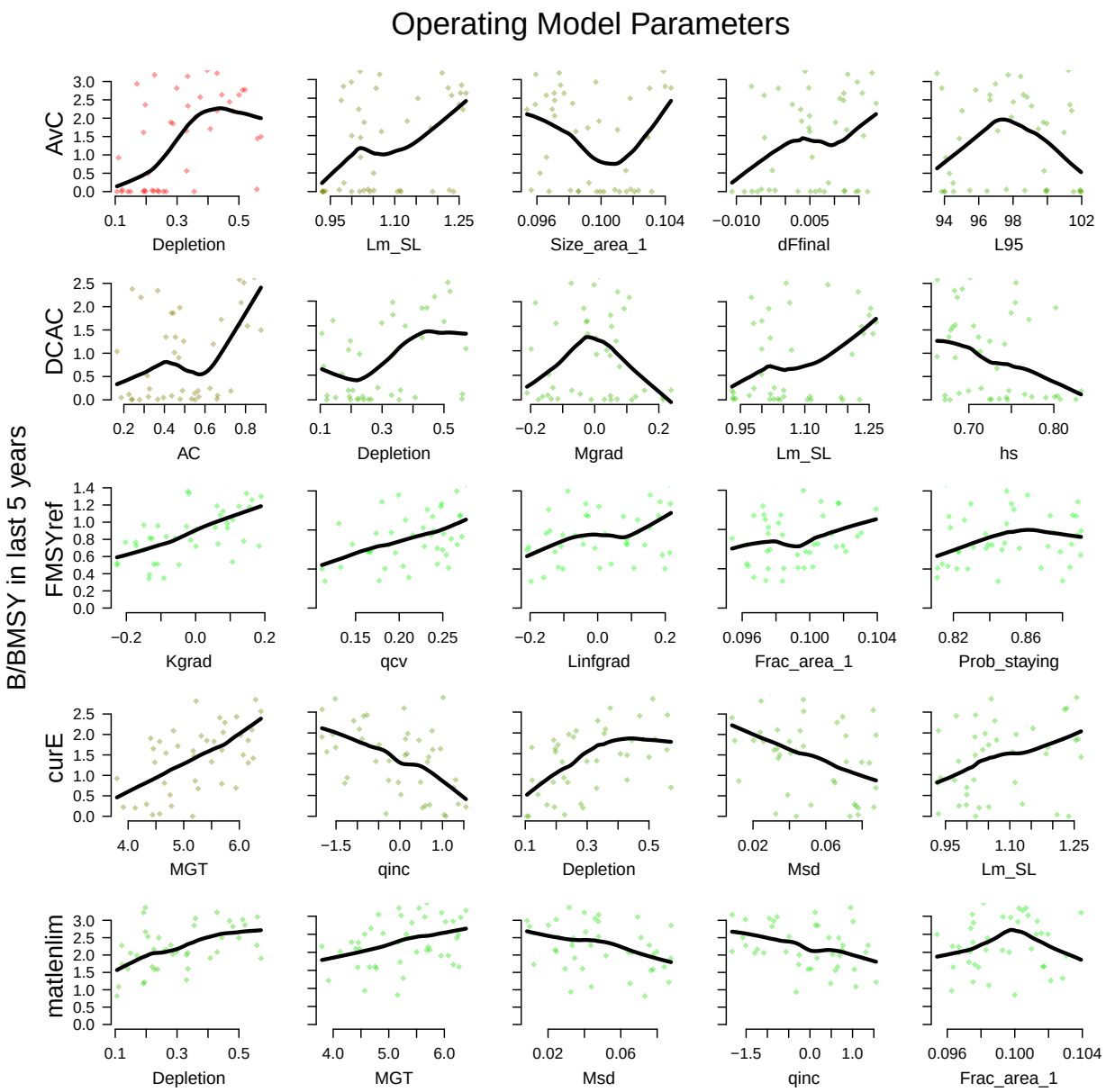


By default, the `VOIplot` function only shows the four Management Procedures with the greatest sensitivity. Here we've made it show all five methods using the `nMP` argument.

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:

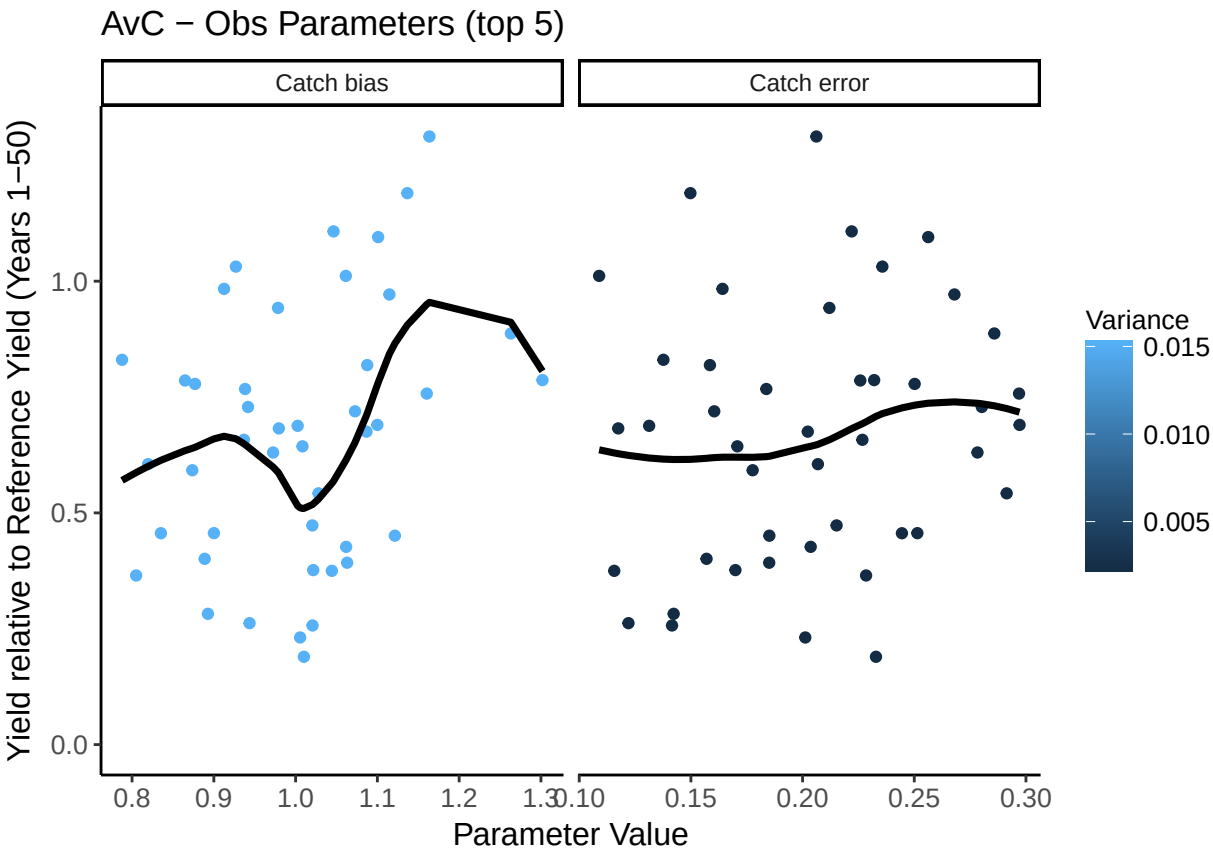
```
VOIplot(MSE, Par="OM", nMP=5, YVar="B")
```



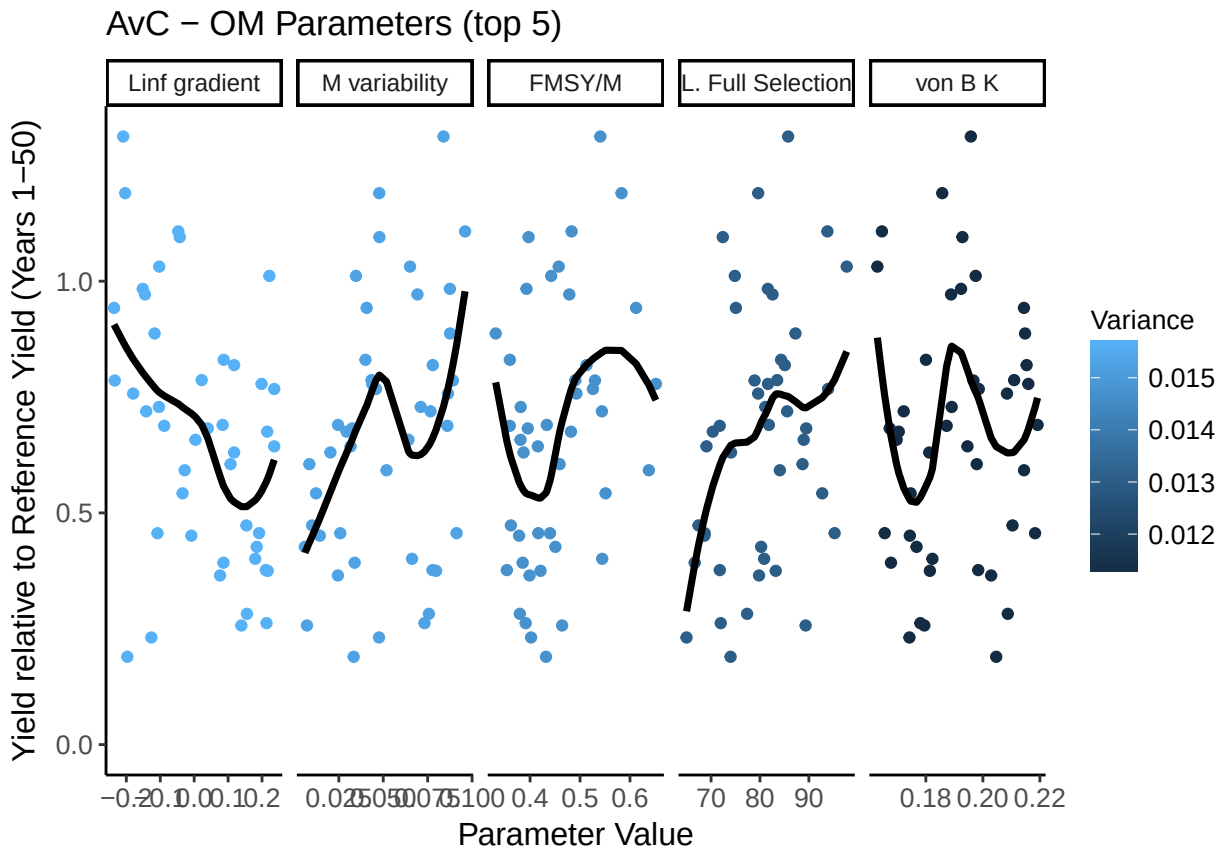
This result shows, perhaps unsurprisingly, that the final biomass is often strongly sensitive to the initial depletion, particularly for the DCAC and matlenlim methods.

The `VOIplot2` function is an updated version of `VOIplot` that uses the `PM` functions and plots the Value of Information for a single MP:

```
VOIplot2(MSE)
```

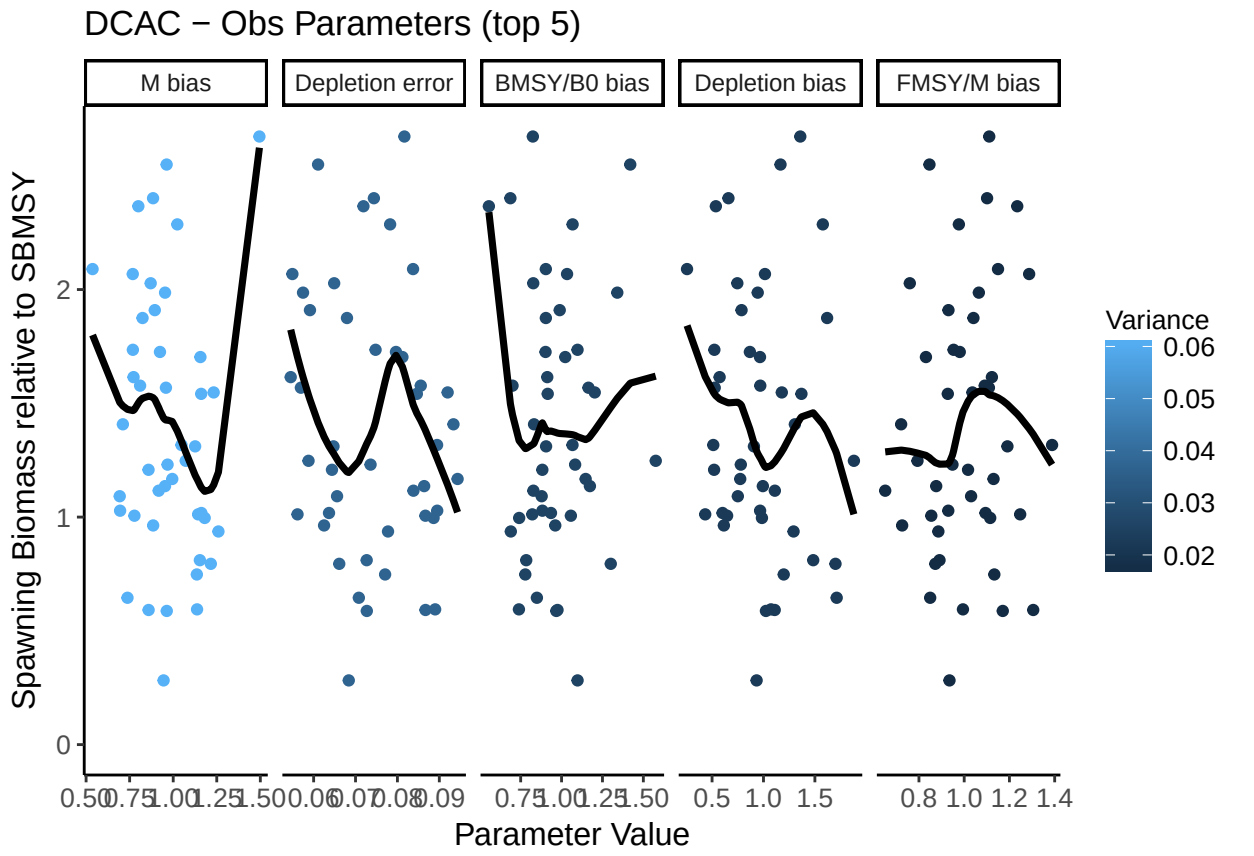



```
VOIplot2(MSE, type="OM")
```

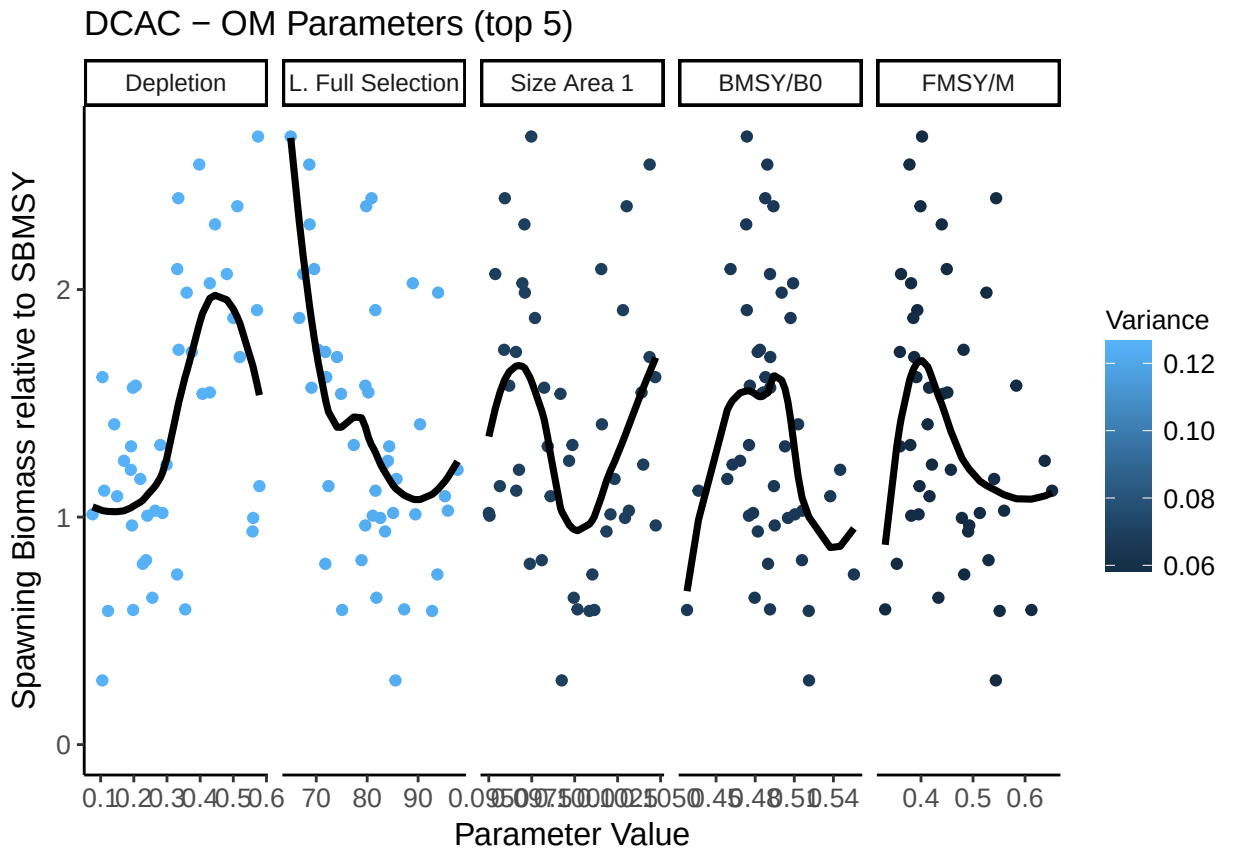


and with a different MP and performance metric:

```
VOIplot2(MSE, MP="DCAC", PM="P100")
```

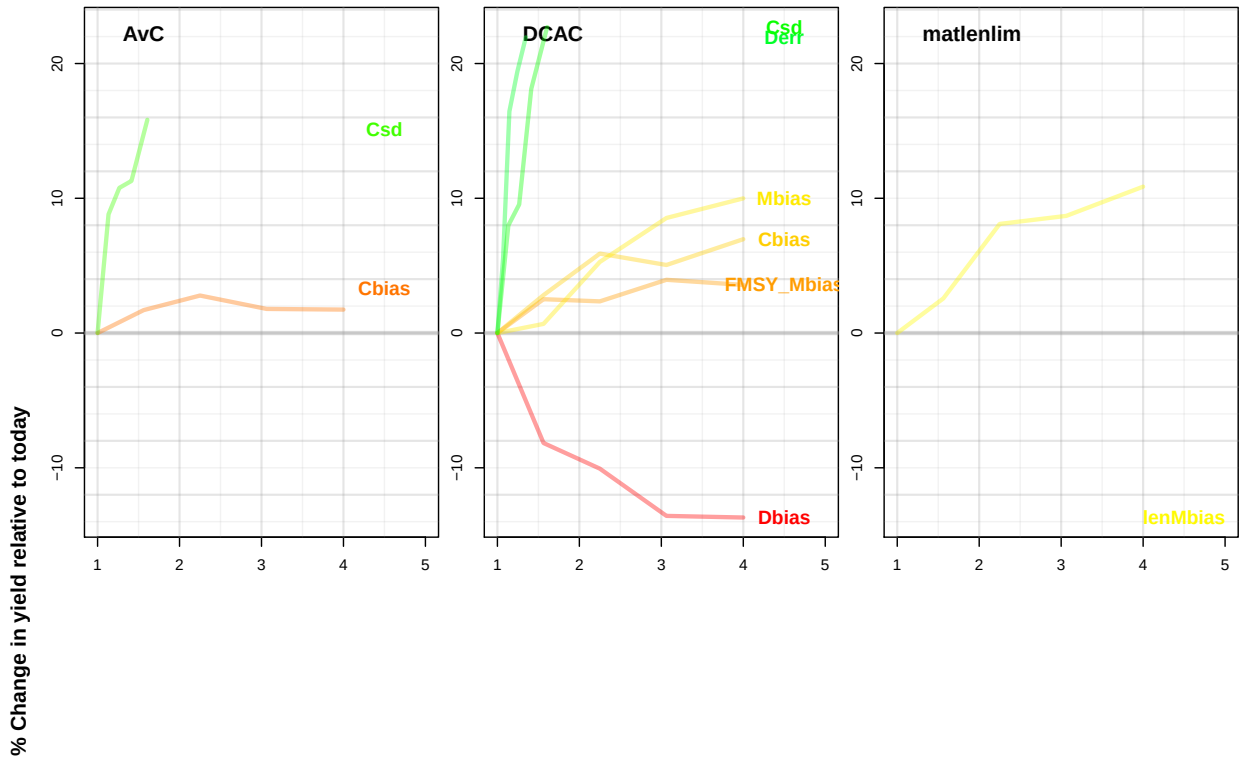



```
VOIplot2(MSE, MP="DCAC", type="OM", PM="P100")
```



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(MSE)
```



`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 17

The Fishery Data Object

`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.

17.1 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, plot = FALSE)
## NULL
```

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

17.2 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 18

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 19

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.

19.1 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.

19.2 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.

19.3 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:/Users/User/Documents/R/win-library/3.5/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.

19.4 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 20

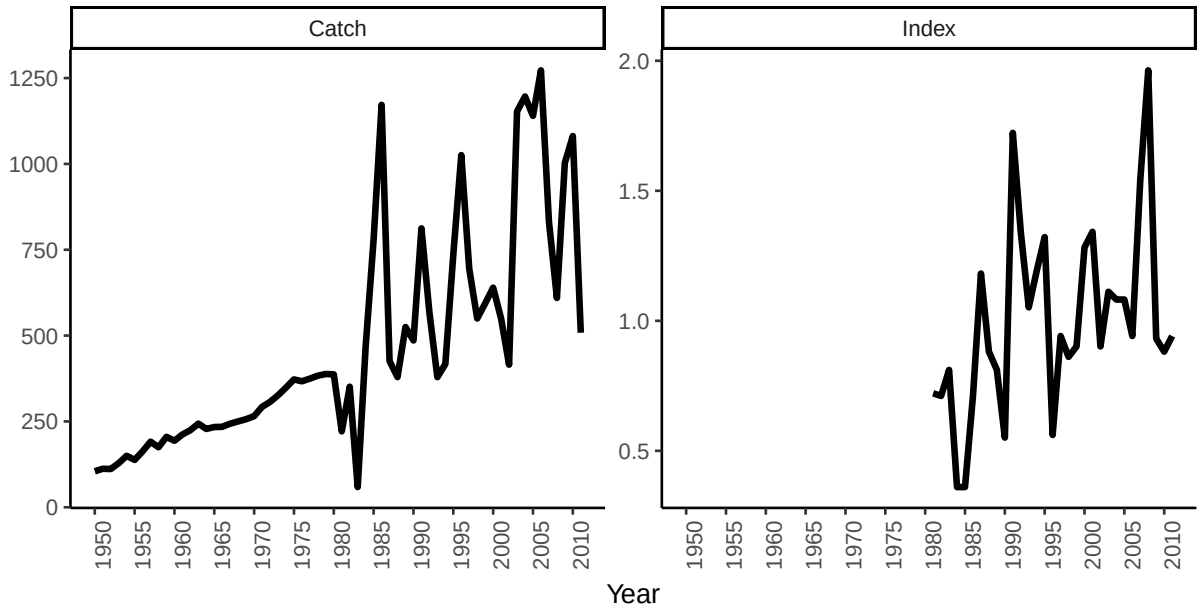
Plotting Data Objects

A generic summary function is available to visualize the data in a `Data` object. By default the `summary` function waits for user input before displaying the next plot, this option can be switched off using `wait=FALSE`:

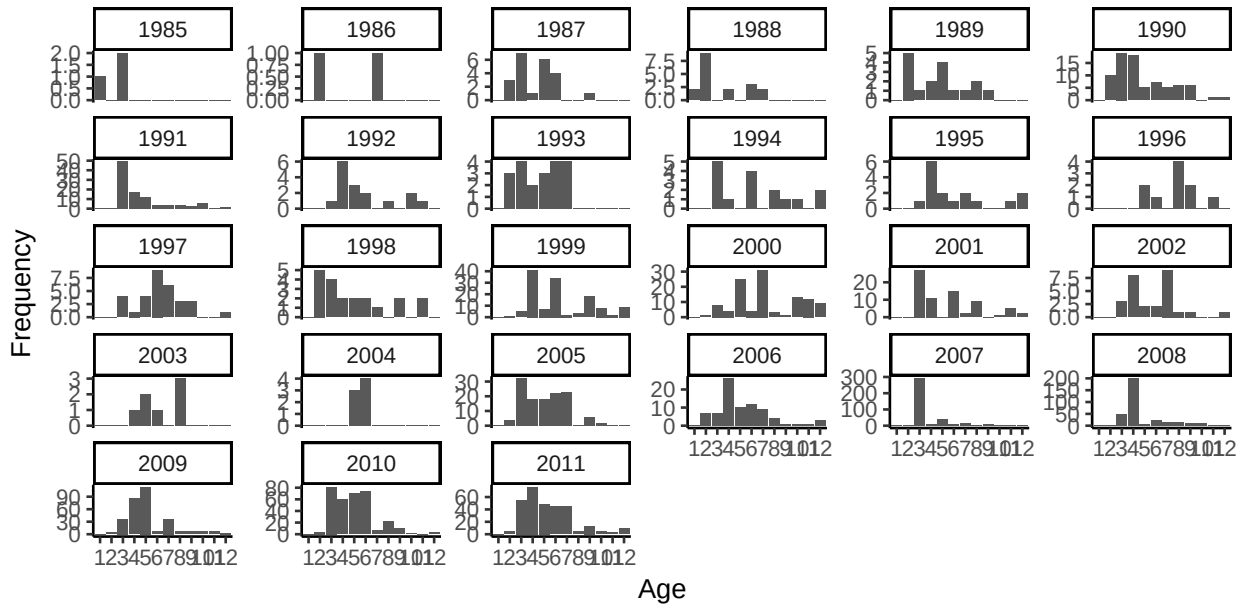
```
summary(Cobia, wait=FALSE)
```

```
## Plotting Time-Series
```

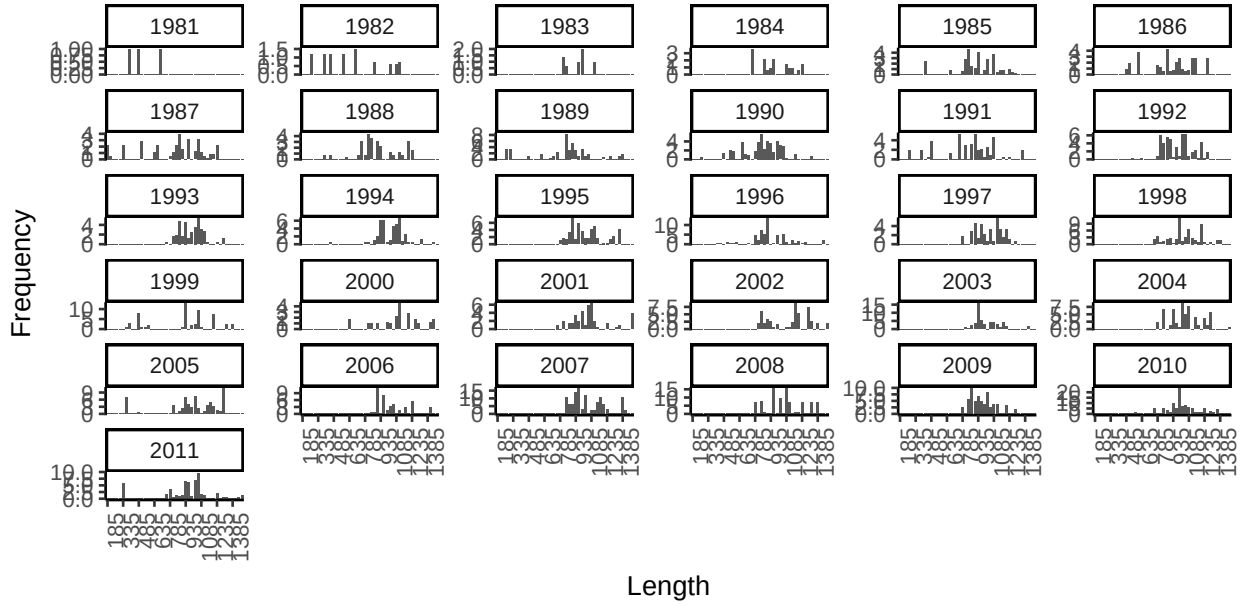
```
## Plotting Catch-at-Age
```

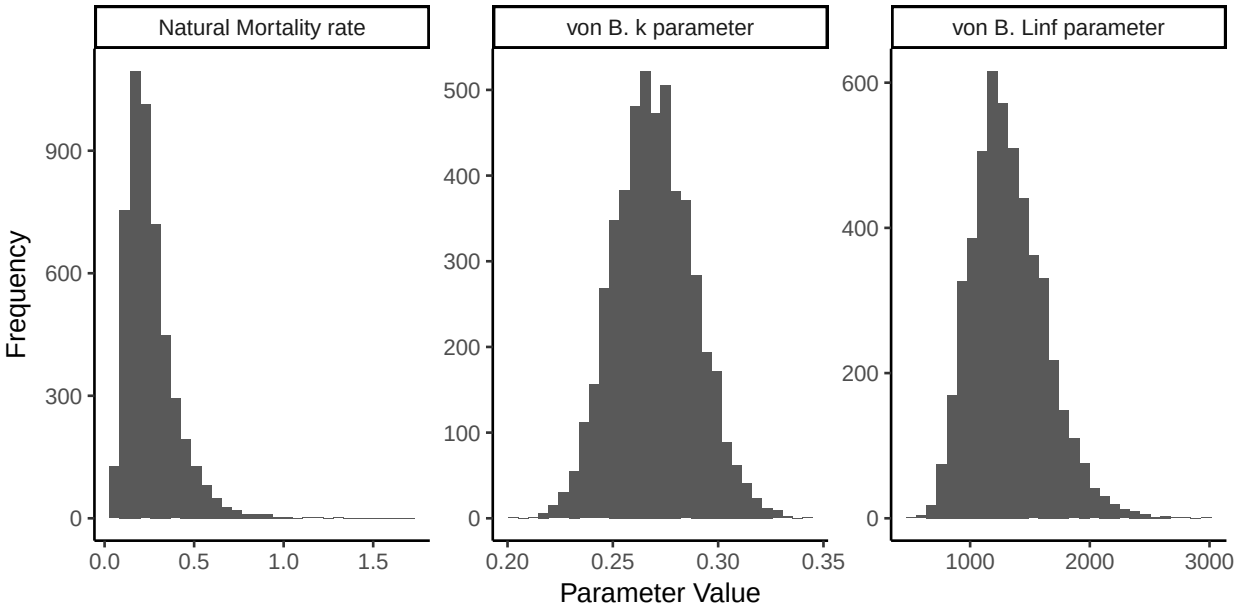



```
## Plotting Catch-at-Length
```



Plotting Parameter Distributions




Natural Mortality rate
von B. k parameter
von B. Linf parameter
Frequency
Parameter Value
0
300
600
900
0.0
0.5
1.0
1.5
100
200
300
400
500
0.20
0.25
0.30
0.35
200
400
600
1000
2000
3000

Chapter 21

Determining Feasible and Available Management Procedures

Although all management procedures can be tested in the simulation, it is often the case that not all MPs can actually be applied in a fishery. This can happen for two reasons:

1. Insufficient data exists to use an MP, for example, a MP may use catch-at-age data which does not exist for the fishery
2. Management constraints such as issues with enforcement or legal requirements may mean some management options are not possible.

Management procedures which can be applied given the current fishery data are referred to as *Available*.

Management procedures that return management recommendations that are, at least in theory, applicable to the fishery are referred to as *Feasible*.

DLMtool has functions to identify MPs that are *Available* and *Feasible*, and also provides information for what additional data are required to allow *Not Available* MPs to be used.

21.1 Feasible MPs

The `Fease` function can be used to determine which MPs are *Feasible*.

For example, if only TAC management is feasible in a fishery, the feasible MPs are:

```
Fease(TAC=TRUE, TAE=FALSE, SL=FALSE, Spatial=FALSE)
```

```
## Feasible management:

## TAC - total allowable catch

## No Data object provided. Returning feasible MPs

##  [1] "AvC"         "avgMP"       "BK"          "BK_CC"       "BK_ML"
##  [6] "CC1"         "CC2"         "CC3"         "CC4"         "CC5"
## [11] "CompSRA"     "CompSRA4010" "DAAC"        "DBSRA"       "DBSRA_40"
## [16] "DBSRA4010"   "DCAC"        "DCAC_40"     "DCAC_ML"     "DCAC4010"
## [21] "DCACs"       "DD"          "DD4010"      "DepF"        "DynF"
## [26] "Fadapt"      "Fdem"        "Fdem_CC"     "Fdem_ML"     "FMSYref"
## [31] "FMSYref50"   "FMSYref75"   "Fratio"      "Fratio_CC"   "Fratio_ML"
## [36] "Fratio4010"  "GB_CC"       "GB_slope"    "GB_target"   "Gcontrol"
```

```
## [41] "HDAAC"       "ICI"         "ICI2"        "Iratio"      "Islope1"
## [46] "Islope2"     "Islope4"     "IT10"        "IT5"         "Itarget1"
## [51] "Itarget2"    "Itarget3"    "Itarget4"    "ITM"         "L95target"
## [56] "Lratio_BHI"  "Lratio_BHI2" "LstepCC1"    "LstepCC2"    "LstepCC3"
## [61] "LstepCC4"    "Ltarget1"    "Ltarget2"    "Ltarget3"    "Ltarget4"
## [66] "MCD"         "MCD4010"     "NFref"       "Rcontrol"    "Rcontrol2"
## [71] "SBT1"        "SBT2"        "SPmod"       "SPMSY"       "SPslope"
## [76] "SPSRA"       "SPSRA_ML"    "TCPUE"
```

We can confirm that all of these MPs are output controls (return a TAC) by using the `MPtype` function:

```
feaseMPs <- Fease(TAC=TRUE, TAE=FALSE, SL=FALSE, Spatial=FALSE)
```

```
## Feasible management:

## TAC - total allowable catch

## No Data object provided. Returning feasible MPs
```

```
MPtype(feaseMPs)
```

```
##             MP      Type Recs
## 1          AvC    Output  TAC
## 2        avgMP    Output  TAC
## 3           BK    Output  TAC
## 4        BK_CC    Output  TAC
## 5        BK_ML    Output  TAC
## 6          CC1    Output  TAC
## 7          CC2    Output  TAC
## 8          CC3    Output  TAC
## 9          CC4    Output  TAC
## 10         CC5    Output  TAC
## 11     CompSRA    Output  TAC
## 12 CompSRA4010    Output  TAC
## 13        DAAC    Output  TAC
## 14       DBSRA    Output  TAC
## 15    DBSRA_40    Output  TAC
## 16   DBSRA4010    Output  TAC
## 17        DCAC    Output  TAC
## 18     DCAC_40    Output  TAC
## 19     DCAC_ML    Output  TAC
## 20    DCAC4010    Output  TAC
## 21       DCACs    Output  TAC
## 22          DD    Output  TAC
## 23      DD4010    Output  TAC
## 24        DepF    Output  TAC
## 25        DynF    Output  TAC
## 26      Fadapt    Output  TAC
## 27        Fdem    Output  TAC
## 28     Fdem_CC    Output  TAC
## 29     Fdem_ML    Output  TAC
## 30      Fratio    Output  TAC
## 31   Fratio_CC    Output  TAC
## 32   Fratio_ML    Output  TAC
## 33  Fratio4010    Output  TAC
## 34       GB_CC    Output  TAC
## 35    GB_slope    Output  TAC
```

```
## 36   GB_target    Output  TAC
## 37    Gcontrol    Output  TAC
## 38       HDAAC    Output  TAC
## 39         ICI    Output  TAC
## 40        ICI2    Output  TAC
## 41      Iratio    Output  TAC
## 42     Islope1    Output  TAC
## 43     Islope2    Output  TAC
## 44     Islope4    Output  TAC
## 45        IT10    Output  TAC
## 46         IT5    Output  TAC
## 47    Itarget1    Output  TAC
## 48    Itarget2    Output  TAC
## 49    Itarget3    Output  TAC
## 50    Itarget4    Output  TAC
## 51         ITM    Output  TAC
## 52   L95target    Output  TAC
## 53  Lratio_BHI    Output  TAC
## 54 Lratio_BHI2    Output  TAC
## 55    LstepCC1    Output  TAC
## 56    LstepCC2    Output  TAC
## 57    LstepCC3    Output  TAC
## 58    LstepCC4    Output  TAC
## 59    Ltarget1    Output  TAC
## 60    Ltarget2    Output  TAC
## 61    Ltarget3    Output  TAC
## 62    Ltarget4    Output  TAC
## 63         MCD    Output  TAC
## 64     MCD4010    Output  TAC
## 65    Rcontrol    Output  TAC
## 66   Rcontrol2    Output  TAC
## 67        SBT1    Output  TAC
## 68        SBT2    Output  TAC
## 69       SPmod    Output  TAC
## 70       SPMSY    Output  TAC
## 71     SPslope    Output  TAC
## 72       SPSRA    Output  TAC
## 73    SPSRA_ML    Output  TAC
## 74       TCPUE    Output  TAC
## 75     FMSYref Reference  TAC
## 76   FMSYref50 Reference  TAC
## 77   FMSYref75 Reference  TAC
## 78       NFref Reference  TAC
```

If a `Data` object is provided to `Fease`, the function will return the names of MPs that are both feasible in terms of management methods, and available in terms of the fishery data:

```
feaseMPs <- Fease(Atlantic_mackerel, TAC=FALSE, TAE=TRUE, SL=TRUE, Spatial=FALSE)
```

```
## Feasible management:

## TAE - total allowable effort

## SL - size selectivity

## Data object provided. Returning feasible and available MPs
```

```
MPtype(feaseMPs)
```

```
##             MP  Type Recs
## 1         curE Input  TAE
## 2       curE75 Input  TAE
## 3          DDe Input  TAE
## 4        DDe75 Input  TAE
## 5         DDes Input  TAE
## 6        DTe40 Input  TAE
## 7        DTe50 Input  TAE
## 8    ItargetE1 Input  TAE
## 9    ItargetE2 Input  TAE
## 10   ItargetE3 Input  TAE
## 11   ItargetE4 Input  TAE
## 12   matlenlim Input   SL
## 13  matlenlim2 Input   SL
## 14 minlenLopt1 Input   SL
## 15     slotlim Input   SL
## 16     TCPUE_e Input  TAE
```

21.2 Available MPs

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] "CC2"          "CC3"          "CC4"          "CC5"
##  [9] "curE"         "curE75"       "DAAC"         "DBSRA"
## [13] "DBSRA_40"     "DBSRA4010"    "DCAC"         "DCAC_40"
## [17] "DCAC4010"     "DCACs"        "DD"           "DD4010"
## [21] "DDe"          "DDe75"        "DDes"         "DepF"
## [25] "DTe40"        "DTe50"        "DynF"         "Fadapt"
## [29] "Fdem"         "Fratio"       "Fratio4010"   "GB_slope"
## [33] "Gcontrol"     "HDAAC"        "ICI"          "ICI2"
## [37] "Iratio"       "Islope1"      "Islope2"      "Islope4"
## [41] "Itarget1"     "Itarget1_MPA" "Itarget2"     "Itarget3"
## [45] "Itarget4"     "ItargetE1"    "ItargetE2"    "ItargetE3"
## [49] "ItargetE4"    "matlenlim"    "matlenlim2"   "MCD"
## [53] "MCD4010"      "minlenLopt1"  "MRnoreal"     "MRreal"
## [57] "NFref"        "Rcontrol"     "Rcontrol2"    "SBT1"
## [61] "slotlim"      "SPmod"        "SPMSY"        "SPslope"
## [65] "SPSRA"        "YPR"          "avgMP"        "TCPUE_e"
## [69] "THC"
```

If all management methods are feasible, the list of MPs returned by the `Can` function will be the same as those returned by the `Fease` function when the `Data` object is provided.

21.3 Unavailable MPs

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]
##  [1,] "BK_CC"
##  [2,] "BK_ML"
##  [3,] "CompSRA"
##  [4,] "CompSRA4010"
##  [5,] "DCAC_ML"
##  [6,] "EtargetLopt"
##  [7,] "Fdem_CC"
##  [8,] "Fdem_ML"
##  [9,] "FMSYref"
## [10,] "FMSYref50"
## [11,] "FMSYref75"
## [12,] "Fratio_CC"
## [13,] "Fratio_ML"
## [14,] "GB_CC"
## [15,] "GB_target"
## [16,] "IT10"
## [17,] "IT5"
## [18,] "ITe10"
## [19,] "ITe5"
## [20,] "ITM"
## [21,] "L95target"
## [22,] "LBSPR"
## [23,] "Lratio_BHI"
## [24,] "Lratio_BHI2"
## [25,] "LstepCC1"
## [26,] "LstepCC2"
## [27,] "LstepCC3"
## [28,] "LstepCC4"
## [29,] "LstepCE1"
## [30,] "LstepCE2"
## [31,] "Ltarget1"
## [32,] "Ltarget2"
## [33,] "Ltarget3"
## [34,] "Ltarget4"
## [35,] "LtargetE1"
## [36,] "LtargetE4"
## [37,] "SBT2"
## [38,] "SPSRA_ML"
## [39,] "YPR_CC"
## [40,] "YPR_ML"
## [41,] "TCPUE"
##       [,2]
##  [1,] "Missing data: CAA"
##  [2,] "Missing data: CAL"
##  [3,] "Missing data: CAA"
##  [4,] "Missing data: CAA"
##  [5,] "Missing data: CAL"
##  [6,] "Missing data: ML"
##  [7,] "Missing data: CAA"
##  [8,] "Missing data: CAL"
##  [9,] "MP returned an error. Check MP function and/or Data object."
```

```
## [10,] "MP returned an error. Check MP function and/or Data object."
## [11,] "MP returned an error. Check MP function and/or Data object."
## [12,] "Missing data: CAA"
## [13,] "Missing data: CAL"
## [14,] "Missing data: Cref"
## [15,] "Missing data: Cref, Iref"
## [16,] "Missing data: Iref, MPrec"
## [17,] "Missing data: Iref, MPrec"
## [18,] "Missing data: Iref"
## [19,] "Missing data: Iref"
## [20,] "Missing data: Iref, MPrec"
## [21,] "Missing data: ML"
## [22,] "Missing data: CAL"
## [23,] "Missing data: CAL"
## [24,] "Missing data: CAL"
## [25,] "Missing data: ML"
## [26,] "Missing data: ML"
## [27,] "Missing data: ML"
## [28,] "Missing data: ML"
## [29,] "Missing data: ML"
## [30,] "Missing data: ML"
## [31,] "Missing data: ML"
## [32,] "Missing data: ML"
## [33,] "Missing data: ML"
## [34,] "Missing data: ML"
## [35,] "Missing data: ML"
## [36,] "Missing data: ML"
## [37,] "Missing data: Rec, Cref"
## [38,] "Missing data: CAL"
## [39,] "Missing data: CAA"
## [40,] "Missing data: CAL"
## [41,] "Missing data: MPrec"
```

Chapter 22

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
```

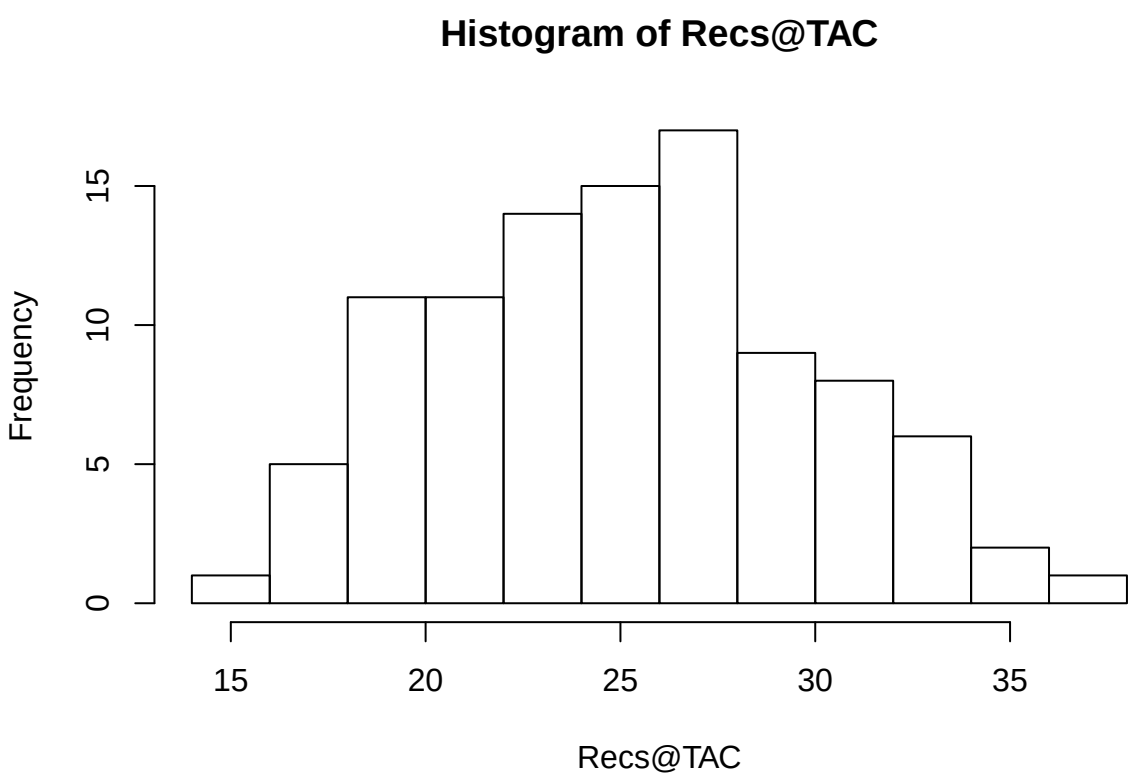
The `runMP` prints out the MP recommendations to the console. In the case of a TAC, where multiple repititions where (see `reps` = 1000 above) used the `runMP` function prints the median TAC recommendation.

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
```

```
## Method Rcontrol produced greater than 50% NA values
```

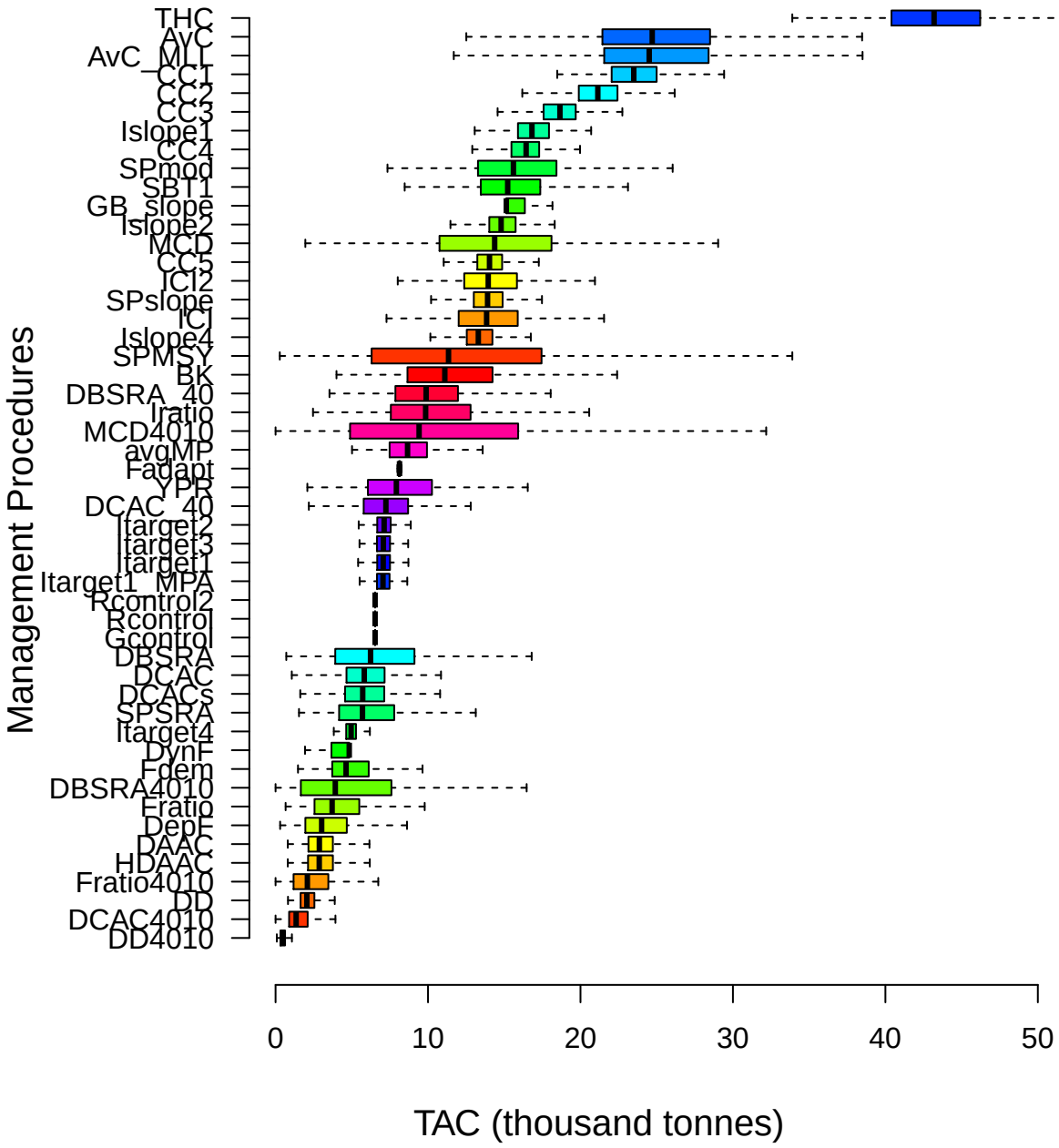
```
##                TAC Effort   LR5    LFR     HS Area 1 Area 2
## AvC          24.70
## AvC_MLL      24.51        90.25  95.00
## BK           11.10
## CC1          23.50
## CC2          21.14
## CC3          18.65
## CC4          16.43
## CC5          14.03
## curE                1.00
## curE75              0.75
## DAAC          2.87
## DBSRA         6.23
```

```
## DBSRA_40      9.87
## DBSRA4010     3.92
## DCAC          5.82
## DCAC_40       7.24
## DCAC4010      1.35
## DCACs         5.72
## DD            2.05
## DD4010        0.45
## DDe                0.09
## DDe75              0.07
## DDes               0.90
## DepF          3.02
## DTe40              0.90
## DTe50              0.90
## DynF          4.83
## Fadapt        8.13
## Fdem          4.62
## Fratio        3.71
## Fratio4010    2.09
## GB_slope     15.14
## Gcontrol      6.53
## HDAAC         2.86
## ICI          13.84
## ICI2         13.95
## Iratio        9.84
## Islope1      16.82
## Islope2      14.80
## Islope4      13.30
## Itarget1      7.06
## Itarget1_MPA  7.05                                  0      1
## Itarget2      7.12
## Itarget3      7.07
## Itarget4      4.96
## ItargetE1          0.85
## ItargetE2          0.85
## ItargetE3          0.85
## ItargetE4          0.85
## matlenlim               90.25  95.00
## matlenlim2              99.28 104.50
## MCD          14.37
## MCD4010       9.41
## minlenLopt1             89.75  99.72
## MRnoreal                                          0      1
## MRreal                                            0      1
## NFref         0.01
## Rcontrol      6.53
## Rcontrol2     6.53
## SBT1         15.22
## slotlim                 99.28 104.50 142.55
## SPmod        15.60
## SPMSY        11.35
## SPslope      13.91
## SPSRA         5.70
## YPR           7.92
```

```
## avgMP          8.66
## TCPUE_e                0.95
## THC           43.19
```

The TAC recommendations from each Output control can be plotted:

```
boxplot(Atlantic_mackerel)
```



Advanced DLMtool

Chapter 23

Averaging MPs

In some cases users may wish to provide management advice by averaging the recommendations from several different well-performing management procedures. This of course is a new management procedure in itself, and should be tested in MSE before being adopted for use in a fishery.

The `makeMeanMP` function can be used to create a new MP that averages the results of several MPs.

For example, suppose we wished to develop an MP that averages the results of 4 output control MPs: `BK`, `DBSRA`, `Fadapt` and `Rcontrol`.

This can be achieved by the following:

```
avgMP <- makeMeanMP(c("BK", "DBSRA", "Fadapt", "Rcontrol"))
class(avgMP)
```

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

And now we can test our new MP in MSE. We will run a decent run of simulations so the results are stable:

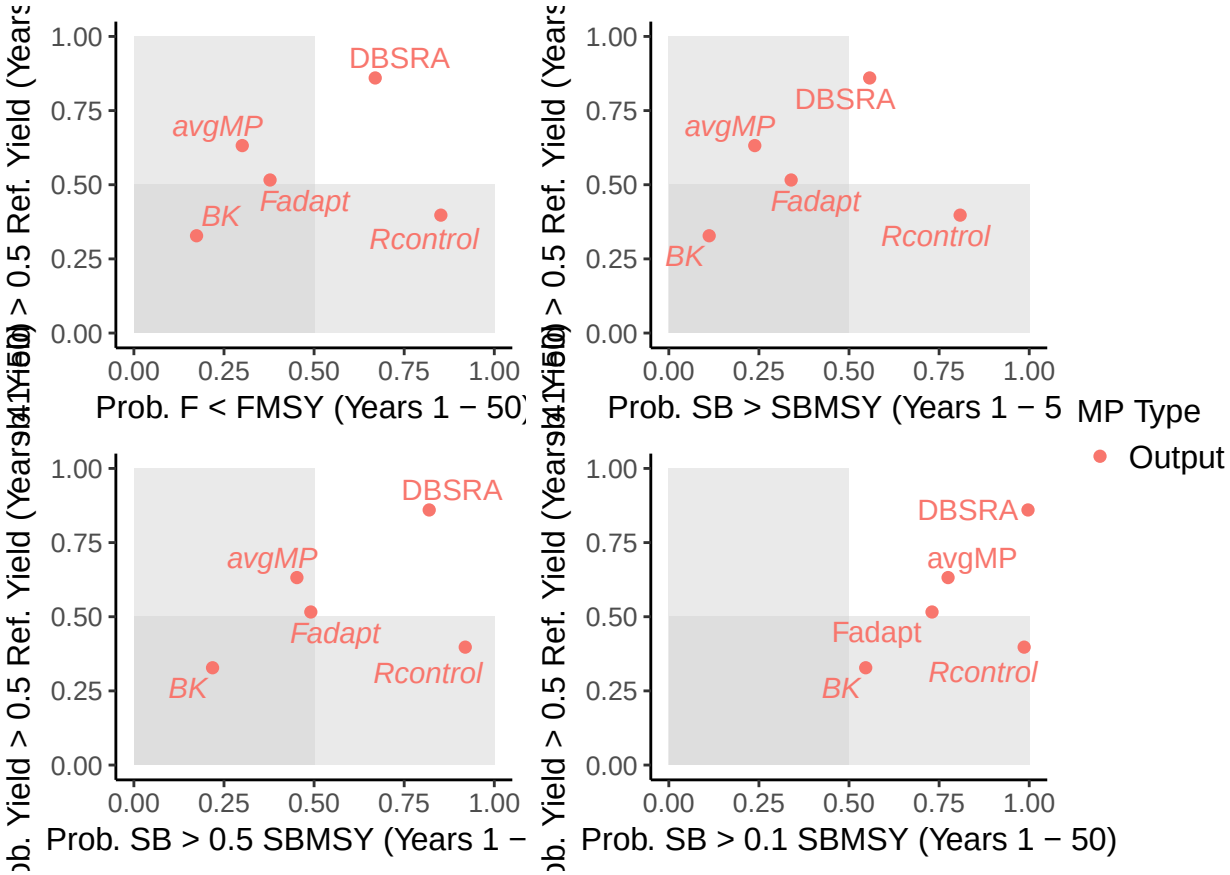
```
OM <- DLMtool::testOM
OM@nsim <- 200
MSE <- runMSE(OM, MPs=c("avgMP", "BK", "DBSRA", "Fadapt", "Rcontrol"), parallel = TRUE)
```

```
## Running MSE in parallel on 10 processors
```

```
## MSE completed
```

How did our newly created averaging MP perform?

```
Tplot(MSE)
```




```
##          MP PNOF  LTY P100  P50  P10 Satisificed
## 1    avgMP 0.30 0.63 0.24 0.45 0.77       FALSE
## 2       BK 0.17 0.33 0.11 0.22 0.55       FALSE
## 3    DBSRA 0.67 0.86 0.56 0.82 1.00        TRUE
## 4   Fadapt 0.38 0.52 0.34 0.49 0.73       FALSE
## 5 Rcontrol 0.85 0.40 0.81 0.92 0.99       FALSE
```

More information on creating your own MPs can be found in the Developing Custom Management Procedures chapter.

Chapter 24

Evaluating OM

The `Turing` function has been designed to evaluate an Operating Model against a Data object from the same fishery. The function generates 5 random samples of Data from the `OM` object and plots these together with the corresponding data in the `Data` object.

Ideally, in a well conditioned OM one should not be able to visually detect which of the plots are the real data and which have been artifically generated by the operating model.

The `Turing` function takes an object of class `OM` and an object of class `Data`. It first plots the simulated and real data and then waits for user input before revelaing which of the plots are the real data from the `Data` object.

We use the `wait=FALSE` argument here so that each plot is printed without waiting for user input.

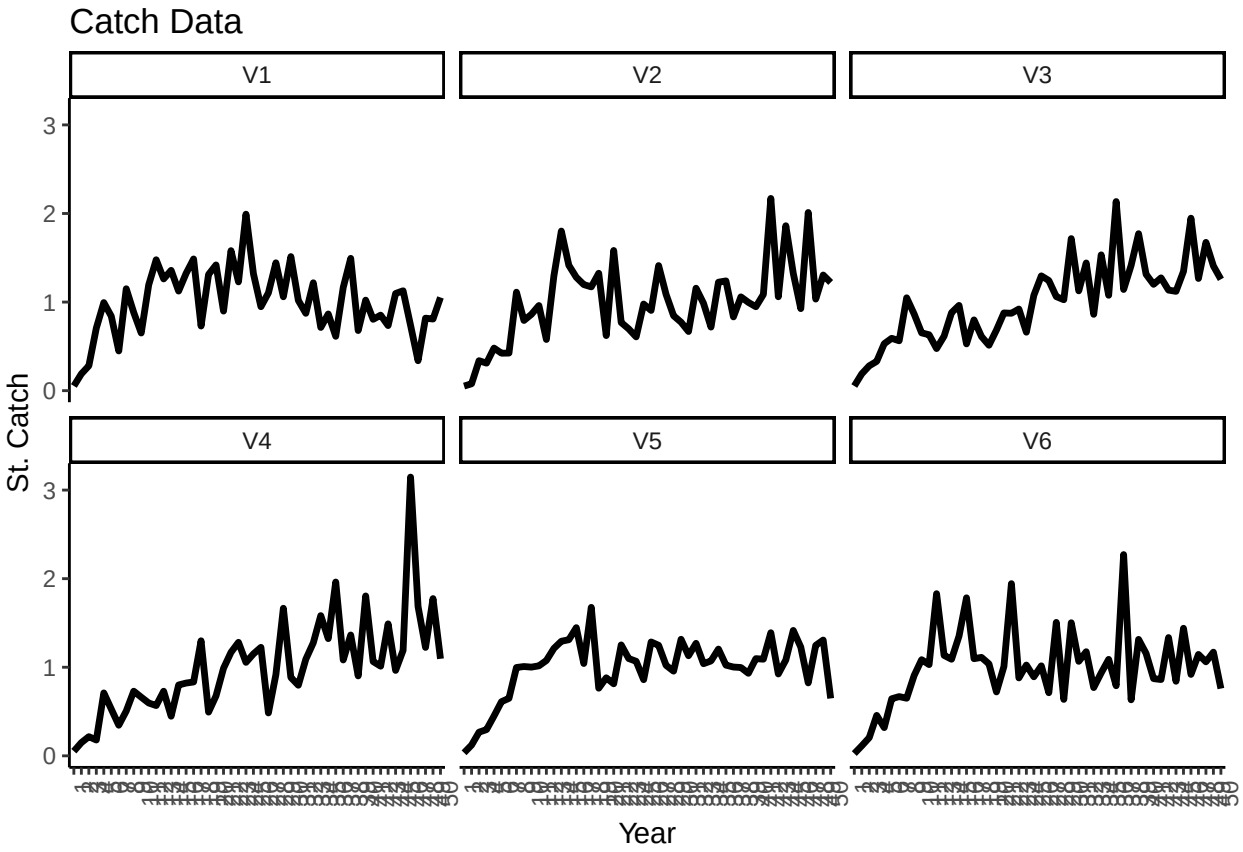
In this example we are using a `Data` object that was simulated using the same `OM`, so it shouldn't be suprising that it is difficult to detect which of the plots are from the `Data` object:

```
Turing(DLMtool::testOM, DLMtool::SimulatedData, wait=FALSE)
```

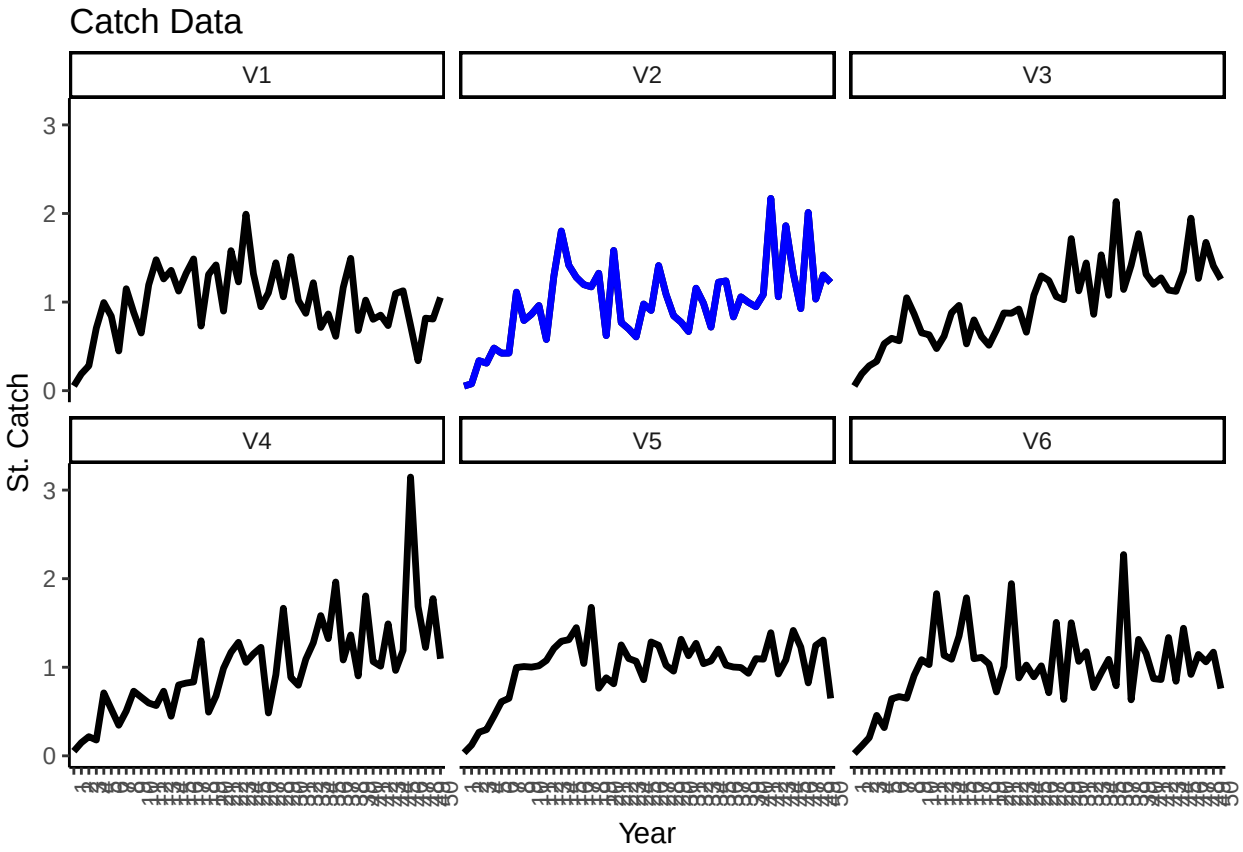
```
## Simulating Data
```

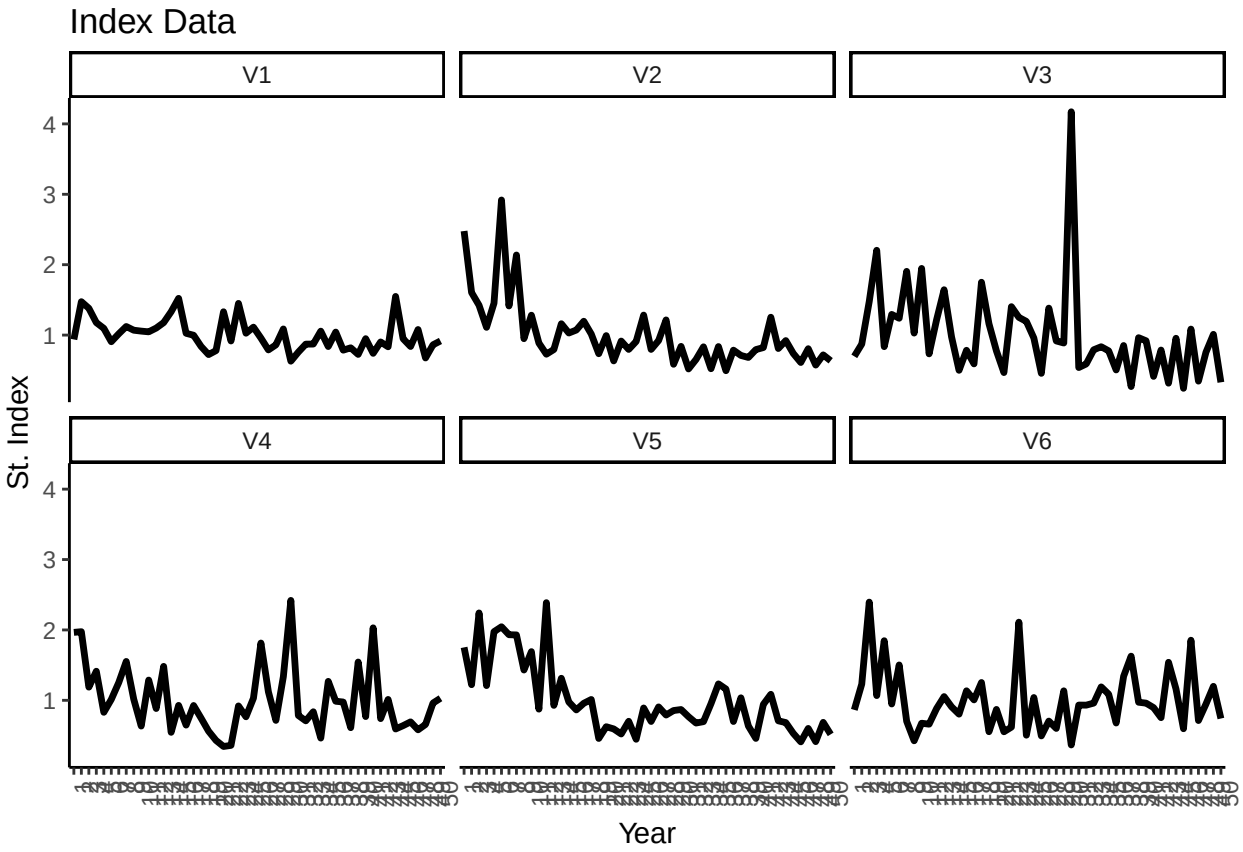
```
## Randomly sampling 5 iterations
```

```
## Plotting: Catch Data
```

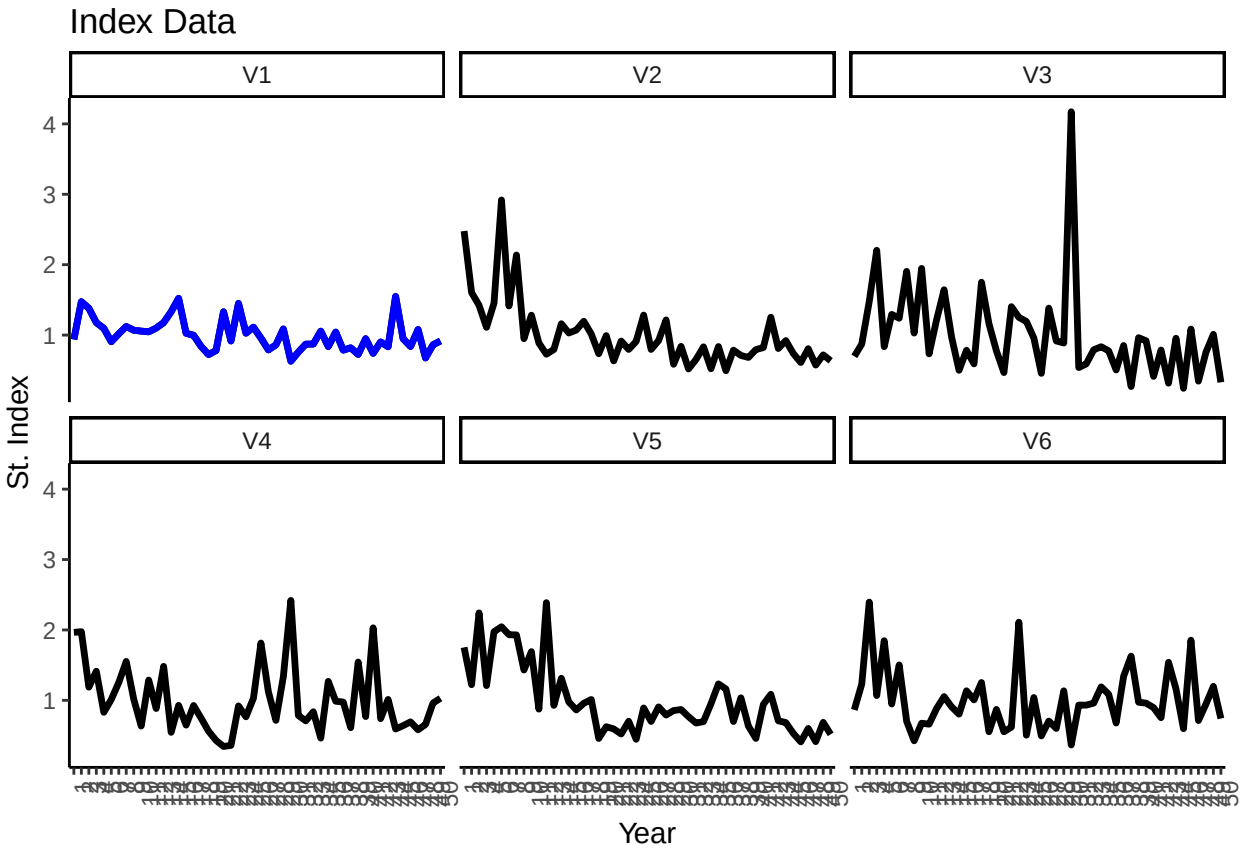


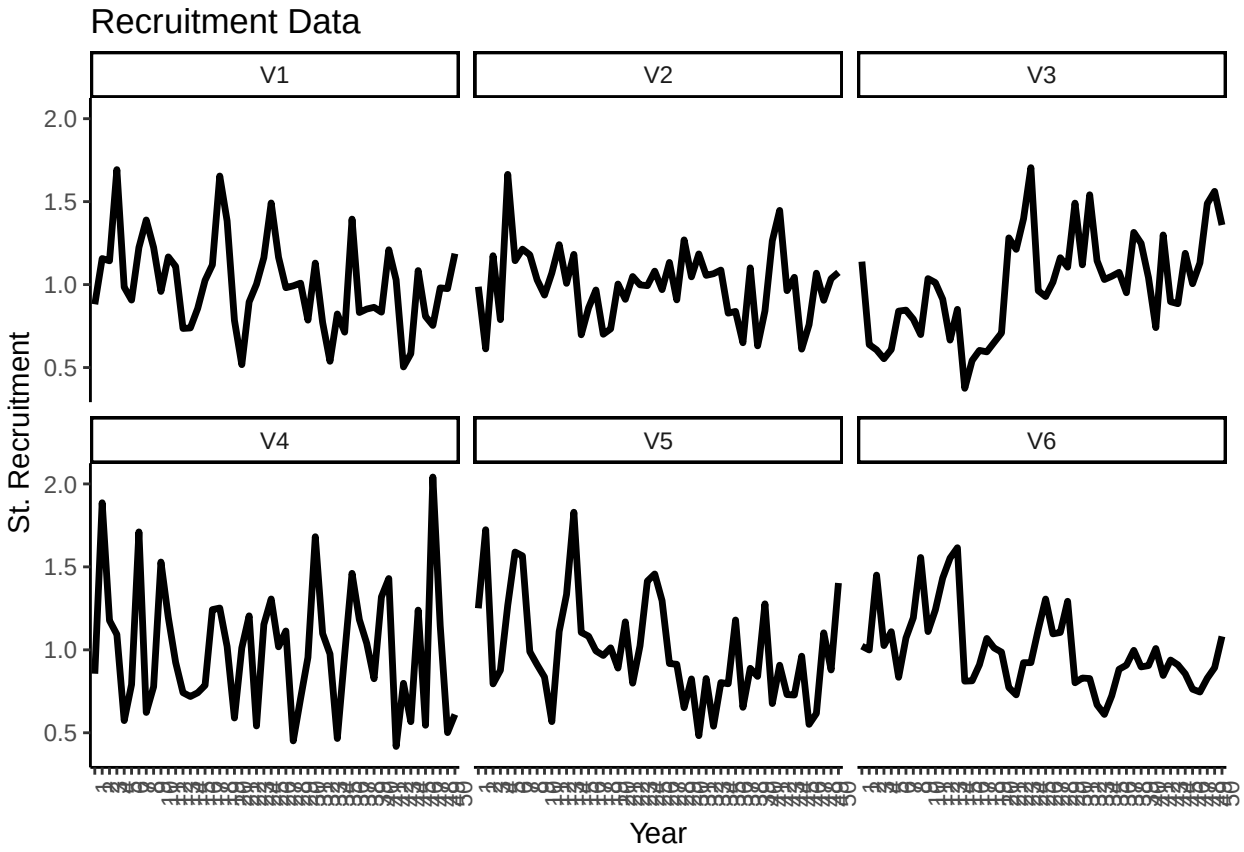

```
## Plotting: Index Data
```


Catch Data
V1
V2
V3
V4
V5
V6
St. Catch
Year
0
1
2
3

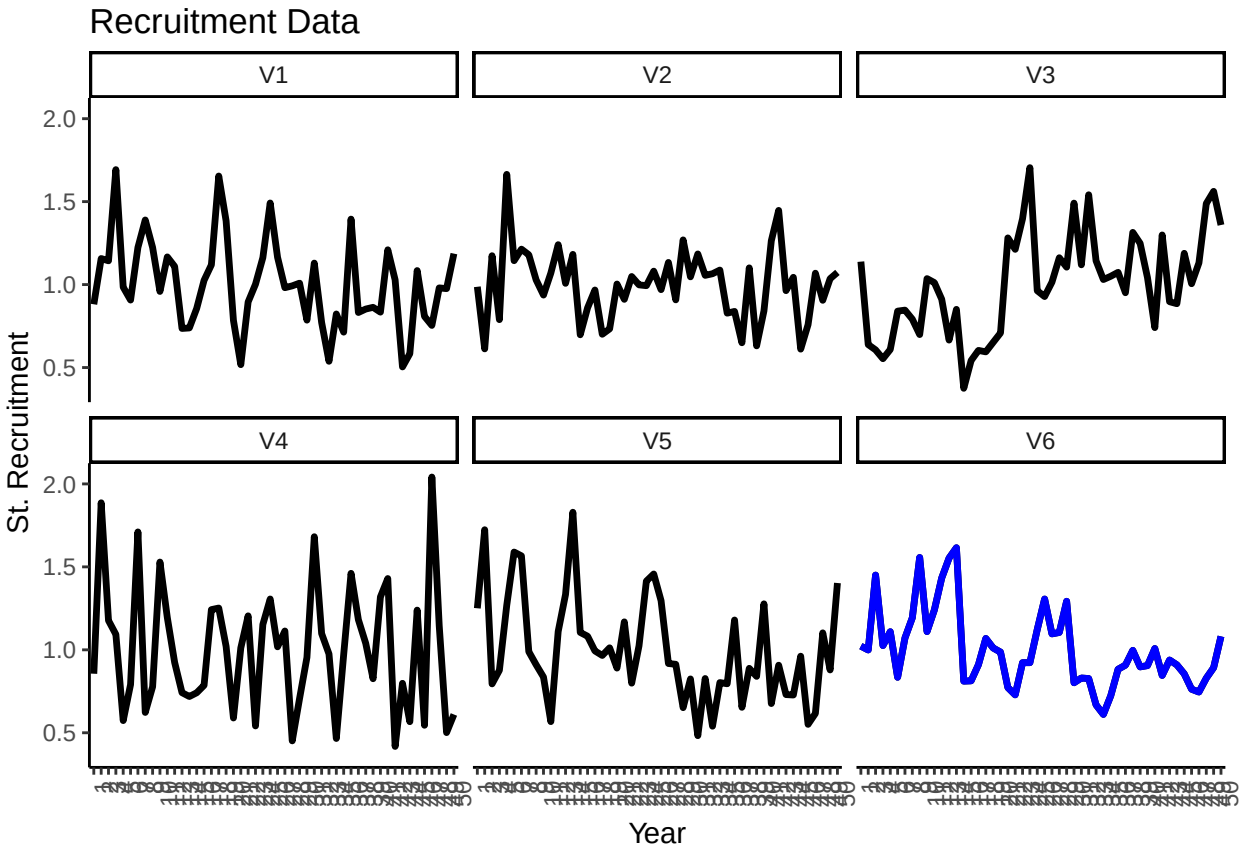


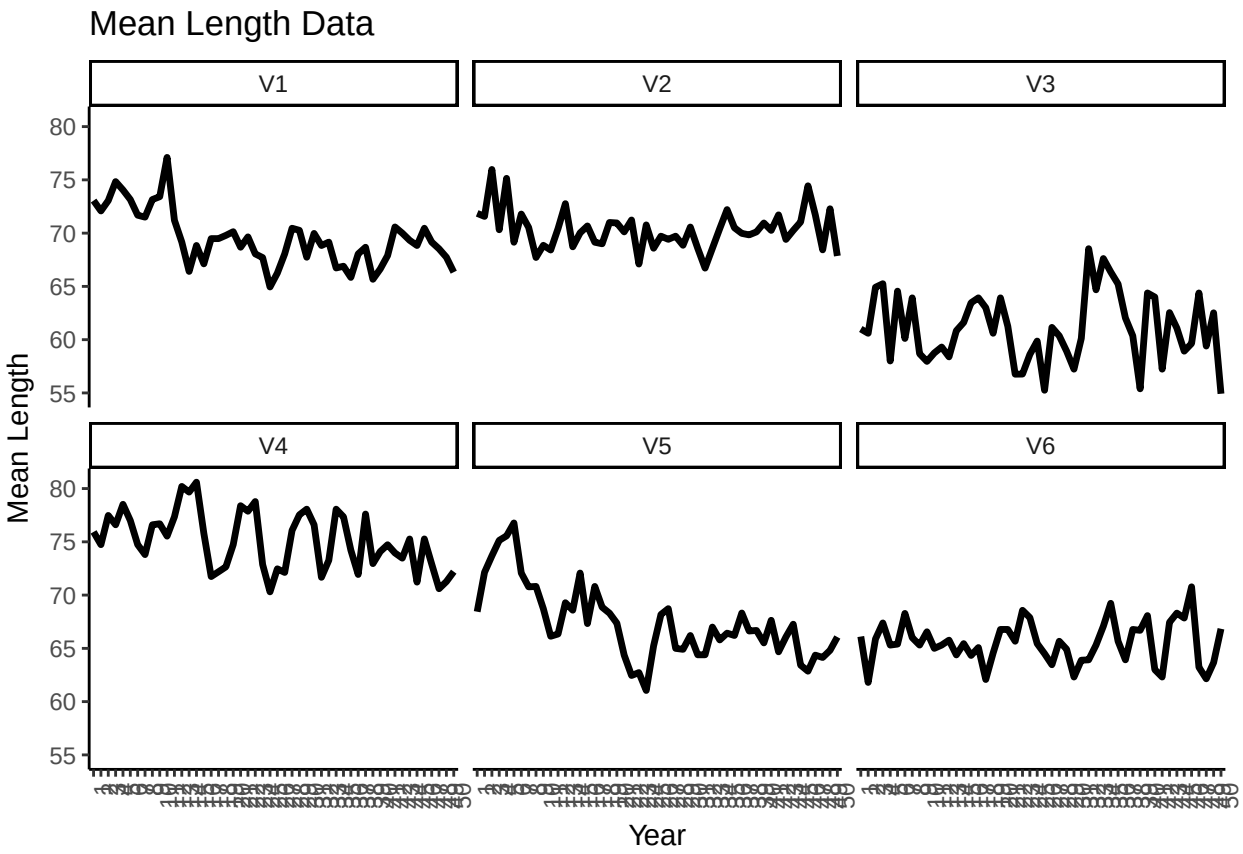

```
## Plotting: Recruitment Data
```


Index Data
V1
V2
V3
V4
V5
V6
St. Index
Year

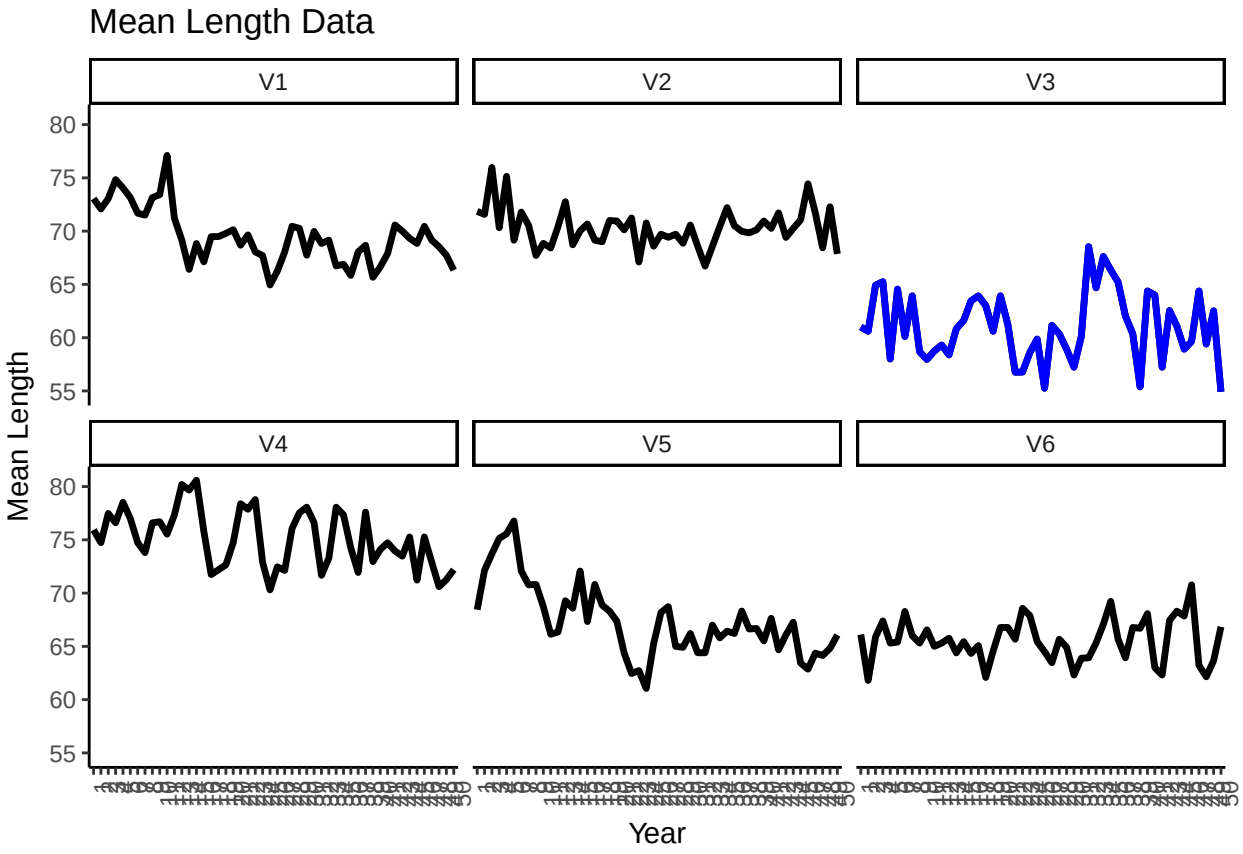


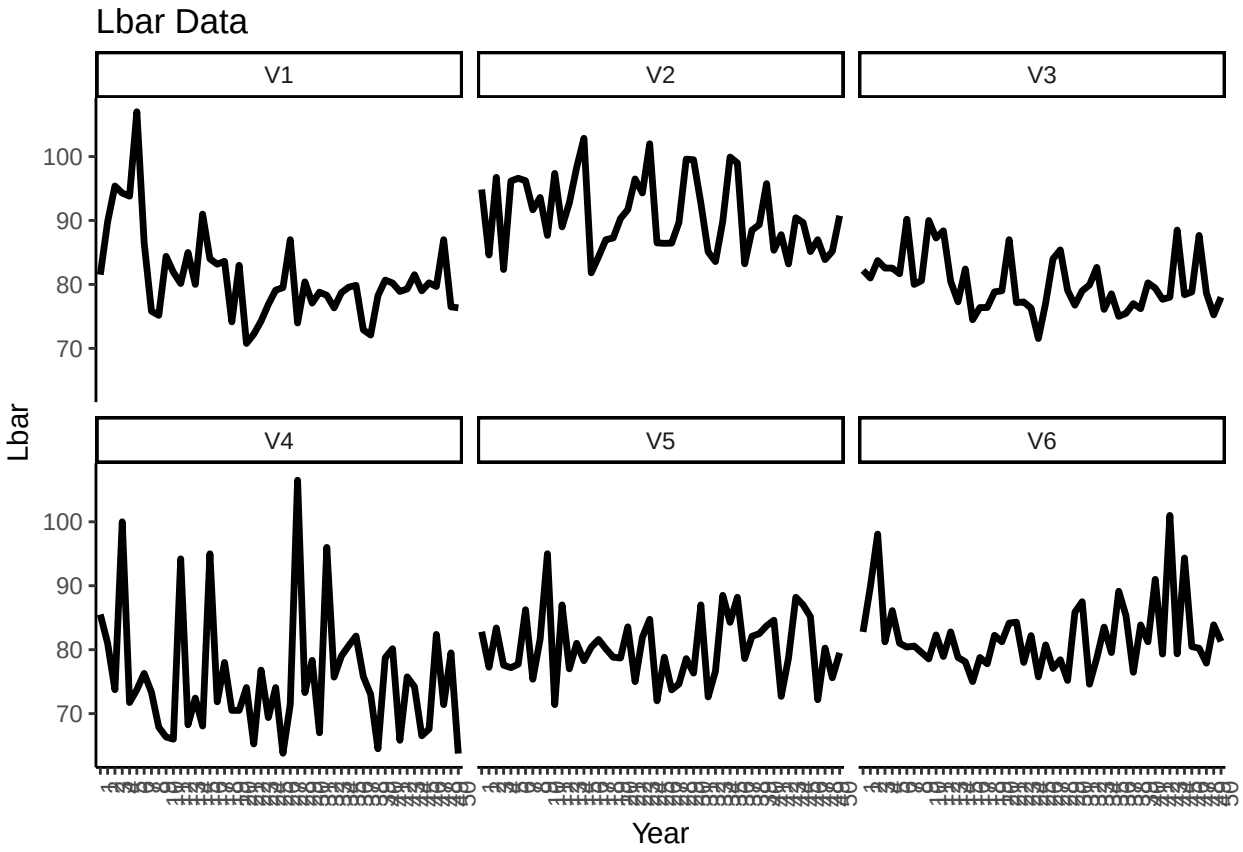

```
## Plotting: Mean Length Data
```


Recruitment Data
V1
V2
V3
V4
V5
V6
St. Recruitment
2.0
1.5
1.0
0.5
Year

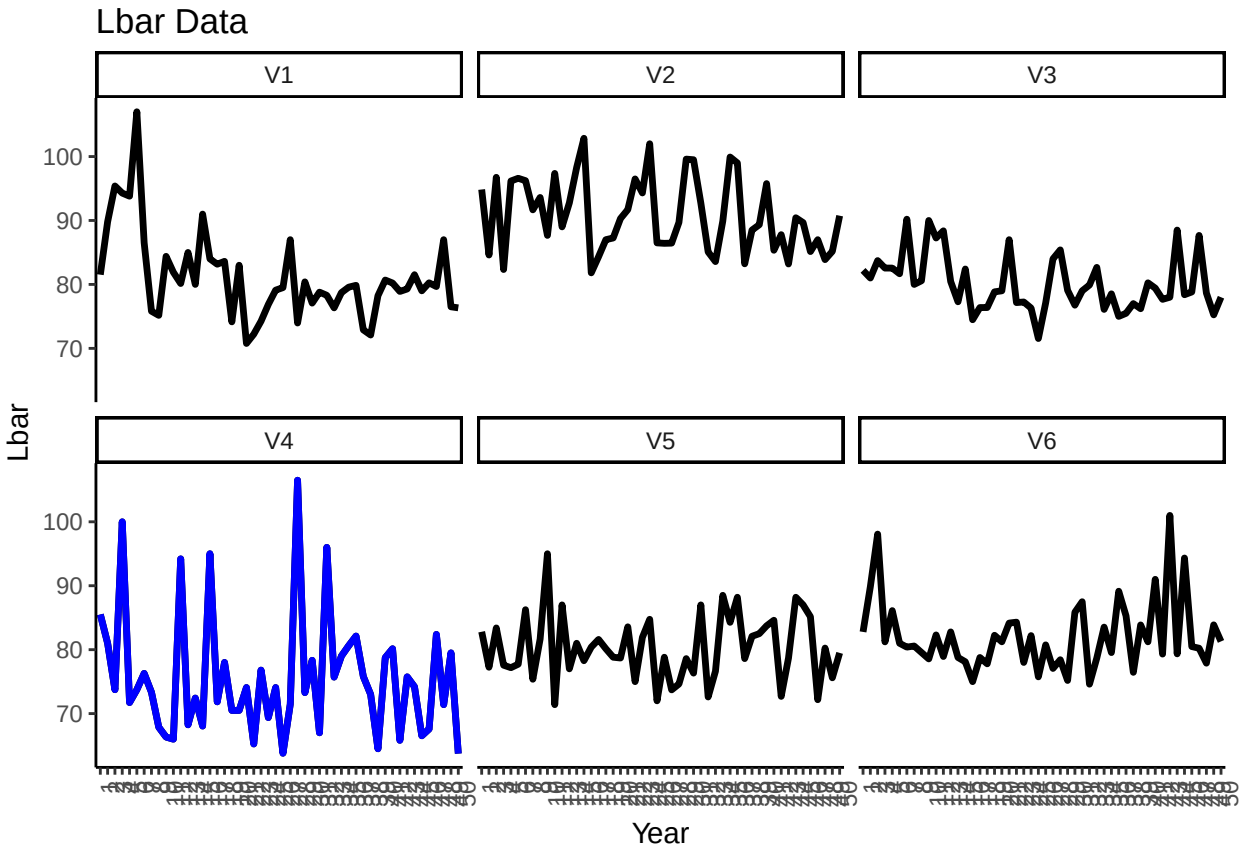


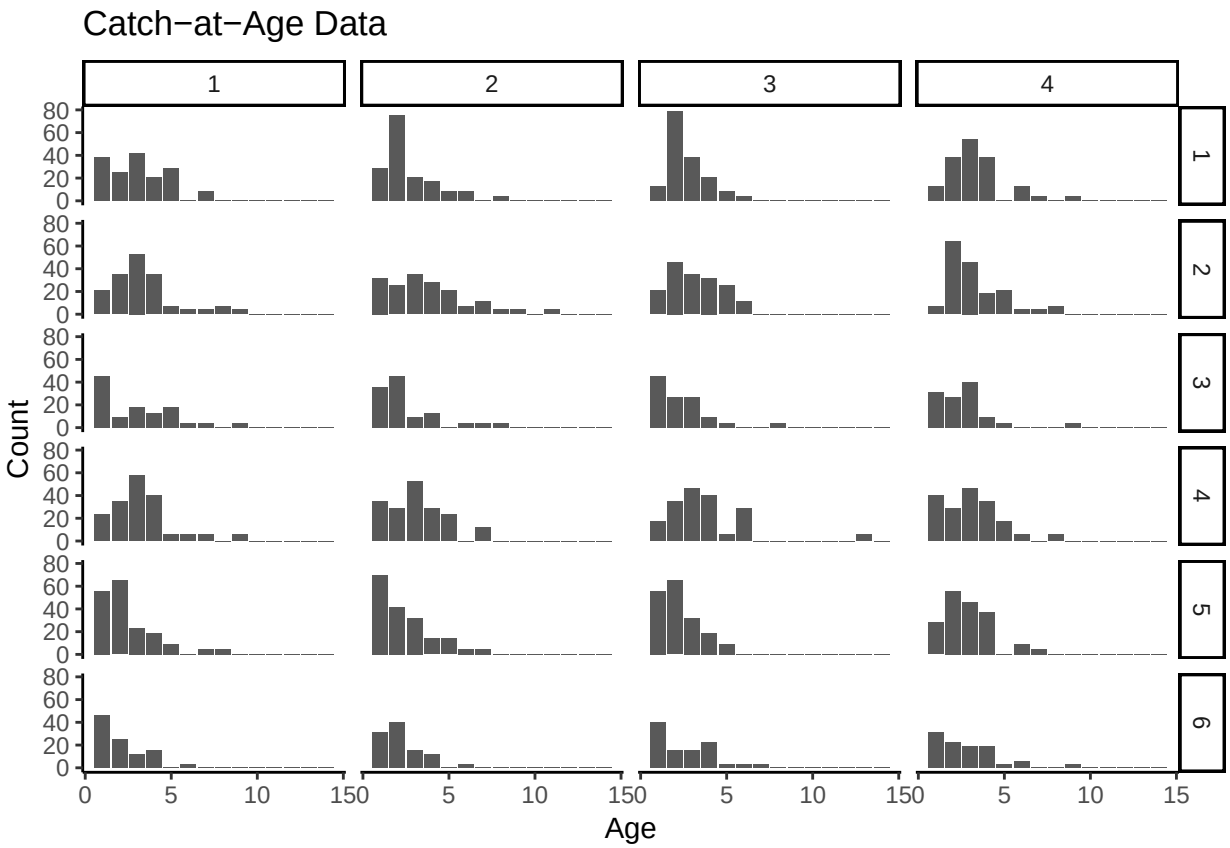

```
## Plotting: Lbar Data
```


Mean Length Data
V1
V2
V3
V4
V5
V6
80
75
70
65
60
55
Mean Length
Year

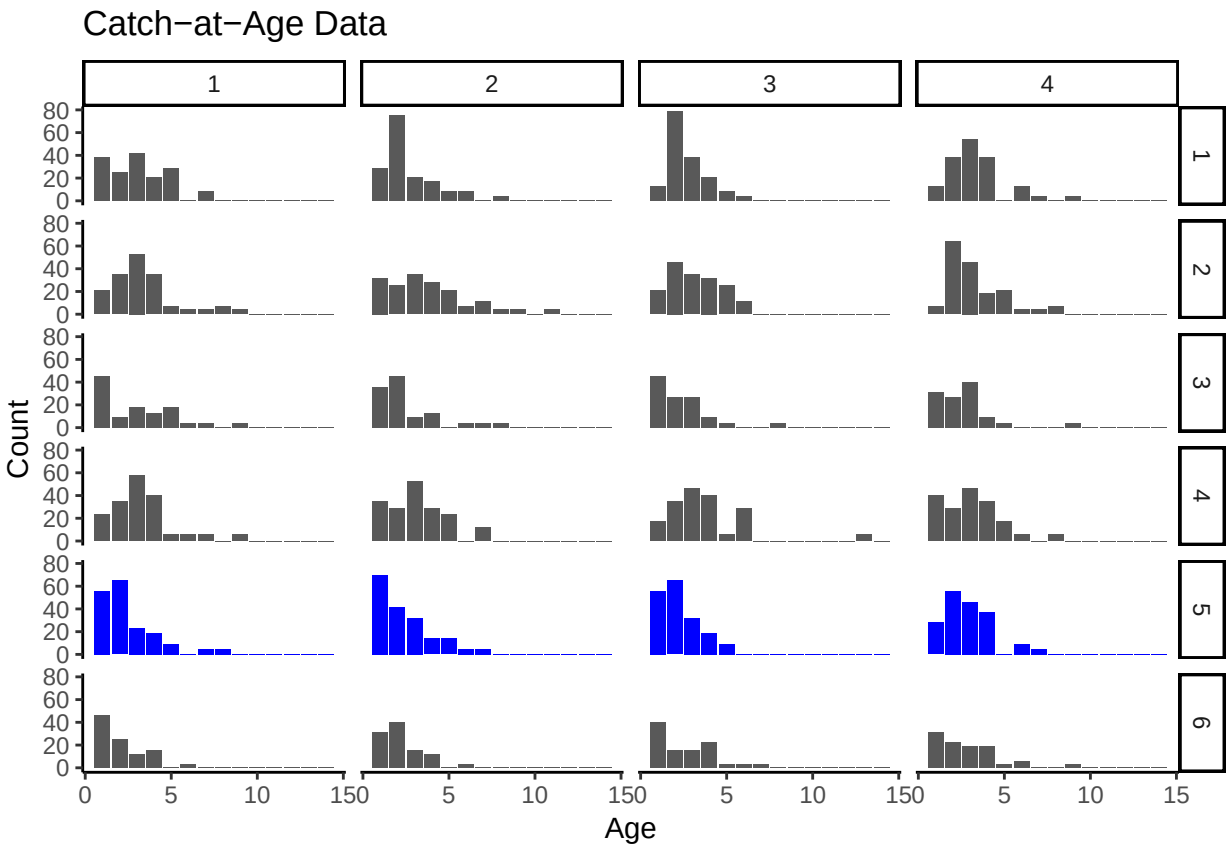


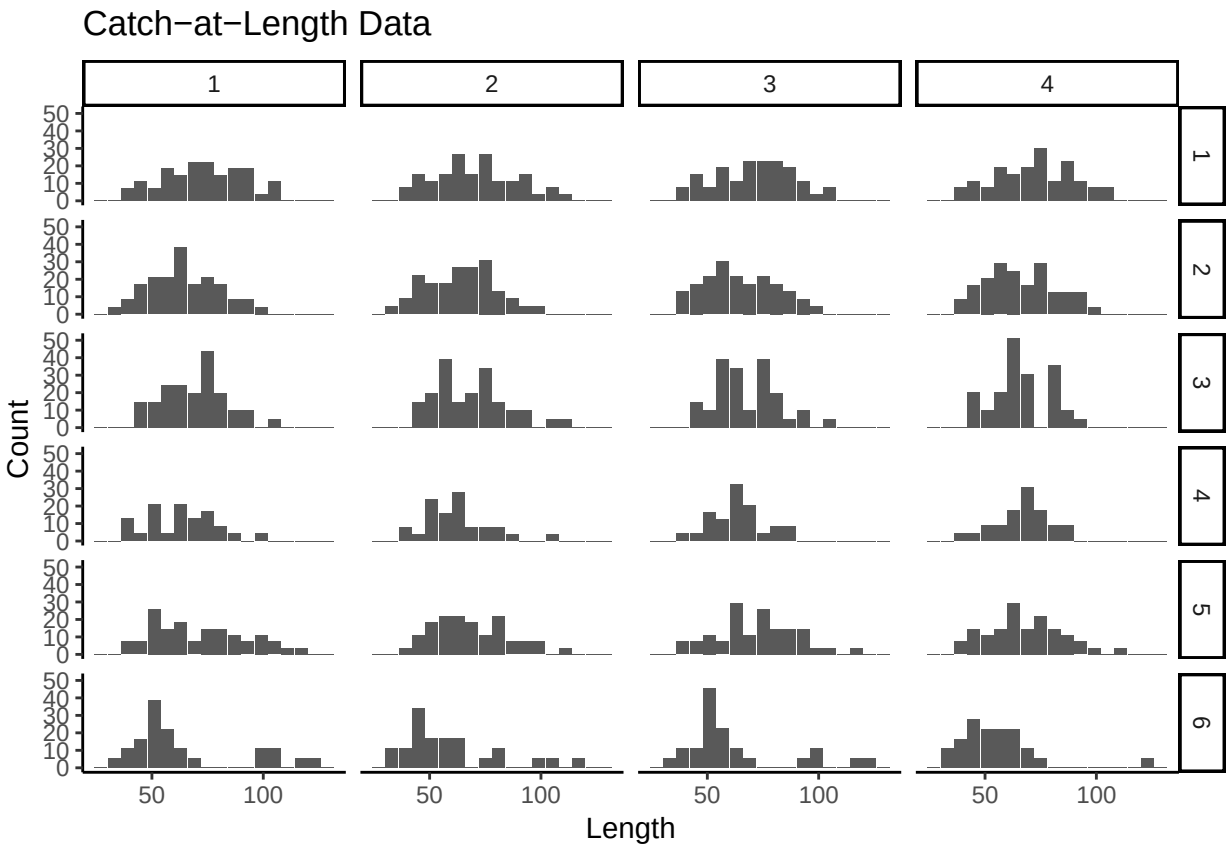

```
## Plotting: Catch-at-Age Data
```

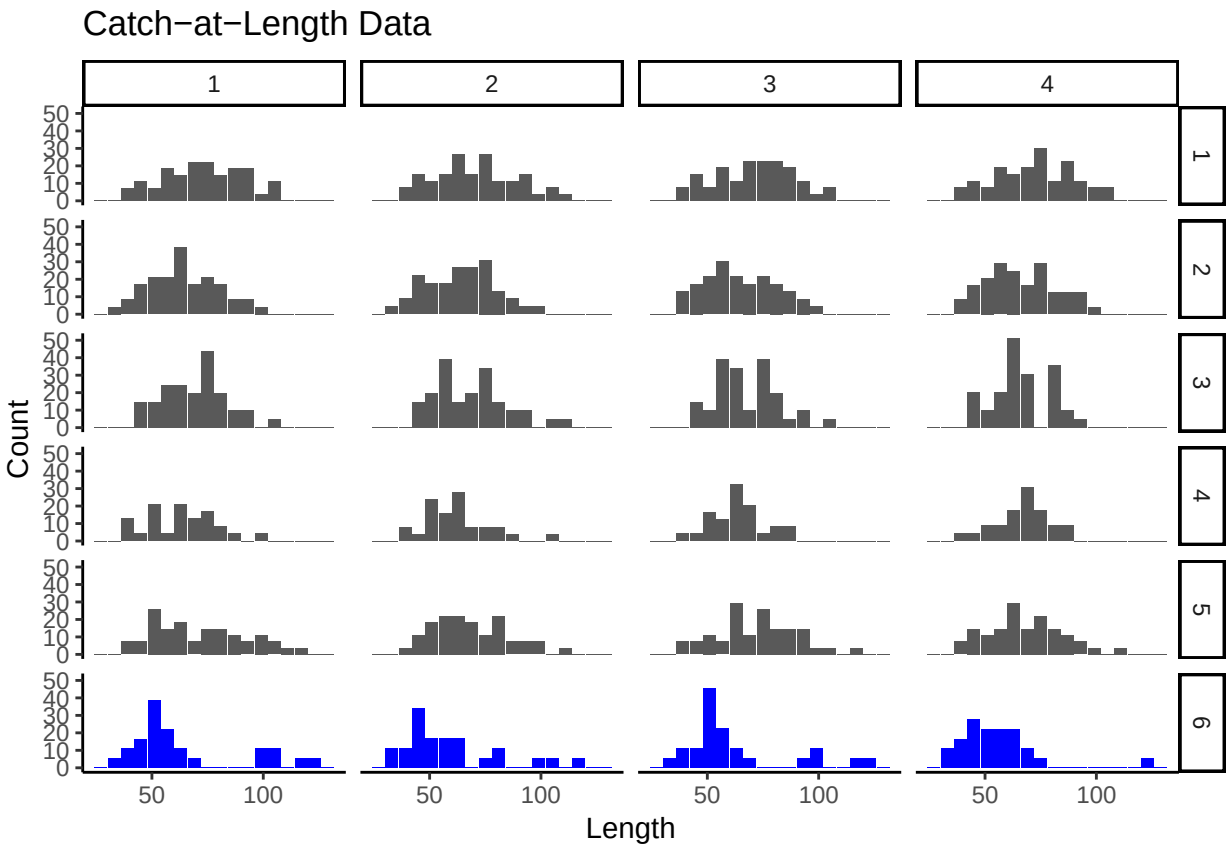

Lbar Data
V1
V2
V3
V4
V5
V6
100
90
80
70
Lbar
Year




```
## Plotting: Catch-at-Length Data
```


Catch−at−Age Data
1
2
3
4
1
2
3
4
5
6
Count
Age
0
20
40
60
80
0
5
10
150
15


Catch−at−Length Data
1
2
3
4
1
2
3
4
5
6
Count
Length
0
10
20
30
40
50
50
100



The `Turing` function is useful for evaluating if your OM adequately produces fishery data that appears similiar (e.g as variable) as your real observed data.

Chapter 25

Customizing the Operating Model

25.1 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.

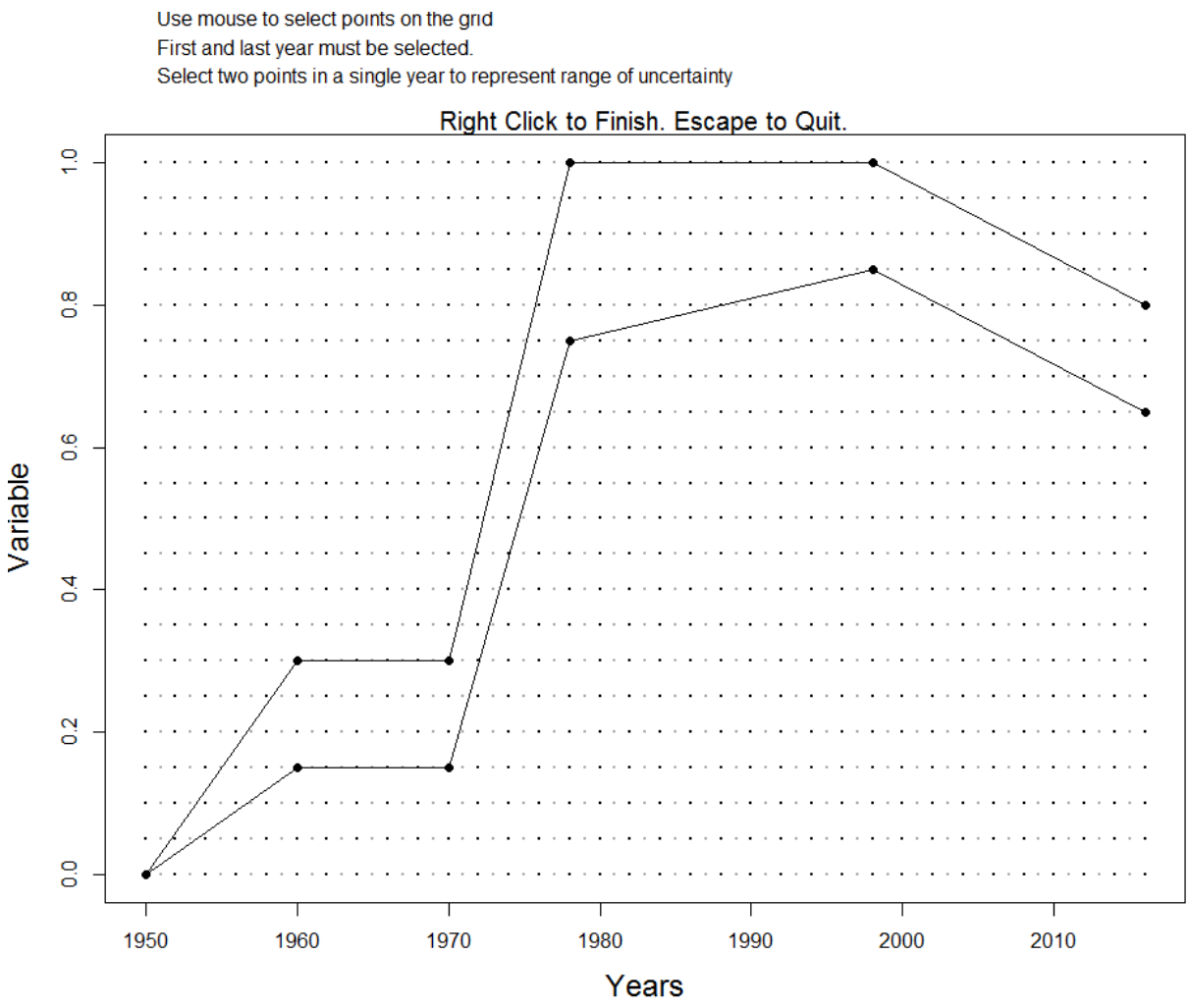
25.1.1 Historical Trends in Fishing Mortality

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.

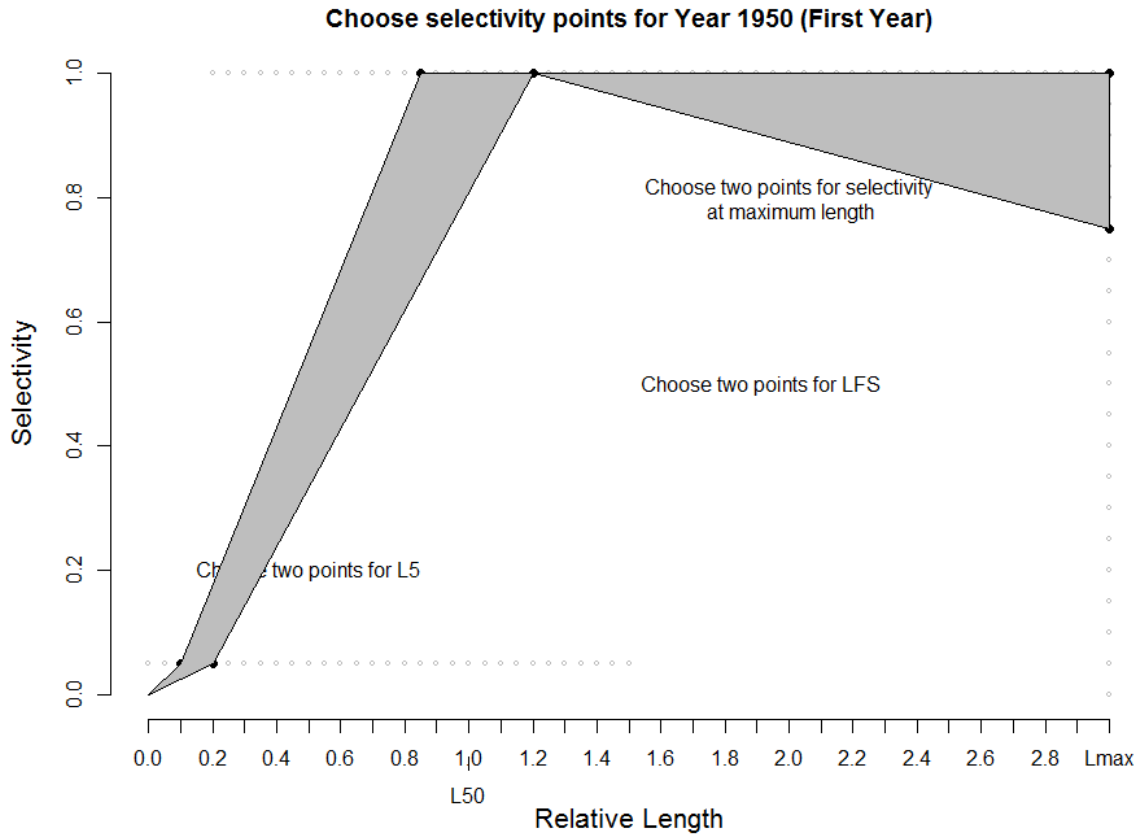
25.1.2 Historical Trends in Selectivity Pattern

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.

25.1.3 Including Existing MPAs

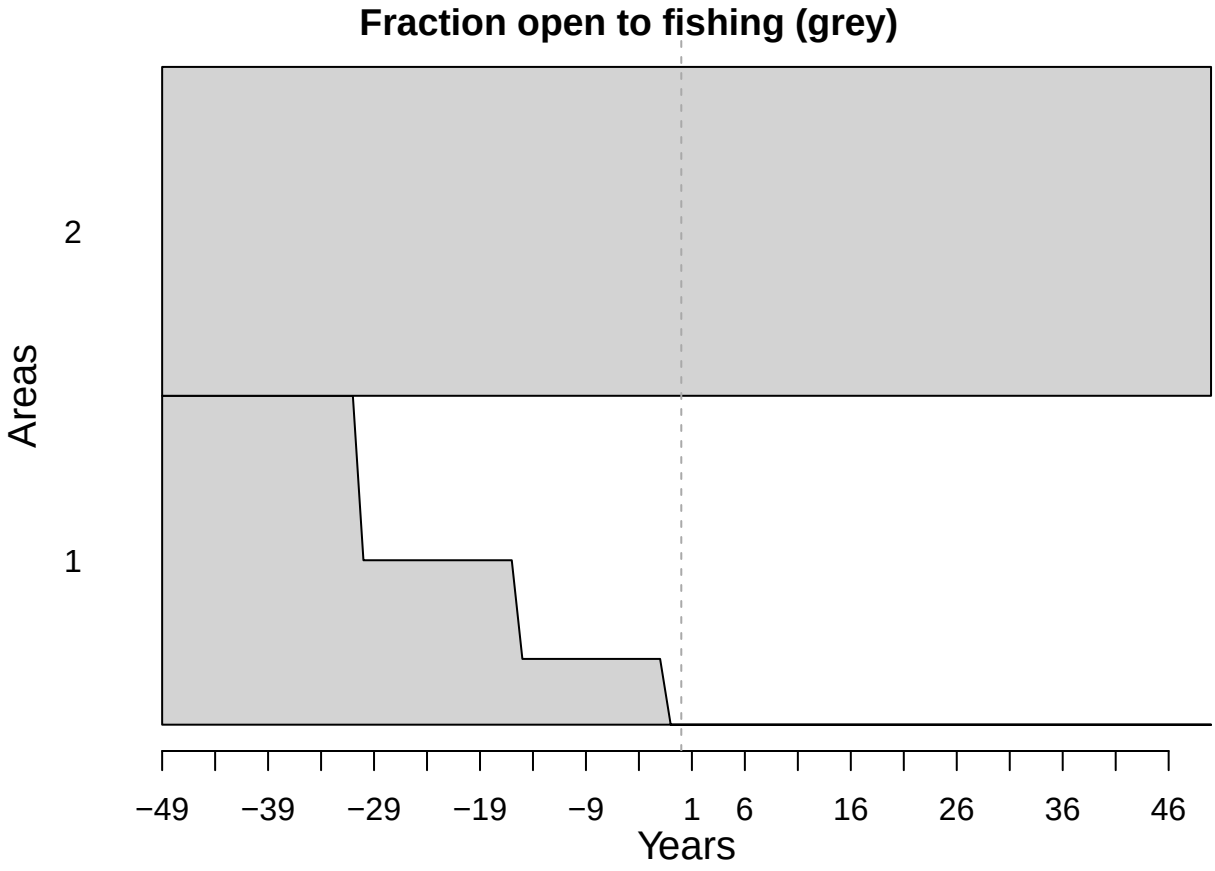
By default the MSE assumes that there are no spatial closures in the historical period. Existing spatial closures can be accounted for with the `MPA` slot in the `Fleet` or `OM` object.

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)
```



25.2 Size-Specific Natural Mortality

25.2.1 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.

25.2.2 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.

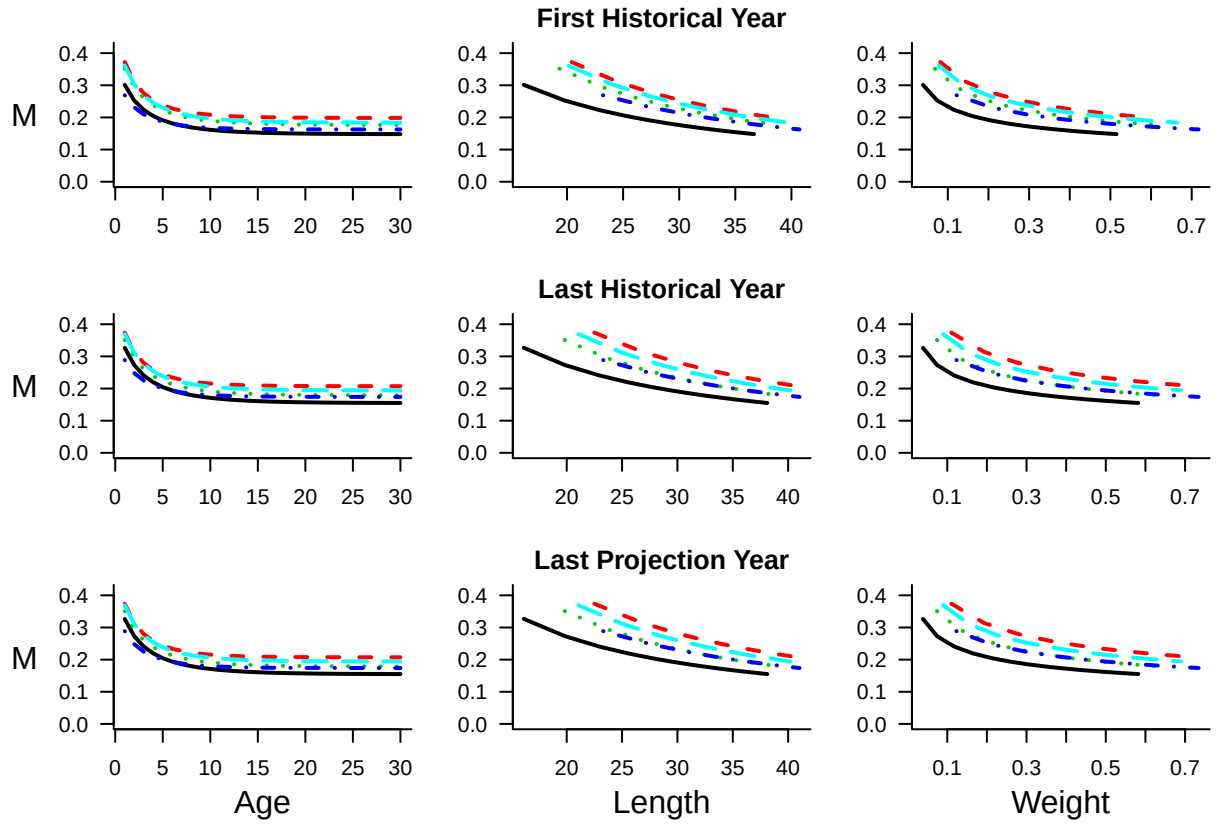
Because DLMtool uses an age-structured model, M is calculated as a function of age:

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

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:

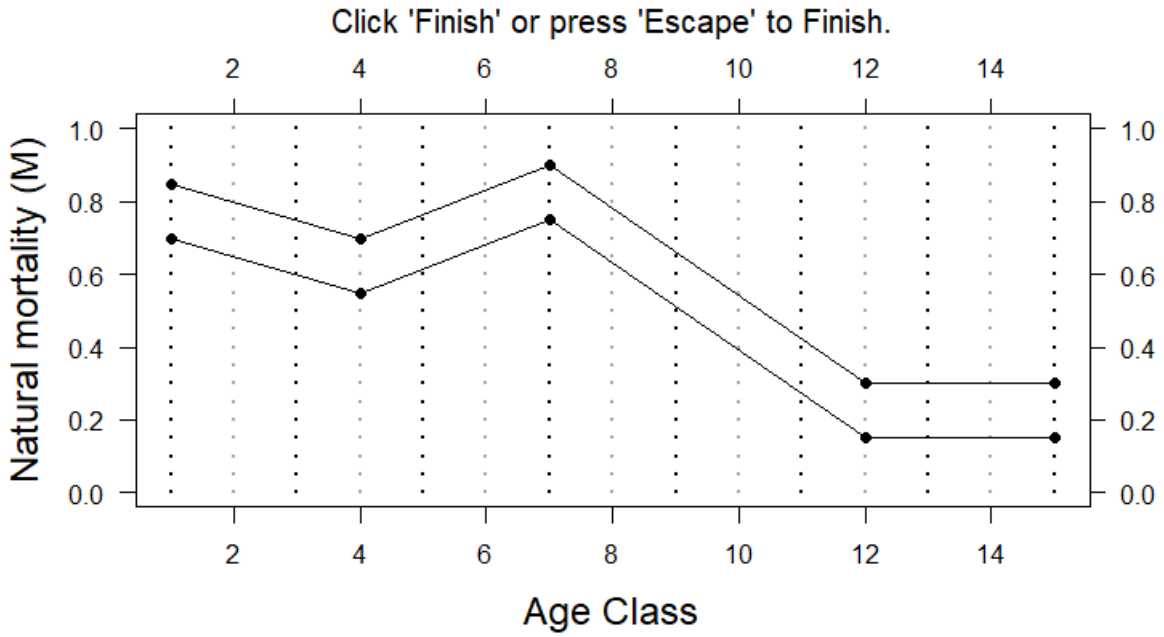
```
Mackerel@Mexp <- c(-0.315, -0.261)
plotM(Mackerel)
```



25.2.3 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)
```



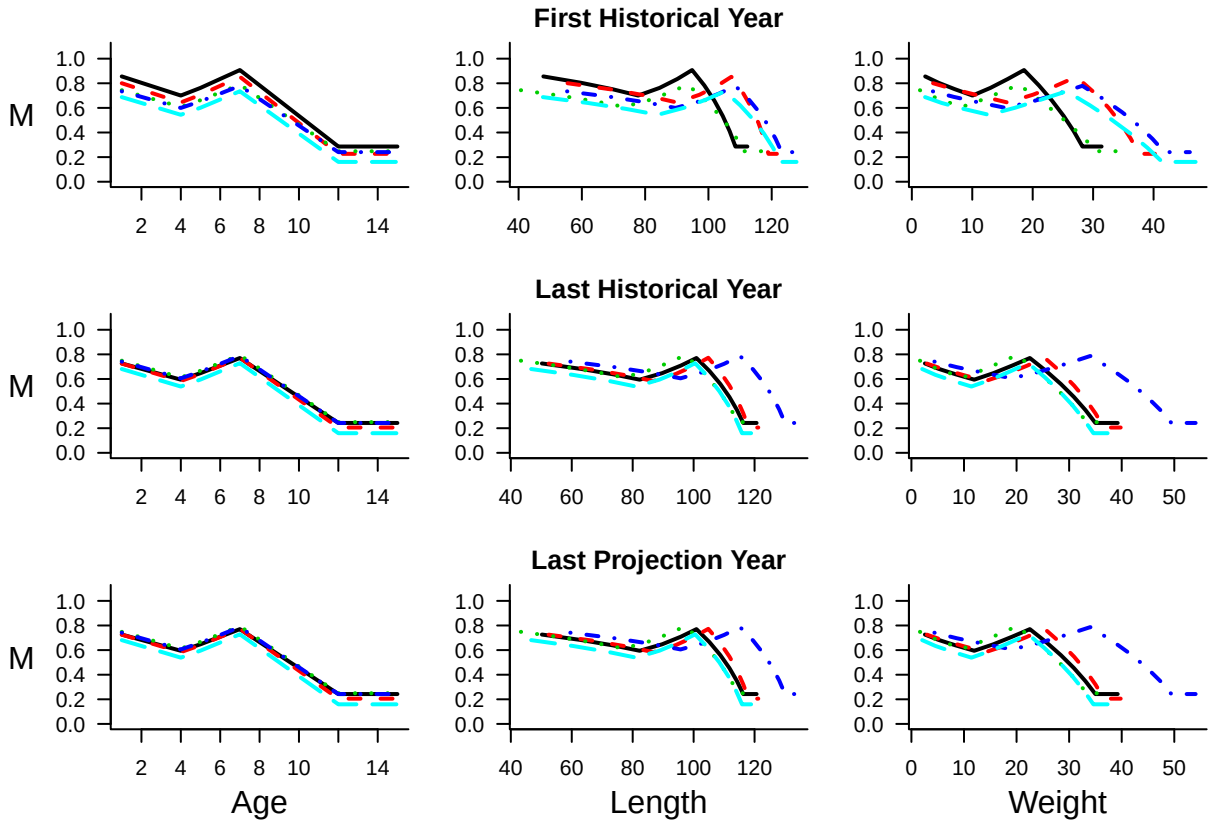
Click here for a larger version of the image.

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:

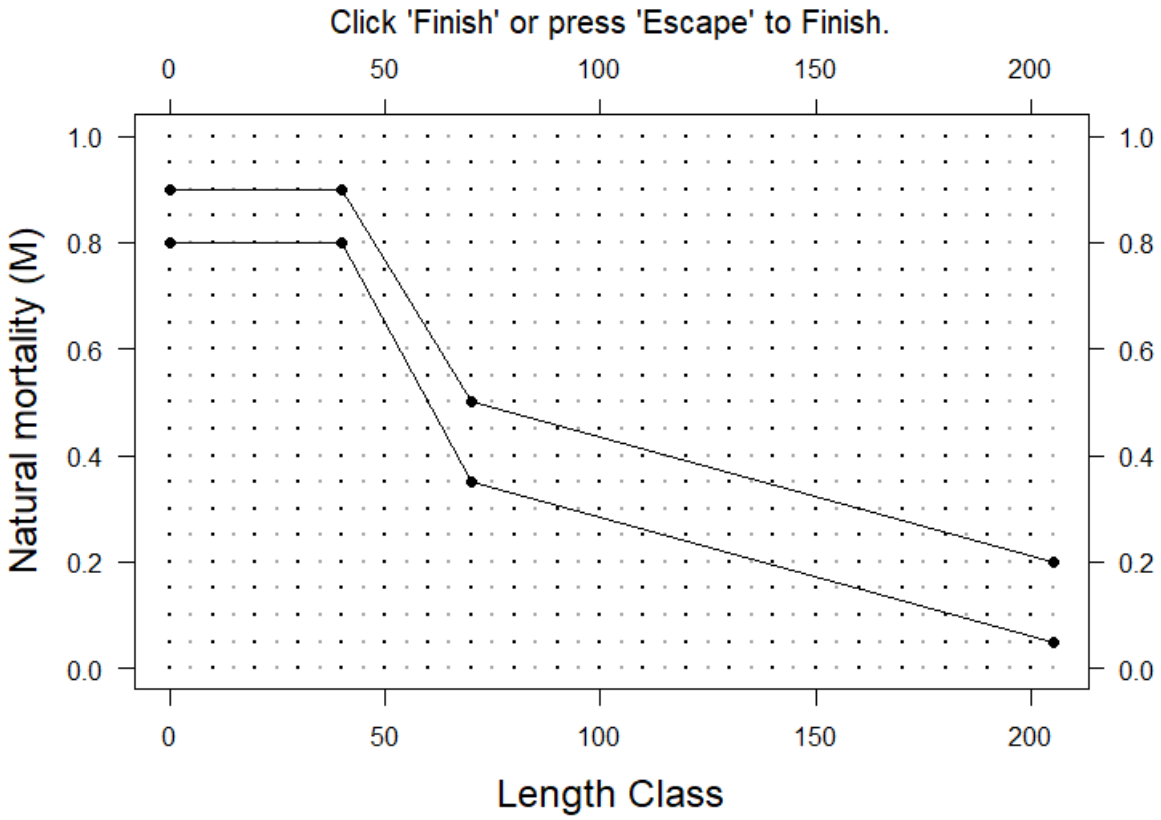
```
plotM(OM)
```



25.2.4 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:

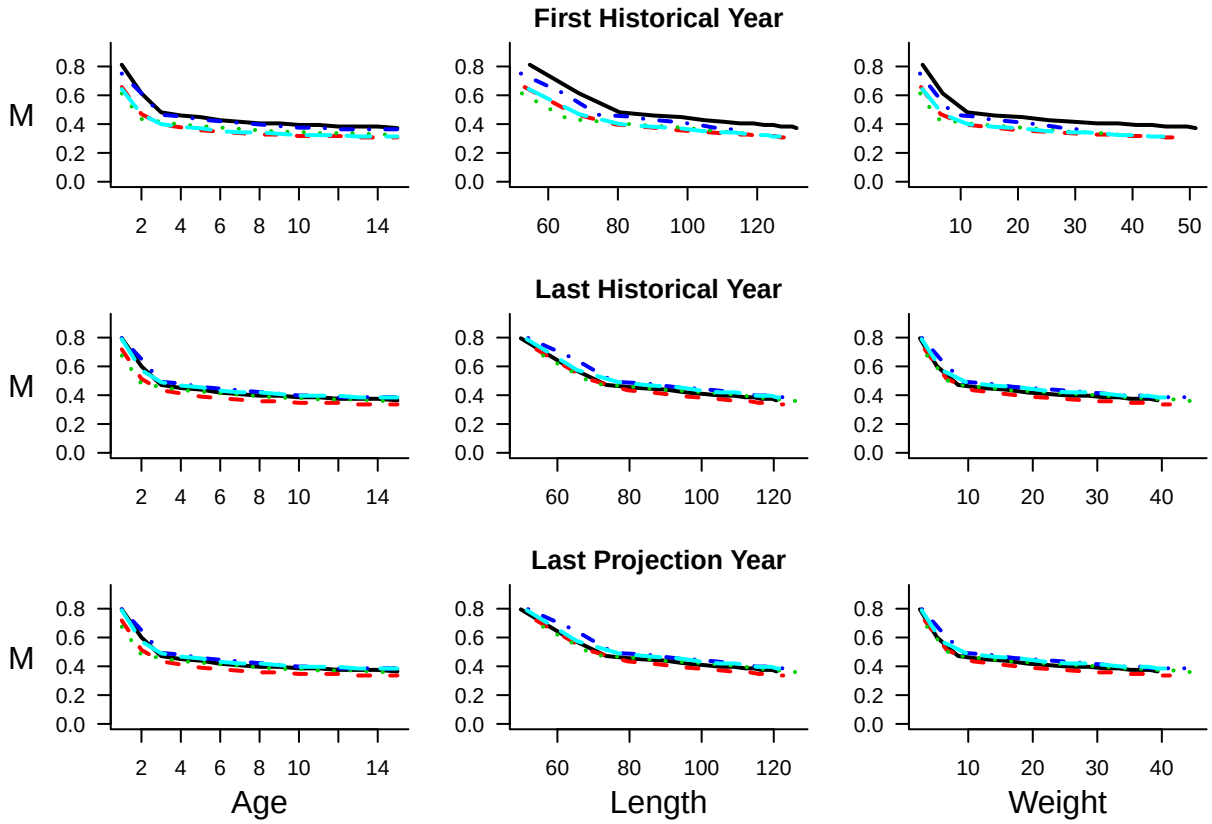
```
str(OM@cpars)
```

```
## 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
```



25.3 Selection, Retention and Discard Mortality

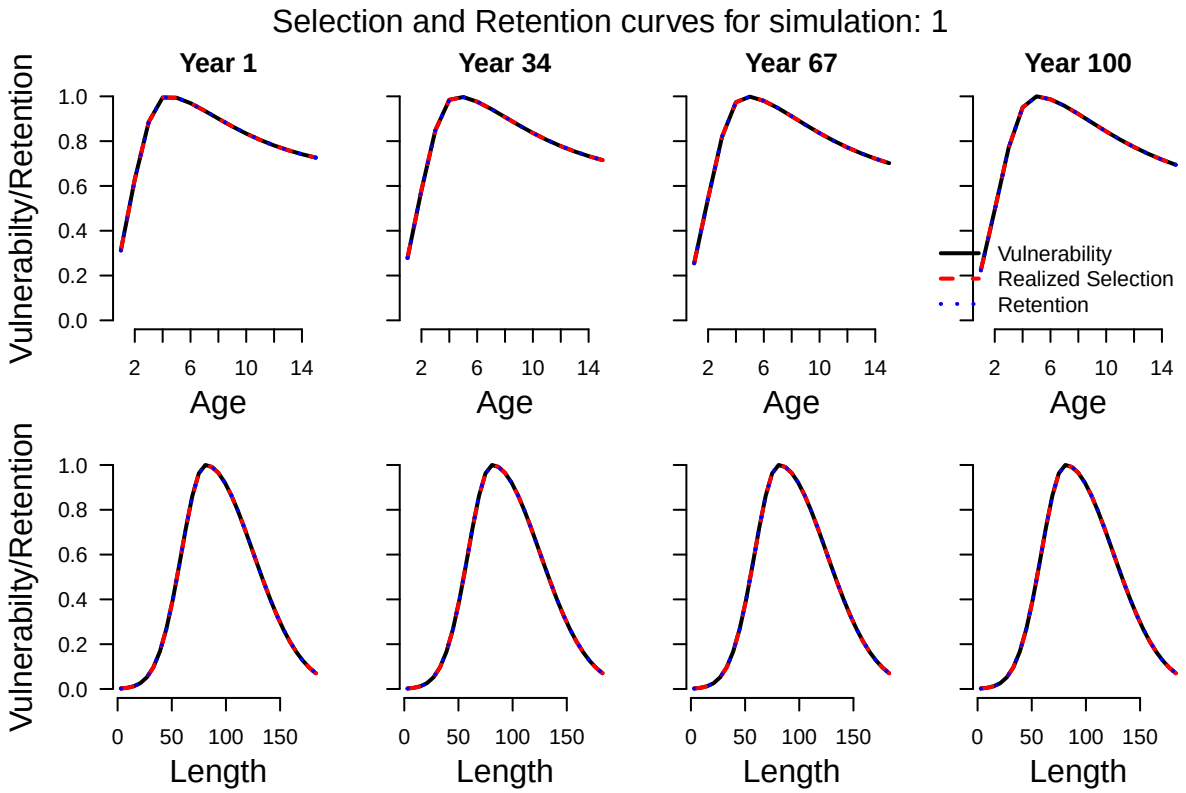
25.3.1 Fishery Selection Curve

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.

25.3.2 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.

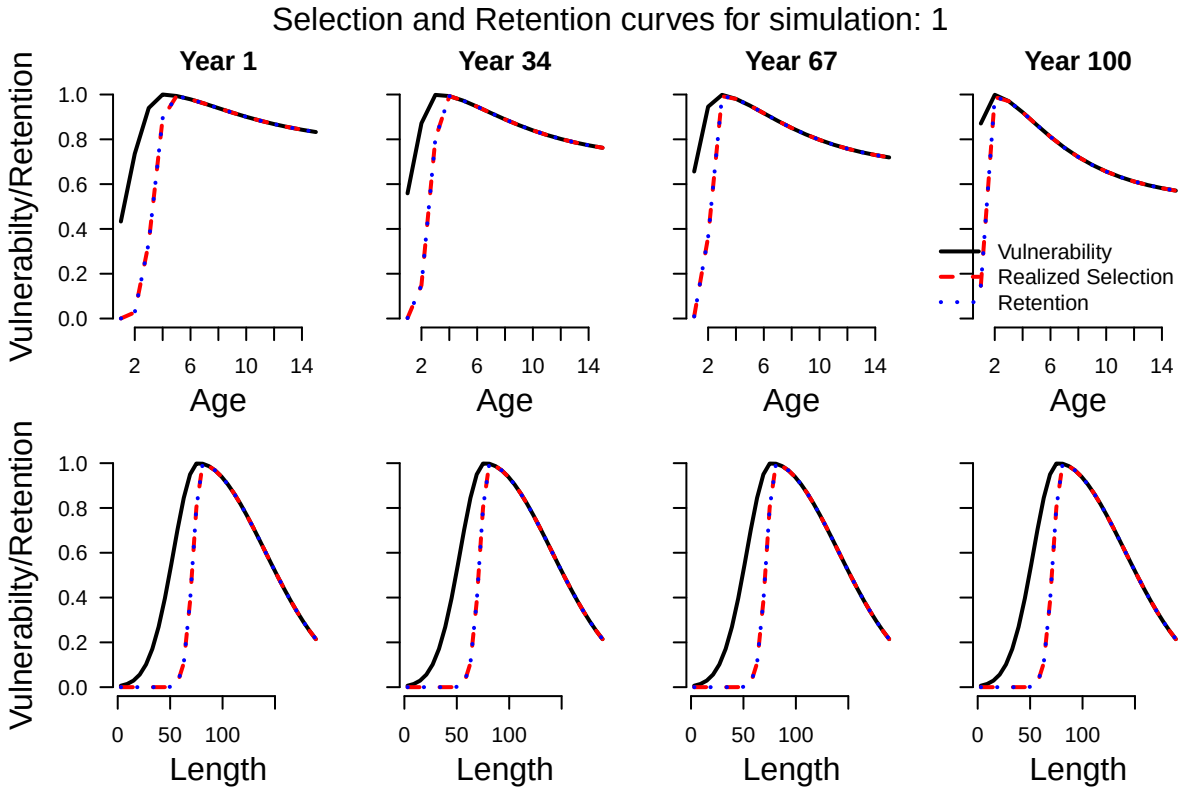
The retention curve can be modified by providing values for these slots:

```
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.

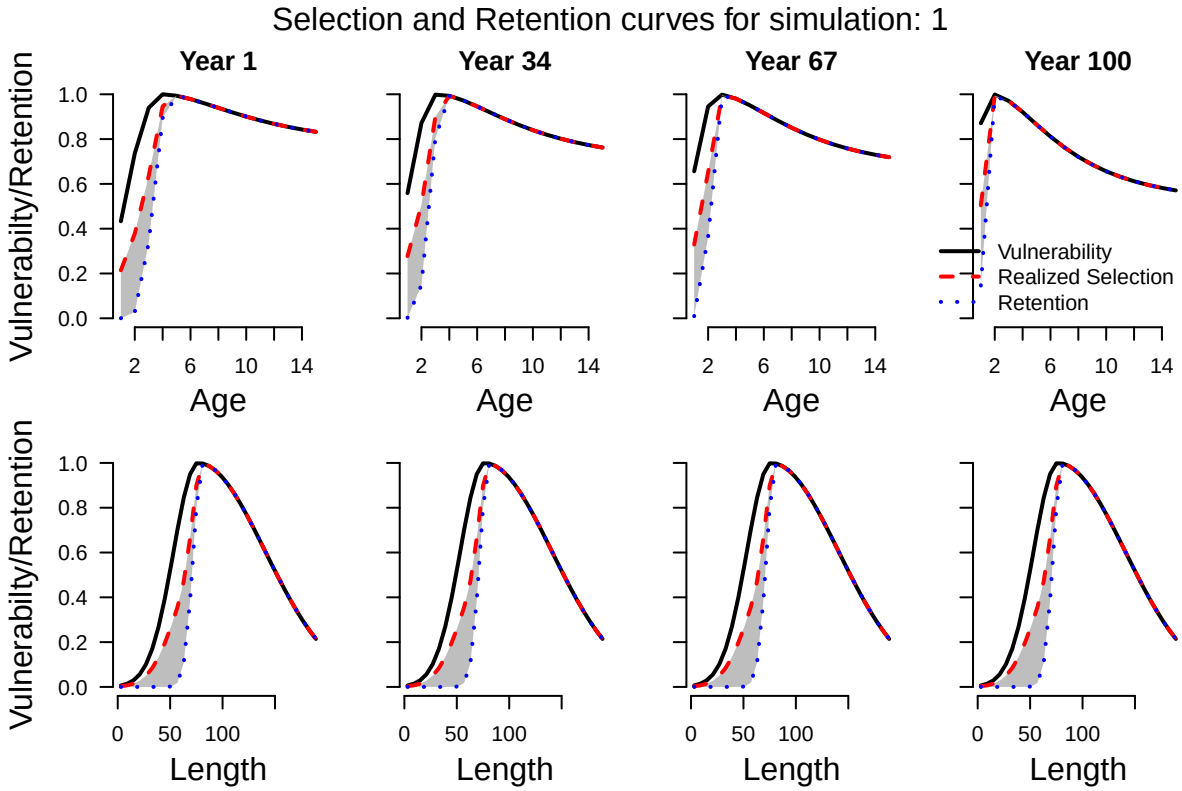
Because we are assuming no discard mortality in this case, the realized selection and retention curves are equivalent. This means that although fish of age/length between the vulnerability and retention curves are selected by the fishery, they are discarded with 100% survival and therefore are not removed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

25.3.3 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.

25.3.4 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.

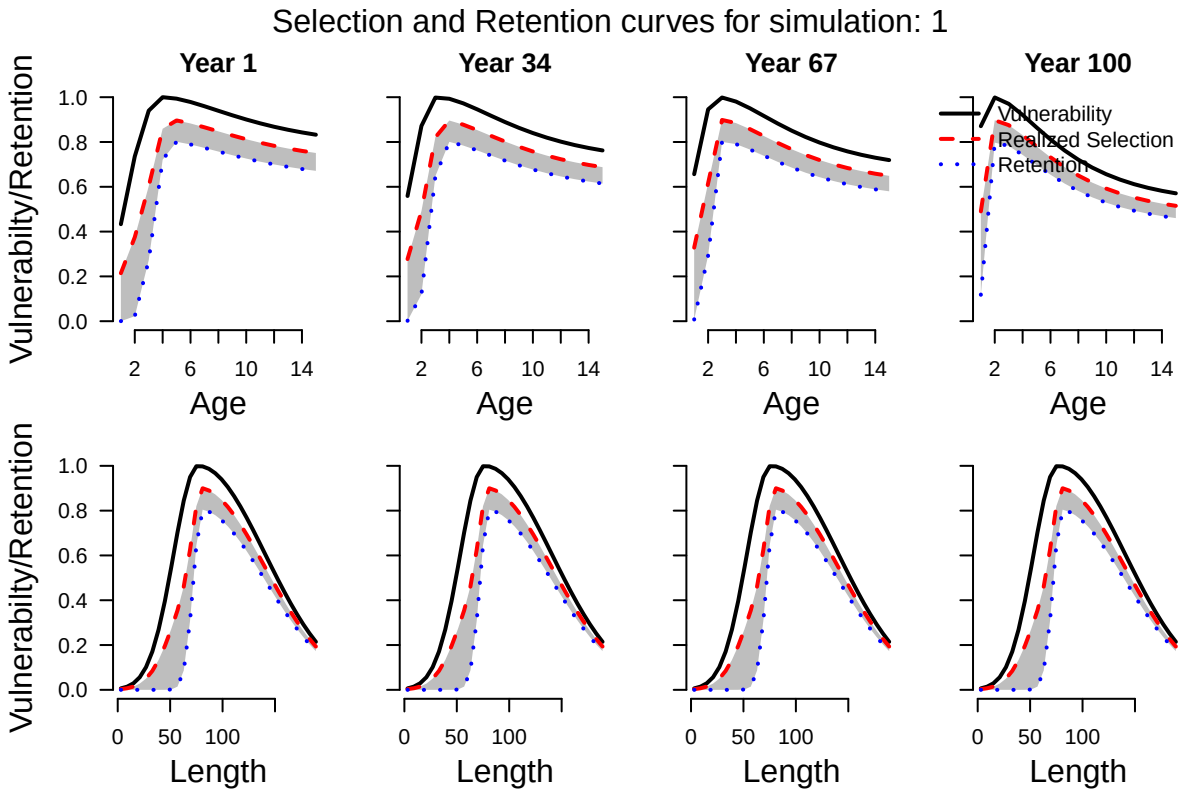
For example, here we assume that between 10 and 20% of all age/size classes are discarded by the fishery:

```
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
```



25.4 Variable Management Interval

TO DO

Chapter 26

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.

26.1 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] "CC2"         "CC3"         "CC4"         "CC5"         "CompSRA"
## [11] "CompSRA4010" "DAAC"        "DBSRA"       "DBSRA_40"    "DBSRA4010"
## [16] "DCAC"        "DCAC_40"     "DCAC_ML"     "DCAC4010"    "DCACs"
## [21] "DD"          "DD4010"      "DepF"        "DynF"        "Fadapt"
## [26] "Fdem"        "Fdem_CC"     "Fdem_ML"     "Fratio"      "Fratio_CC"
## [31] "Fratio_ML"   "Fratio4010"  "GB_CC"       "GB_slope"    "GB_target"
## [36] "Gcontrol"    "HDAAC"       "ICI"         "ICI2"        "Iratio"
## [41] "Islope1"     "Islope2"     "Islope4"     "IT10"        "IT5"
## [46] "Itarget1"    "Itarget2"    "Itarget3"    "Itarget4"    "ITM"
## [51] "L95target"   "Lratio_BHI"  "Lratio_BHI2" "LstepCC1"    "LstepCC2"
## [56] "LstepCC3"    "LstepCC4"    "Ltarget1"    "Ltarget2"    "Ltarget3"
## [61] "Ltarget4"    "MCD"         "MCD4010"     "Rcontrol"    "Rcontrol2"
## [66] "SBT1"        "SBT2"        "SPmod"       "SPMSY"       "SPslope"
```

```
## [71] "SPSRA"       "SPSRA_ML"    "YPR"         "YPR_CC"      "YPR_ML"
## [76] "avgMP"       "TCPUE"       "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, plot = FALSE)
## {
##     rundcac <- DCAC_(x, Data, reps, updateD = TRUE)
##     TAC <- TACfilter(rundcac$dcac)
##     if (plot)
##         DCAC_plot(x, Data, dcac = rundcac$dcac, TAC, Bt_K = rundcac$Bt_K,
##             yrs = 1:length(Data@Year))
##     Rec <- new("Rec")
##     Rec@TAC <- TAC
##     Rec
## }
## <bytecode: 0x00000000193cde18>
## <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 four arguments: `x`, `Data`, `reps` and `plot` (the last argument was added recently and is optional).

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
```

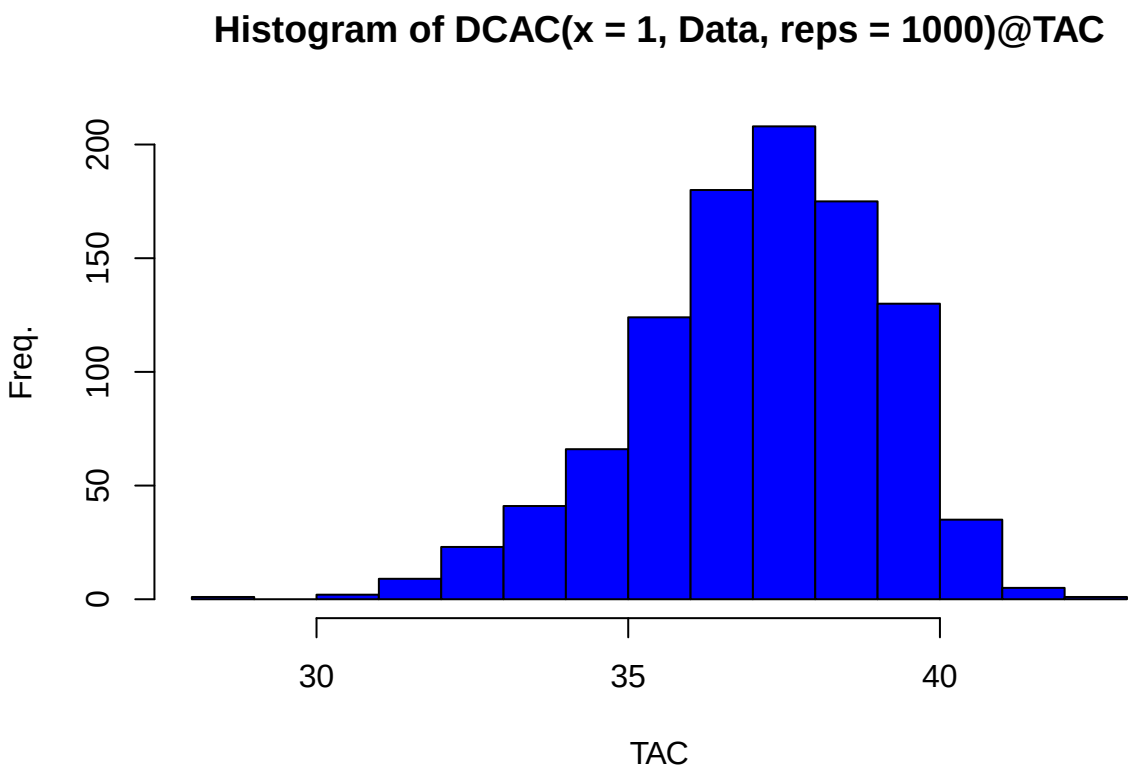
We could generate a single TAC recommendation from these data using DCAC by specifying position 1 (there is only 1 simulation) and by setting reps=1 (we want a single DCAC TAC recommendation)

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

```
## TAC (median)
##     37.49571
```

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")
```



26.2 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 (we will ignore `plot` for now):

- 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] 75.08556 74.85677 80.09622 67.34673 82.91729 66.61270 83.49672
##  [8] 85.44908 79.53824 81.45144
```

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')
```

26.3 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")
```

26.4 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, reps, plot = FALSE)
## {
##     rec <- new("Rec")
##     rec@Effort <- 1 * Data@MPeff[x]
##     if (plot)
##         curE_plot(x, rec, Data)
```

```
##     rec
## }
## <bytecode: 0x0000000019554c98>
## <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, reps, plot = FALSE)
## {
##     rec <- new("Rec")
##     rec@Allocate <- 1
##     rec@Spatial <- c(0, rep(1, Data@nareas - 1))
##     if (plot)
##         barplot(rec@Spatial, xlab = "Area", ylab = "Fraction Open",
##             ylim = c(0, 1), names = 1:Data@nareas)
##     return(rec)
## }
## <bytecode: 0x0000000016f494d0>
## <environment: namespace:DLMtool>
## attr(,"class")
## [1] "MP"
```

In contrast `MRnoreal` does not reallocate fishing effort:

```
MRnoreal
```

```
## function (x, Data, reps, plot = FALSE)
## {
##     rec <- new("Rec")
##     rec@Allocate <- 0
##     rec@Spatial <- c(0, rep(1, Data@nareas - 1))
##     if (plot)
##         barplot(rec@Spatial, xlab = "Area", ylab = "Fraction Open",
##             ylim = c(0, 1), names = 1:Data@nareas)
##     return(rec)
## }
## <bytecode: 0x0000000016f7abd8>
## <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, reps, plot = FALSE)
## {
##     rec <- new("Rec")
##     rec@LFR <- Data@L50[x]
##     rec@LR5 <- rec@LFR * 0.95
##     if (plot)
```

```
##         size_lim_plot(x, Data, rec)
##     rec
## }
## <bytecode: 0x0000000018ff0700>
## <environment: namespace:DLMtool>
## attr(,"class")
## [1] "MP"
```

26.4.1 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')
```

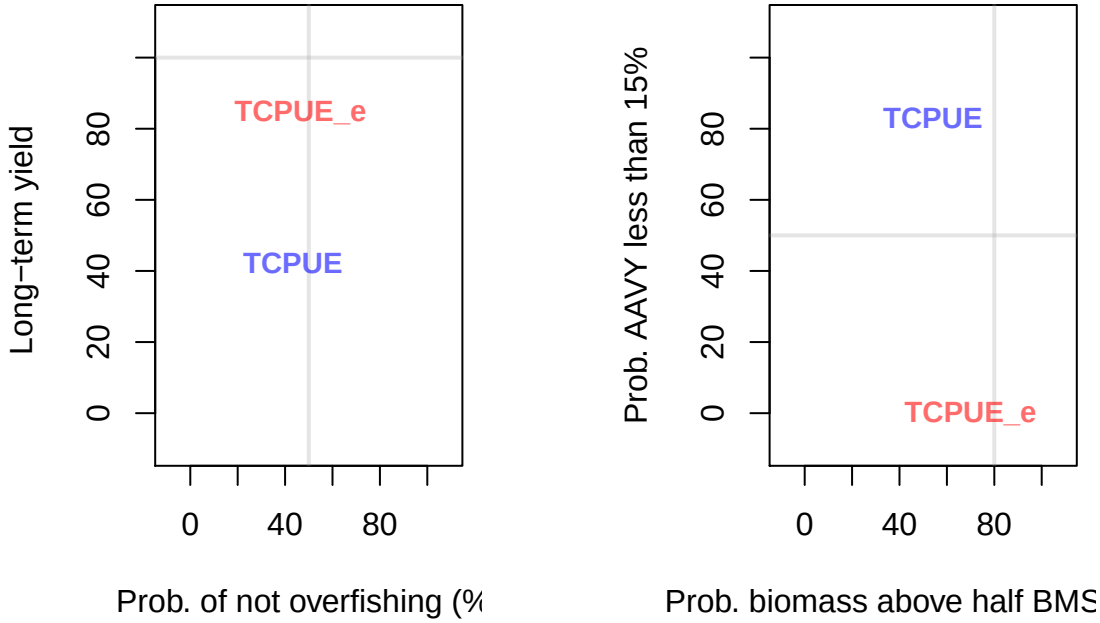
Let's test the two MPs and see how they peform:

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

```
## Running MSE in parallel on 10 processors
```

```
## MSE completed
```

```
NOAA_plot(testMSE)
```




```
##         PNOF  B50  LTY   VY
## TCPUE   43.3 54.1 42.5 83.3
## TCPUE_e 46.5 70.0 84.6  0.0
```

Chapter 27

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 - Note: these functions have now been moved to `MSEtool`). 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.

27.1 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:

```
head(validcpars())
```

```
##    Var.                        Dim.                          Desc.  Type
## 1    R0 numeric vector length nsim            Virgin recruitment Stock
## 2     M numeric vector length nsim             Natural mortality Stock
## 3  Mexp numeric vector length nsim    Lorenzen M-weight exponent Stock
## 4   Msd numeric vector length nsim Inter-annual variability in M Stock
## 5 Mgrad numeric vector length nsim                 Gradient in M Stock
## 6     h numeric vector length nsim                     Steepness Stock
```

The custom parameters are divided into 5 different types: 'Stock', 'Fleet', 'Obs', and 'Imp' corresponding to the OM components of the same names, and 'internal' for internal operating model parameters that over-ride or ignore the values in the `OM` slots.

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.

27.2 Correlated samples

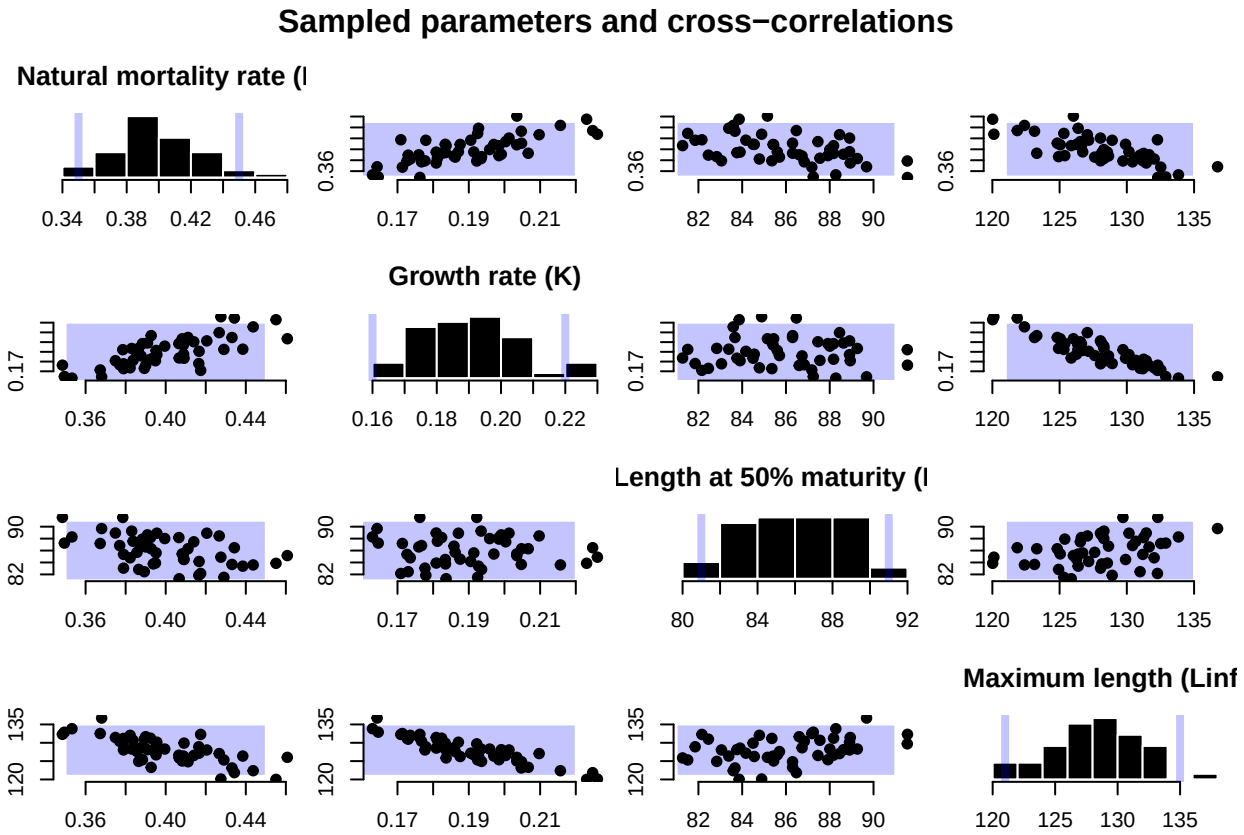
As the `cpars` feature is used to provide correlated samples to the MSE, it is important that the same number of custom parameters are provided for each variable. In most cases, this is simply a vector `nsim` long.

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
```

27.3 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. These are referred to as *internal* custom parameters.

A list of valid internal cpars can be found by using the `validcpars` function. Here, for presentation purposes, we print just the first two columns:

```
val_int <- validcpars('internal')
val_int[,c(1,2)]
```

```
##           Var.                                                    Dim.
## 1     CAL_bins                                        numeric vector
## 2  CAL_binsmid                        numeric vector length(CAL_bins)-1
## 3          L95                                numeric vector length nsim
## 4       Perr_y numeric matrix dim = c(nsim, maxage+proyears+nyears-1)
## 5  M_at_Length                      numeric matrix dim = c(n.lengths, 3)
## 6        Asize                       numeric matrix dim = c(nsim, narea)
## 7       Karray          numeric matrix dim = c(nsim, nyears+proyears)
## 8    Linfarray          numeric matrix dim = c(nsim, nyears+proyears)
## 9       Marray          numeric matrix dim = c(nsim, nyears+proyears)
## 10       Krand          numeric matrix dim = c(nsim, nyears+proyears)
## 11    Linfrand          numeric matrix dim = c(nsim, nyears+proyears)
## 12       Mrand          numeric matrix dim = c(nsim, nyears+proyears)
## 13        ageM          numeric matrix dim = c(nsim, nyears+proyears)
## 14       age95          numeric matrix dim = c(nsim, nyears+proyears)
## 15  M_ageArray   numeric array dim = c(nsim, maxage, nyears+proyears)
## 16     Mat_age   numeric array dim = c(nsim, maxage, nyears+proyears)
## 17      LatASD   numeric array dim = c(nsim, maxage, nyears+proyears)
## 18      Wt_age   numeric array dim = c(nsim, maxage, nyears+proyears)
## 19     Len_age   numeric array dim = c(nsim, maxage, nyears+proyears)
## 20         mov      numeric array dim = c(nsim, maxage, narea, narea)
## 21        Find                     numeric matrix dim = c(nsim, nyears)
## 22     dFfinal                                numeric vector length nsim
## 23           V   numeric array dim = c(nsim, maxage, nyears+proyears)
## 24        retA   numeric array dim = c(nsim, maxage, nyears+proyears)
## 25        retL numeric array dim = c(nsim, nCALbins, nyears+proyears)
## 26    lenMbias                                numeric vector length nsim
## 27       Mbias                                numeric vector length nsim
## 28       Kbias                                numeric vector length nsim
```

```
## 29      t0bias                                numeric vector length nsim
## 30    Linfbias                                numeric vector length nsim
## 31     LFCbias                                numeric vector length nsim
## 32    FMSYbias                                numeric vector length nsim
## 33  FMSY_Mbias                                numeric vector length nsim
## 34 BMSY_B0bias                                numeric vector length nsim
## 35    Irefbias                                numeric vector length nsim
## 36    Brefbias                                numeric vector length nsim
## 37    Crefbias                                numeric vector length nsim
## 38       Dbias                                numeric vector length nsim
## 39       hbias                                numeric vector length nsim
```

We can see that there are 39 valid internal cpars. The internal cpars are parameters that are derived from one or more of the `OM` slots.

For example, `M_ageArray` is an internal array that describes the natural mortality rate at age for each simulation and year. Typically, this array is derived from the `M`, `M2`, `Mexp`, `Mgrad` and `Msd` slots in the `Stock` object. Using the `cpars` feature we can override these values in the `OM` and provide our own values for M for each simulation, age, and year.

Information on the `M_ageArray` variable in cpars can be found in the output of the `validcpars` function:

```
val_int[15,]
```

```
##          Var.                                                    Dim.
## 15 M_ageArray numeric array dim = c(nsim, maxage, nyears+proyears)
##                                                    Desc.     Type
## 15 M by simulation, age and year. No variability is added internal
```

To generate our own values of the M-at-age array we would populate `OM@cpars$M_ageArray` with an array with dimensions `OM@nsim`, `OM@maxage`, `OM@nyears+OM@proyears`.

The `cpars` feature is very powerful but also somewhat complicated, especially if you are using internal custom parameters. For example, by default DLMtool uses the `OM@L50_95` slot (the increment between length at 50% maturity (`OM@L50`) and length at 95% maturity (`L95`)) to calculate the internal parameter `L95`. This is neccessary to ensure that `L95` is always greater than `L50`. Using internal parameters in `cpars` it is possible to pass values to `L95` directly, however now it is up to you to make sure that the `L95` values are greater than the corresponding `L50` values, otherwise you end up with a species where fraction mature decreases with age/size!

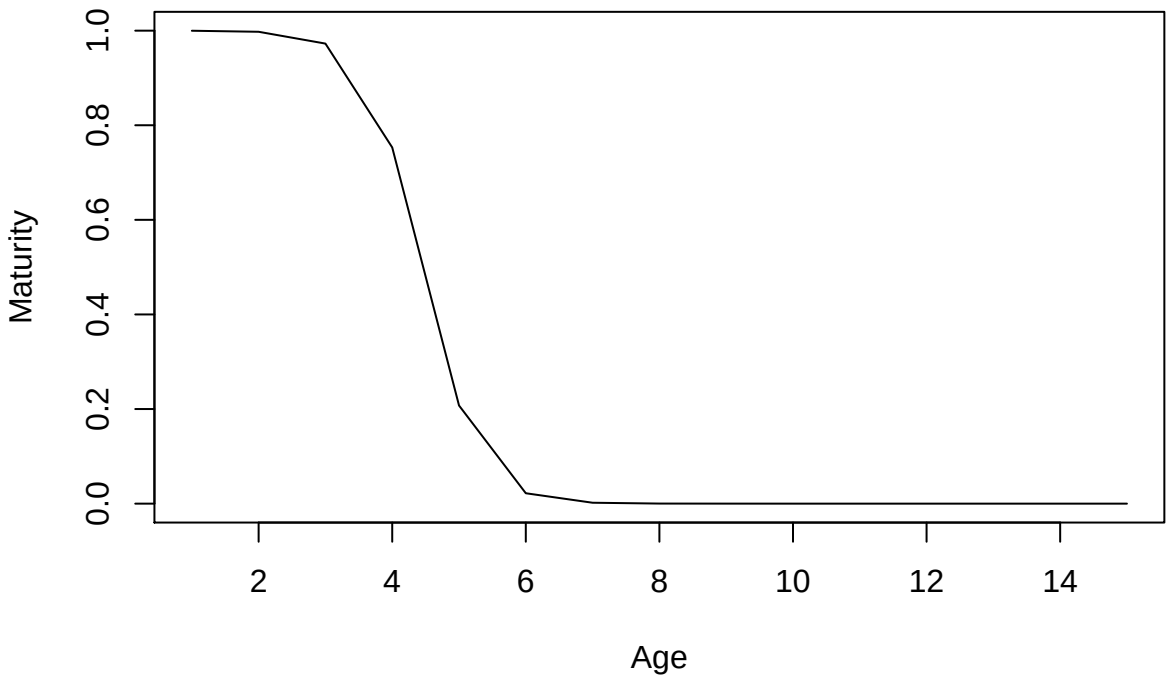
```
OM <- testOM
OM@cpars$L95 <- rep(80, OM@nsim)
temp <- runMSE(OM, Hist=TRUE)
```

```
## Loading operating model

## valid custom parameters (OM@cpars) found:
## L95

## Optimizing for user-specified movement

## Optimizing for user-specified depletion

## 10 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 B-low reference points
```

```
## Calculating reference yield - best fixed F strategy

## Returning historical simulations
data.frame(L50=temp$SampPars$L50, L95=temp$SampPars$L95, diff=temp$SampPars$L95 - temp$SampPars$L50)
```

```
##         L50 L95       diff
## 1  89.04273  80  -9.042727
## 2  84.24741  80  -4.247406
## 3  83.68463  80  -3.684629
## 4  88.46969  80  -8.469689
## 5  83.30473  80  -3.304726
## 6  84.33086  80  -4.330862
## 7  83.87020  80  -3.870200
## 8  84.89773  80  -4.897730
## 9  89.16130  80  -9.161301
## 10 83.98488  80  -3.984878
## 11 86.10303  80  -6.103030
## 12 83.05547  80  -3.055465
## 13 88.86433  80  -8.864327
## 14 87.95561  80  -7.955609
## 15 85.62968  80  -5.629683
## 16 81.58190  80  -1.581904
## 17 90.93998  80 -10.939979
## 18 86.11846  80  -6.118459
## 19 88.88535  80  -8.885348
## 20 83.99212  80  -3.992123
## 21 86.52170  80  -6.521703
## 22 81.59508  80  -1.595075
## 23 84.40471  80  -4.404715
## 24 88.14052  80  -8.140517
## 25 88.68731  80  -8.687307
## 26 85.37907  80  -5.379065
## 27 81.61598  80  -1.615975
## 28 87.74099  80  -7.740989
## 29 83.69400  80  -3.694005
## 30 90.34677  80 -10.346768
## 31 86.88600  80  -6.885997
## 32 85.89163  80  -5.891633
## 33 86.63998  80  -6.639976
## 34 89.23919  80  -9.239189
## 35 90.14945  80 -10.149446
## 36 83.51455  80  -3.514550
## 37 87.37439  80  -7.374391
## 38 90.71974  80 -10.719740
## 39 90.38542  80 -10.385425
## 40 90.03452  80 -10.034515
## 41 83.33089  80  -3.330892
## 42 88.57439  80  -8.574392
## 43 89.45315  80  -9.453146
## 44 81.80111  80  -1.801110
## 45 90.39487  80 -10.394870
## 46 90.97161  80 -10.971610
## 47 87.53739  80  -7.537389
## 48 81.60970  80  -1.609695
```

```
plot(1:OM@maxage, temp$AtAge$Mat_age[1,,1], type="l",
     xlab="Age", ylab="Maturity")
```



We have tried to include checks to ensure the model is simulating credible population dynamics, but as this somewhat contrived example shows, care is needed when using `OM@cpars` to specify internal parameters. We recommend first running the model with `Hist=TRUE` as in the above example and examining the generated values to ensure the parameters you have added in `OM@cpars` are being used as expected.

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 28

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.

28.1 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
```

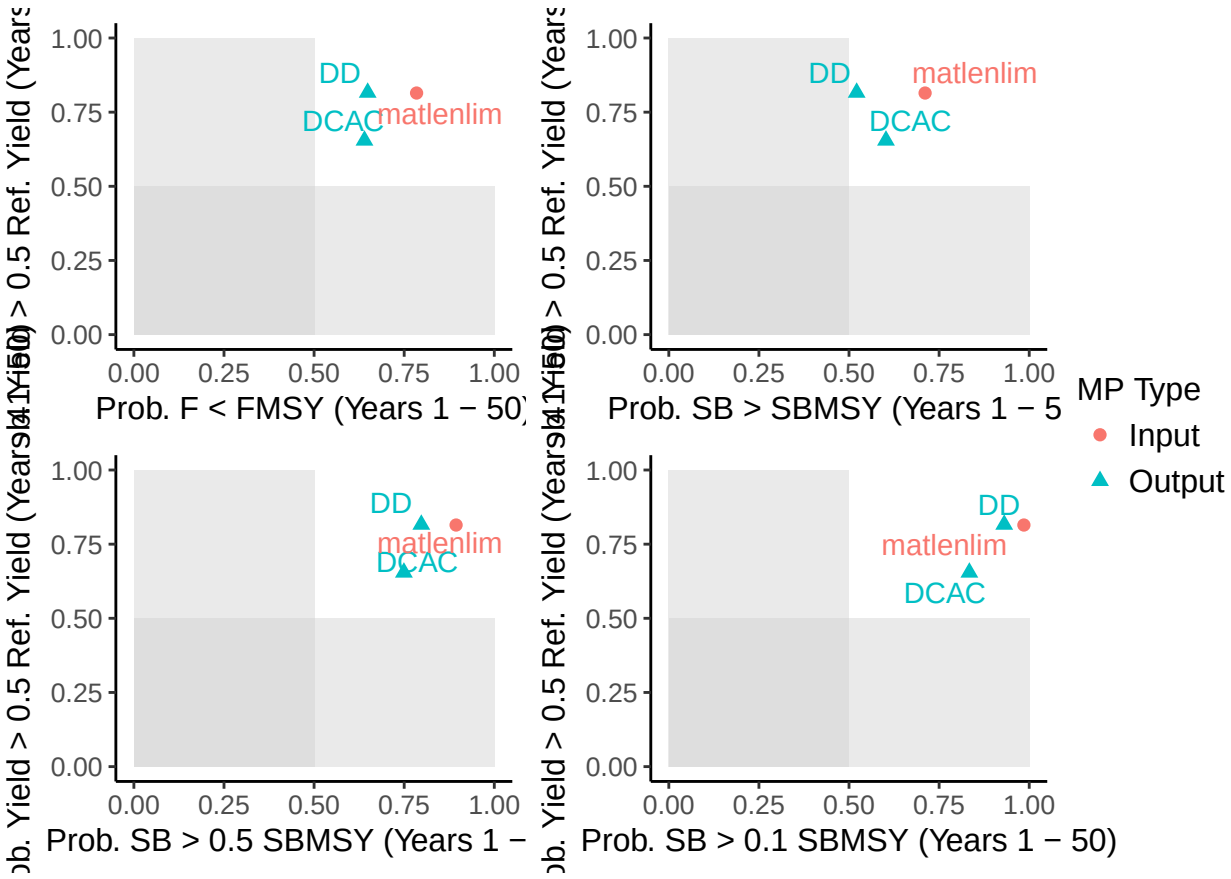
```
##                                       Performance.Metrics
## 1                 Probability of not overfishing (F<FMSY)
## 2                      Spawning Biomass relative to SBMSY
## 3        Average Annual Variability in Yield (Years 1-50)
## 4 Average Yield relative to Reference Yield (Years 41-50)
##
## 1              Prob. F < FMSY (Years 1 - 50)
## 2        Prob. SB > 0.5 SBMSY (Years 1 - 50)
## 3              Prob. AAVY < 20% (Years 1-50)
## 4 Prob. Yield > 0.5 Ref. Yield (Years 41-50)
##
##
## Probability:
##          MP PNOF  P50 AAVY  LTY
## 1    Fratio 0.61 0.69 0.78 0.54
## 2      DCAC 0.64 0.75 0.89 0.65
## 3      Fdem 0.52 0.61 0.76 0.50
## 4        DD 0.65 0.80 0.94 0.82
## 5 matlenlim 0.78 0.89 0.34 0.81
```

```
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)
```




```
##          MP PNOF  LTY P100  P50  P10 Satisificed
## 1      DCAC 0.64 0.65 0.60 0.75 0.83        TRUE
## 2        DD 0.65 0.82 0.52 0.80 0.93        TRUE
## 3 matlenlim 0.78 0.81 0.71 0.89 0.99        TRUE
```

28.2 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 29

Custom Performance Metrics

PM methods were introduced in the Performance Metrics Methods chapter. DLMtool includes several built-in PM methods:

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

```
##  [1] "AAVY"  "LTY"   "P10"   "P100"  "P50"   "PNOF"  "STY"   "Yield"
##  [9] "MeanB" "MeanF"
```

We saw in Customizing the PM Functions that it is straightforward to modify the years that the performance statistics are calculated over using the Yrs argument, and the reference level using the Ref argument. In this section we describe the PM functions in more detail for advanced users who wish to develop their own PM methods.

We will demonstrate this using the P50 performance metric function as an example.

29.1 Necessity of Complexity

While at first glance the PM functions may appear overly complicated, they are actually quite straightforward. This level of complexity is required because functions of class PM not only return the required performance statistic, but also relevant contextual information.

Imagine we want to develop a function to calculate the probability that $B > 0.5B_{\mathrm{MSY}}$, which we will define as *not overfished*. We could do something like this:

```
P_Noverfished <- function(MSEobj) {
  round(apply(MSEobj@B_BMSY >0.5, 2, mean),2)
}
```

We could increase the flexibility of the function by adding additional arguments to specify the years to calculate the statistic over, but for this demonstration we will keep it simple.

Applying our function to calculate the performance statistic is straightforward:

```
MSEobj <- runMSE(silent=TRUE) # run an exampe MSE
p_noverfish <- P_Noverfished(MSEobj) # calculate and store our performance statistic
p_noverfish
```

```
## [1] 0.68 0.78 0.93 0.80 0.99 0.85
```

The p_noverfish variable now contains our performance statistic, which we could use to plot or tabulate results. However, the numeric vector returned by P_Noverfished includes no information on what these

numbers represent. This makes it difficult to include the results in generic plotting or summary functions. Furthermore, it is easy to imagine that with several similiar performance metric functions, it will be easy to loose track of the meaning of the results unless an elaborate variable naming system is used.

The `PM` functions have been designed to avoid this problem by returning all information related to the calculation of the performance statistic. Here we calculate the same performance metric using the relevant `PM` function:

```
p1 <- P50(MSEobj)
```

Note that we don't need to use a descriptive variable name; the `p1` variable includes relevant information on the performance metric. Note also that `PM` function are most often passed directly to plotting or table functions instead of being called and stored directly.

The `p1` variable is an object of class PMobj and includes the following slots:

```
slotNames(p1)
```

```
## [1] "Name"    "Caption" "Stat"    "Ref"     "Prob"    "Mean"    "MPs"
```

The first two slots contain a descriptive name of the performance statistic, and a caption that can be used in plots or summary tables:

```
p1@Name
```

```
## [1] "Spawning Biomass relative to SBMSY"
```

```
p1@Caption
```

```
## [1] "Prob. SB > 0.5 SBMSY (Years 1 - 50)"
```

The last two slots contain the results of performance statistic and the names of the MPs:

```
p1@Mean
```

```
## [1] 0.6829167 0.7845833 0.9287500 0.7958333 0.9933333 0.8512500
```

```
p1@MPs
```

```
## [1] "AvC"       "DCAC"      "FMSYref"   "curE"      "matlenlim" "MRreal"
```

We will look at the other 3 slots in detail in the next section. The main point here is to demonstrate the using functions of class `PM` that return objects of class `PMobj` has the advantage that the function output is completely self-contained and can be used elsewhere without requiring any additional information.

29.2 `PM` Methods in Detail

Let's go through the `P50` function in detail to see how it works:

```
P50
```

```
## function (MSEobj = NULL, Ref = 0.5, Yrs = NULL)
## {
##     Yrs <- ChkYrs(Yrs, MSEobj)
##     PMobj <- new("PMobj")
##     PMobj@Name <- "Spawning Biomass relative to SBMSY"
##     if (Ref != 1) {
##         PMobj@Caption <- paste0("Prob. SB > ", Ref, " SBMSY (Years ",
##             Yrs[1], " - ", Yrs[2], ")")
##     }
##     else {
```

```
##         PMobj@Caption <- paste0("Prob. SB > SBMSY (Years ", Yrs[1],
##             " - ", Yrs[2], ")")
##     }
##     PMobj@Ref <- Ref
##     PMobj@Stat <- MSEobj@B_BMSY[, , Yrs[1]:Yrs[2]]
##     PMobj@Prob <- calcProb(PMobj@Stat > PMobj@Ref)
##     PMobj@Mean <- calcMean(PMobj@Prob)
##     PMobj@MPs <- MSEobj@MPs
##     PMobj
## }
## <bytecode: 0x0000000010e644c8>
## <environment: namespace:DLMtool>
## attr(,"class")
## [1] "PM"
```

Firstly, functions of class `PM` must have three arguments: `MSEobj`, `Ref`, and `Yrs`:

```
args(P50)
```

```
## function (MSEobj = NULL, Ref = 0.5, Yrs = NULL)
## NULL
```

1. The first argument `MSEobj` is obvious, an object of class `MSE` to calculate the performance statistic.
2. The second argument `Ref` must have a default value. This is used as reference for the performance statistic, and will be demonstrated shortly.
3. The third argument `Yrs` can have a default value of `NULL` or specify a numeric vector of length 2 with the first and last years to calculate the performance statistic, or a numeric vector of length 1 in which case if it is positive it is the first `Yrs` and if negative the last `Yrs` of the projection period.

The first line of a `PM` function must be `Yrs <- ChkYrs(Yrs, MSEobj)`. This line updates the `Yrs` variable and makes sure that the specified year indices are valid. For example:

```
ChkYrs(NULL, MSEobj) # returns all projection years
ChkYrs(c(1,10), MSEobj) # returns first 10 years
ChkYrs(c(60,80), MSEobj) # returns message and last 20 years
ChkYrs(5, MSEobj) # first 5 years
ChkYrs(-5, MSEobj) # last 5 years
ChkYrs(c(50,10), MSEobj) # returns an error
```

When the default value for `Yrs` is `NULL`, the `Yrs` variable is updated to include all projection years:

```
Yrs <- ChkYrs(NULL, MSEobj)
Yrs
```

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

Next we create a new object of class `PMobj`, and populate the `Name` slot with a short but descriptive name:

```
PMobj <- new("PMobj")
PMobj@Name <- "Spawning Biomass relative to SBMSY"
```

The next line populates the `Caption` slot with a brief caption including the years over which the performance statistic is calculated. The *if* statement is not crucial, but avoids the redundant `SB > 1 SBMSY` in cases where `Ref=1`.

Next we store the value of the `Ref` argument in the `PMobj@Ref` slot so that information is contained in the function output.

```
PMobj@Ref <- Ref
```

The `Stat` slot is an array that stores the variable which we wish to calculate the performance statistic; an output from the `runMSE` function with dimensions `MSE@nsim`, `MSE@nMPs`, and `MSE@proyears` (or fewer if the argument `Yrs != NULL`).

In this case we want to calculate a performance statistic related to the biomass relative to BMSY, and so we assign the `Stat` slot as follows:

```
PMobj@Stat <- MSEobj@B_BMSY[, , Yrs[1]:Yrs[2]]
```

Note that we are including all simulations and MPs and indexing the years specified in `Yrs`.

Next we use the `calcProb` function to calculate the mean of `PMobj@Stat > PMobj@Ref` over the years dimension. This results in a matrix with dimensions `MSE@nsim, MSE@nMPs`:

```
PMobj@Prob <- calcProb(PMobj@Stat > PMobj@Ref)
```

Note that in order to calculate a probability the argument to the `calcProb` function must be a logical array, which is achieved using the `Ref` slot.

Also note that in this case `PMobj@Stat > PMobj@Ref` is equivalent to `MSEobj@B_BMSY[, , Yrs[1]:Yrs[2]] > 0.5`. The `PM` functions have been designed this way so that in most cases the `PMobj@Prob <- calcProb(PMobj@Stat > PMobj@Ref)` line is identical in all `PM` functions and does not need to be modified. The exception to this is if we don't want to calculate a probability but want the actual mean values of `PMobj@Stat`, demonstrated in the example below.

In the next line we calculate the mean of `PMobj@Prob` over simulations using the `calcMean` function:

```
PMobj@Mean <- calcMean(PMobj@Prob)
```

Similiar to the previous line, this line is identical in all `PM` functions and can be simply copy/pasted from other `PM` functions without being modified. The `Mean` slot is a numeric vector of length `MSEobj@nMPs` with the overall performance statistic, in this case the probability of $B > 0.5B_{\text{MSY}}$ across all simulations and years.

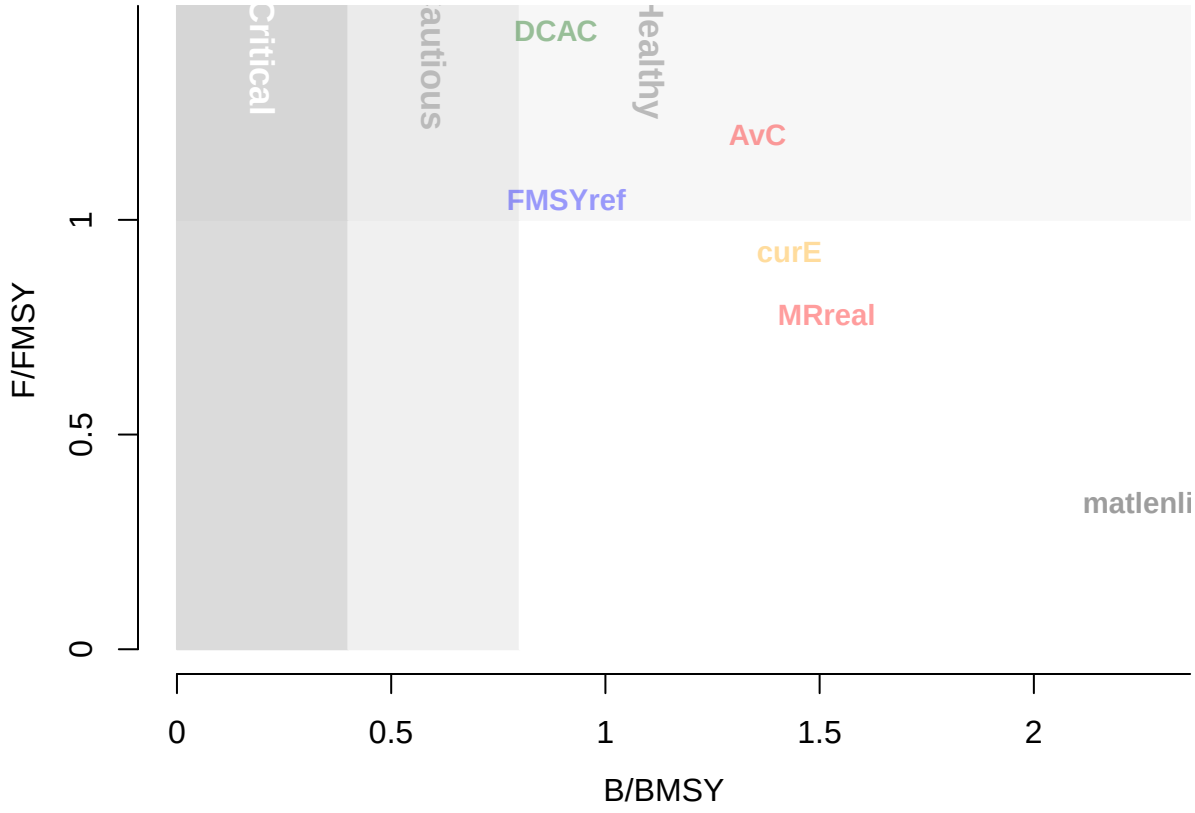
Finally, we store the names of the MPs and return the `PMobj`.

29.3 Creating Example `PM`s and Plot

As an example, we will create another version of `DFO_plot` using some custom `PM` functions and a customized version of `TradePlot`.

First we create the plot using `DFO_plot`:

```
DFO_plot(MSEobj)
```



From the help documentation (`?DFO_plot`) we can see that this function plots mean biomass relative to BMSY and fishing mortality rate relative to FMSY over the final 5 years of the projection.

First we'll develop a `PM` function to calculate the mean B/BMSY for the last 5 years of the projection period. Notice that this is very similiar to `P50` described above, with the modification of the `Caption` and the `Prob` slots, and the `Yrs` argument. We are calculating a mean here instead of a probability and are not using the `Ref` argument:

```
MeanB <- function(MSEobj = NULL, Ref = 1, Yrs = -5) {
    Yrs <- ChkYrs(Yrs, MSEobj)
    PMobj <- new("PMobj")
    PMobj@Name <- "Spawning Biomass relative to SBMSY"
    PMobj@Caption <- paste0("Mean SB/SBMSY (Years ", Yrs[1], " - ", Yrs[2], ")")

    PMobj@Ref <- Ref
    PMobj@Stat <- MSEobj@B_BMSY[, , Yrs[1]:Yrs[2]]
    PMobj@Prob <- calcProb(PMobj@Stat)
    PMobj@Mean <- calcMean(PMobj@Prob)
    PMobj@MPs <- MSEobj@MPs
    PMobj
}
```

We develop a `PM` function to calculate average F/FMSY in a similar way:

```
MeanF <- function(MSEobj = NULL, Ref = 1, Yrs = -5) {
    Yrs <- ChkYrs(Yrs, MSEobj)
    PMobj <- new("PMobj")
    PMobj@Name <- "Fishing Mortality relative to FMSY"
```

```
    PMobj@Caption <- paste0("Mean F/FMSY (Years ", Yrs[1], " - ", Yrs[2], ")")

    PMobj@Ref <- Ref
    PMobj@Stat <- MSEobj@F_FMSY[, , Yrs[1]:Yrs[2]]
    PMobj@Prob <- calcProb(PMobj@Stat)
    PMobj@Mean <- calcMean(PMobj@Prob)
    PMobj@MPs <- MSEobj@MPs
    PMobj
}
```

Similar to developing custom MPs we need to tell R that these new functions are `PM` methods:

```
class(MeanB) <- "PM"
class(MeanF) <- "PM"
```

Now we can test our performance metric functions:

```
data.frame(MP=MeanB(MSEobj)@MPs, B_BMSY=MeanB(MSEobj)@Mean, F_FMSY=MeanF(MSEobj)@Mean)
```

```
##          MP    B_BMSY    F_FMSY
## 1       AvC 1.3553162 1.1987906
## 2      DCAC 0.8845365 1.4408156
## 3   FMSYref 0.9090203 1.0466349
## 4      curE 1.4298469 0.9264642
## 5 matlenlim 2.2738462 0.3421662
## 6    MRreal 1.5169029 0.7801777
```

How do these results compare to what is shown in `DFO_plot`?

We could also use the `summary` function with our new `PM` functions, but note that these results are not probabilities:

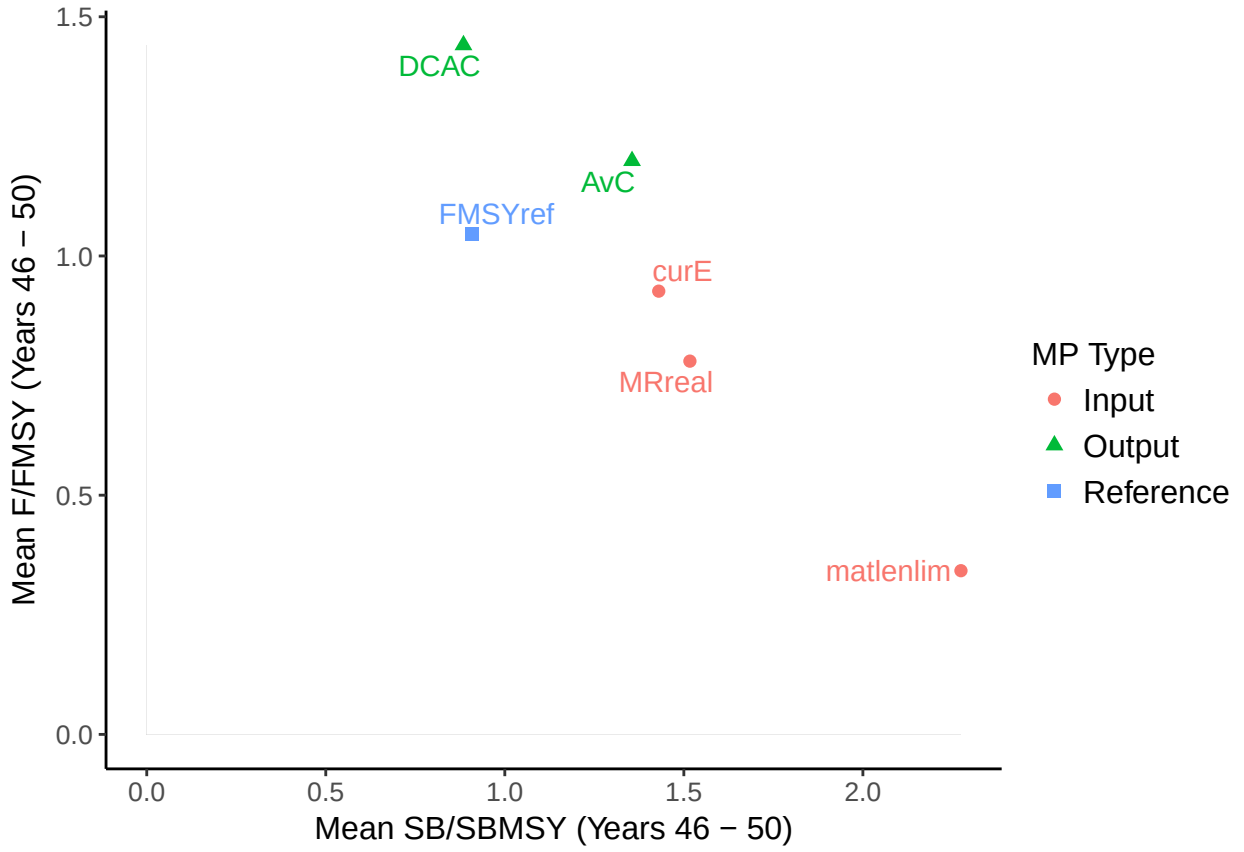
```
summary(MSEobj, 'MeanB', 'MeanF')
```

```
## Calculating Performance Metrics

##                   Performance.Metrics
## 1 Spawning Biomass relative to SBMSY  Mean SB/SBMSY (Years 46 - 50)
## 2 Fishing Mortality relative to FMSY    Mean F/FMSY (Years 46 - 50)
##
##
## Probability:
##          MP MeanB MeanF
## 1       AvC  1.40  1.20
## 2      DCAC  0.88  1.40
## 3   FMSYref  0.91  1.00
## 4      curE  1.40  0.93
## 5 matlenlim  2.30  0.34
## 6    MRreal  1.50  0.78
```

Finally, we will develop a customized plotting function to reproduce the image produced by `DFO_plot`.

We can produced something fairly similar quite quickly using the `TradePlot` function:

```
TradePlot(MSEobj, 'MeanB', 'MeanF', Lims=c(0,0))
```

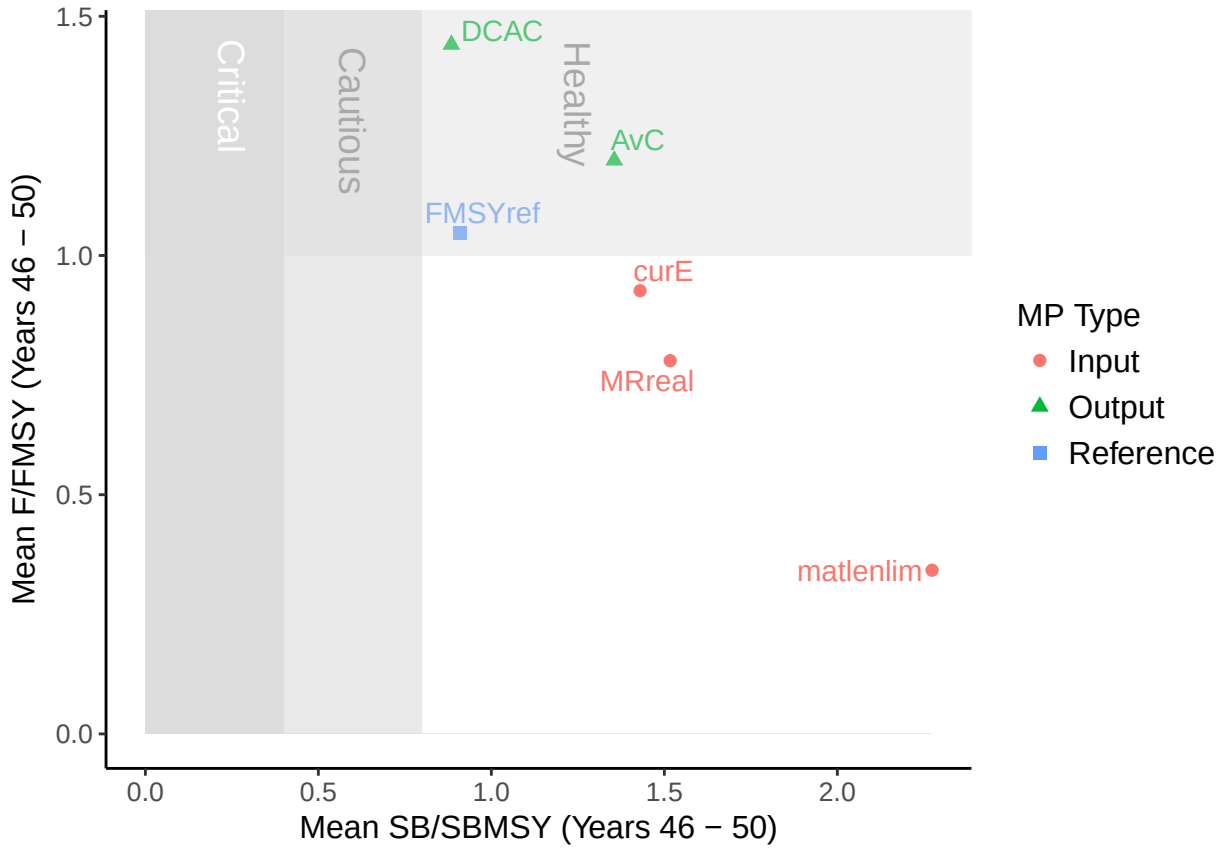



```
##          MP MeanB MeanF Satisificed
## 1       AvC  1.40  1.20        TRUE
## 2      DCAC  0.88  1.40        TRUE
## 3   FMSYref  0.91  1.00        TRUE
## 4      curE  1.40  0.93        TRUE
## 5 matlenlim  2.30  0.34        TRUE
## 6    MRreal  1.50  0.78        TRUE
```

Adding the shaded polygons and text requires a little more tweaking and some knowledge of the `ggplot2` package. We will wrap up our code in a function:

```
NewPlot <- function(MSEobj) {
  # create but don't show the plot
  P <- TradePlot(MSEobj, 'MeanB', 'MeanF', Lims=c(0,0), Show=FALSE)
  P1 <- P$Plots[[1]] # the ggplot objects are returned as a list

  # add the shaded regions and the text
  P1 + ggplot2::geom_rect(ggplot2::aes(xmin=c(0,0,0), xmax=c(0.4, 0.8, Inf),
                                       ymin=c(0,0,1), ymax=rep(Inf,3)),
                          alpha=c(0.8, 0.6, 0.4), fill="grey86") +
    ggplot2::annotate(geom = "text", x = c(0.25, 0.6, 1.25), y = Inf,
                      label = c("Critical", "Cautious", "Healthy") ,
                      color = c('white', 'darkgray', 'darkgray'),
                      size=5, angle = 270, hjust=-0.25)

}
```

```
NewPlot(MSEobj)
```



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.