# Casal2 R Library Functionality

*C.Marsh*

*02/02/2020*

## Introduction

The aim of this document is to describe and demonstrate functionality in the R library that is currently available. This includes both deterministic and esitmation runs, and MCMC output. This could also serve as a best practice guide. Remember that the R-library function can only intepret output that is available, and Casal2's reporting is completely user defined i.e. if you want to plot derived quantities in R there must be `@report` for derived quantities.

There are three types of outputs from a Casal2 model run.

- a single MPD run (`-r` or `-e`)
- a multirun MPD run `-r -i multi_par_file.out`, `-e -i multi_par_file.out`, `-p` or `-s 10`
- MCMC run `-m`

We will demonstrate the use of R libraries for all three of these outputs. All files that are used in this demonstration are available in the `extdata` file in the `casal2` R package, from the path below.

```
library(casal2)
fpath <- system.file("extdata", package="casal2")
```

## Single Run

When Casal2 is either run deterministically (`casal2 -r`) or as an estimation run (`casal2 -e`), there are a range of options available for plotting important derived quantities and parameter estimates of a model.

```
# this will create a list like object called single_run
single_run = extract.mpd(file = "estimate.out", path = fpath)
# look at what objects are in the list
names(single_run)
```

```
## [1] "Init" "Sep_Feb" "Mar_May"
## [4] "Jun_Aug" "summary" "objective"
## [7] "Rec" "Mortality" "SSB"
## [10] "obs_tan" "tan_at_age" "eastF_at_age"
## [13] "westF_at_age" "eastFSel" "chatTANSel"
## [16] "westFSel" "warnings_encounted"
```

If this is an MPD run the, first two things you want to view are warnings and the minimisers convergence. These will always be reported for an estimation.

```
# number of warnings encountered
single_run$warnings_encounted$warnings_found
```

```
## [1] 2
```

```
# An example of a warning
single_run$warnings_encounted$warning_0
```

```
## [1] "estimated parameter
```

```
'process[Recruitment].ycs_values{2010}' was within 0.001 of
upper bound 3"
```

```
# Did the model converge
single_run$minimiser_result$Result
```

```
## NULL
```

```
# What was the reason for this result?
single_run$minimiser_result$Message
```

```
## NULL
```

See also the section on checking convergence for estimation below for helping with diagnosing MPD convergence. Once you are satisfied with a model run convergence, you will want to look at the goodness of fit to data. This is best done by looking at residuals.

Plot observed vs expected values for fits

```
plot.fits(model = single_run, report_label = "obs_tan", plot.it = T)
```

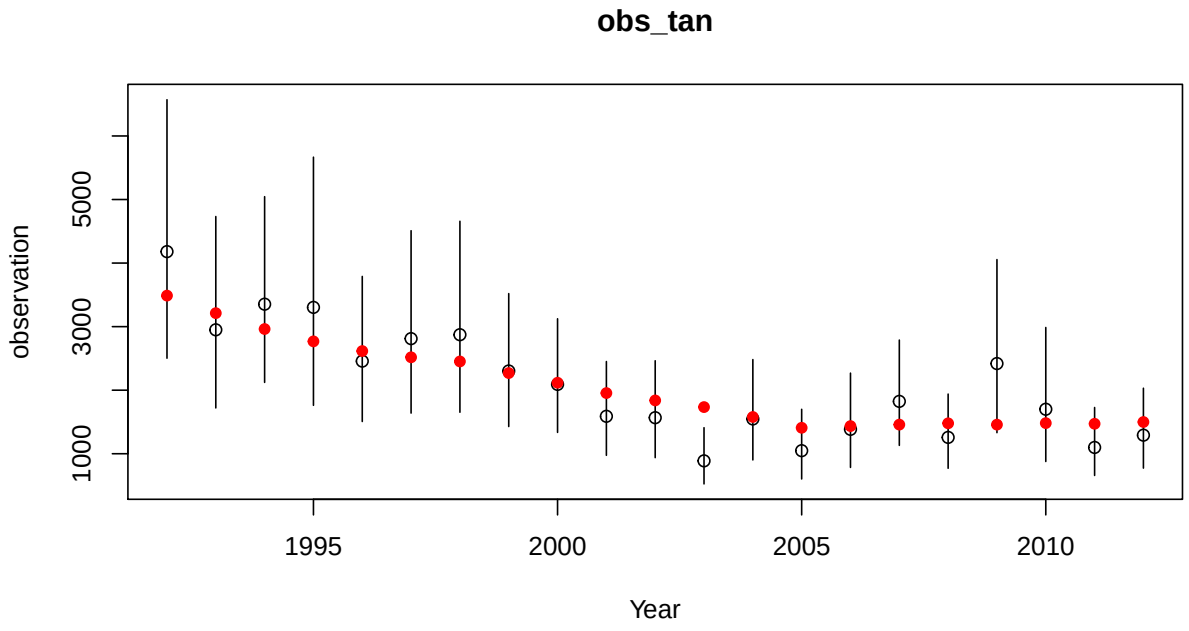



Figure 1: Observed (black) with plus or minus two standard errors, with the models fitted value (red)

```
plot.fits(model = single_run, report_label = "westF_at_age", plot.it = T)
```

```
## Plotting mean age.
```

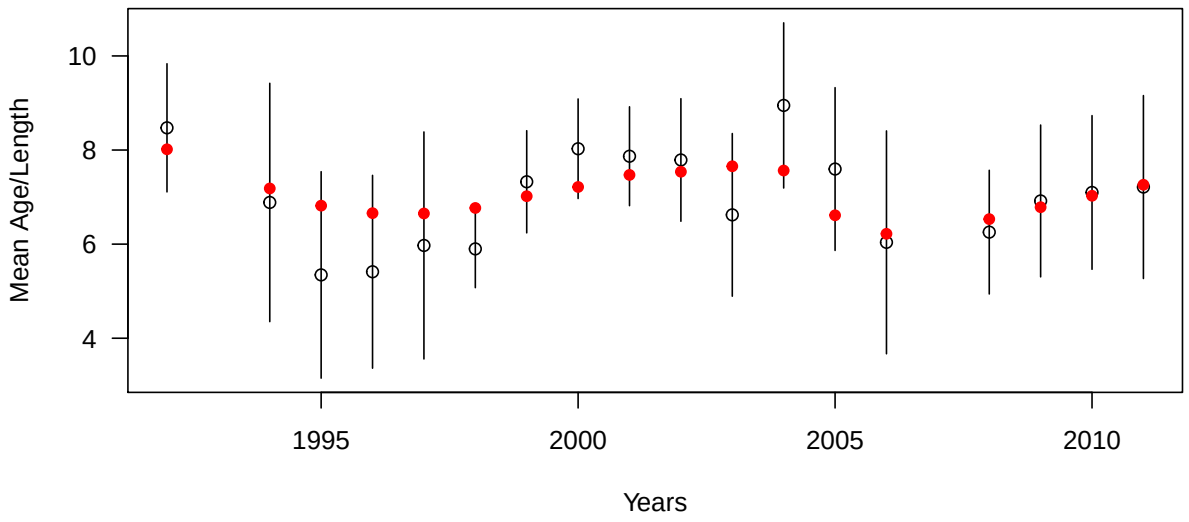



Figure 2: Observed (black) with plus or minus two standard errors, with the models fitted value (red)

```
plot.fits(model = single_run, report_label = "eastF_at_age", plot.it = T)
```

```
## Plotting mean age.
```

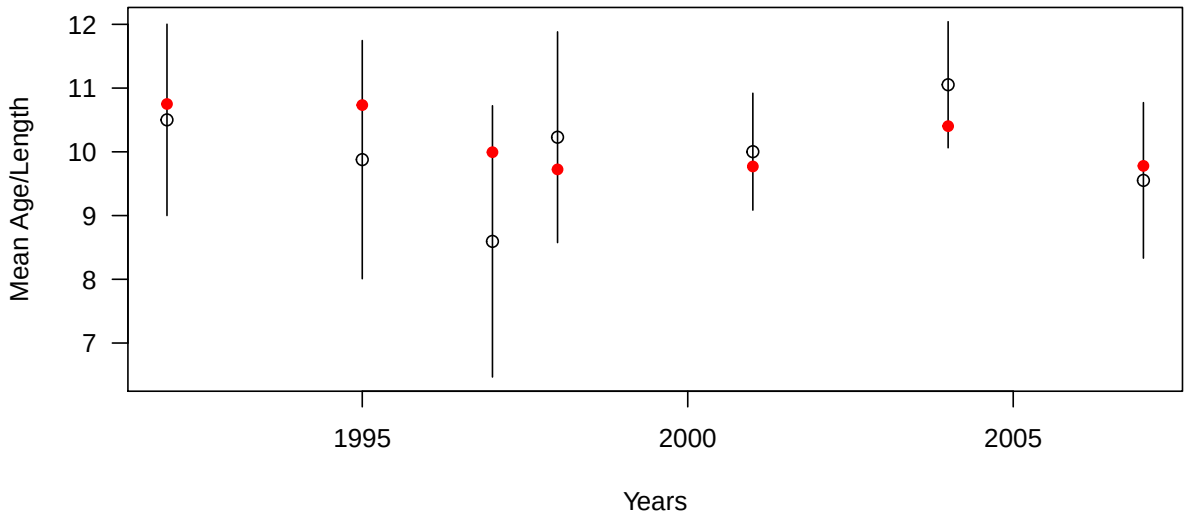



Figure 3: Observed (black) with plus or minus two standard errors, with the models fitted value (red)

If you have requested Casal2 to report either normalised or pearsons residuals for any of the obsevation's you should also plot them against the theoretical standard normal quantiles.

```
qqnorm(single_run$obs_tan$Values$normalised_residuals, pch = 1, frame = FALSE)
qqline(single_run$obs_tan$Values$normalised_residuals, col = "steelblue", lwd = 2)
```

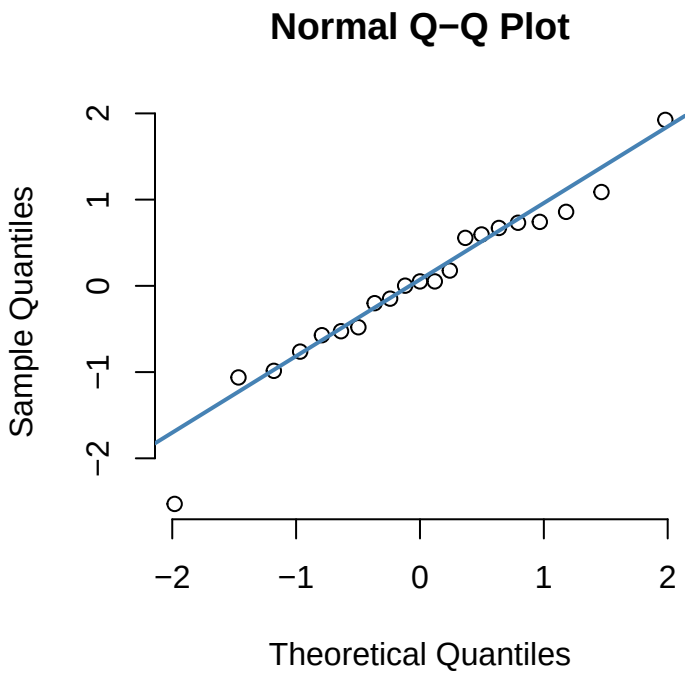



Figure 4: Sample residuals vs theoretical residuals for the biomass observation

This can also be done for multinomial pearsons residuals, with them and both normalised residuals $\mathcal{N}(0,1)$

Once you are satisfied with the goodness of fit of the model, then you will want to look at derived quantities and parameters. The R library can do a very basic plot using the '$plot.derived_quantities()$' function.

```
plot.derived_quantities(model = single_run, report_label = "SSB", plot.it = T)
```

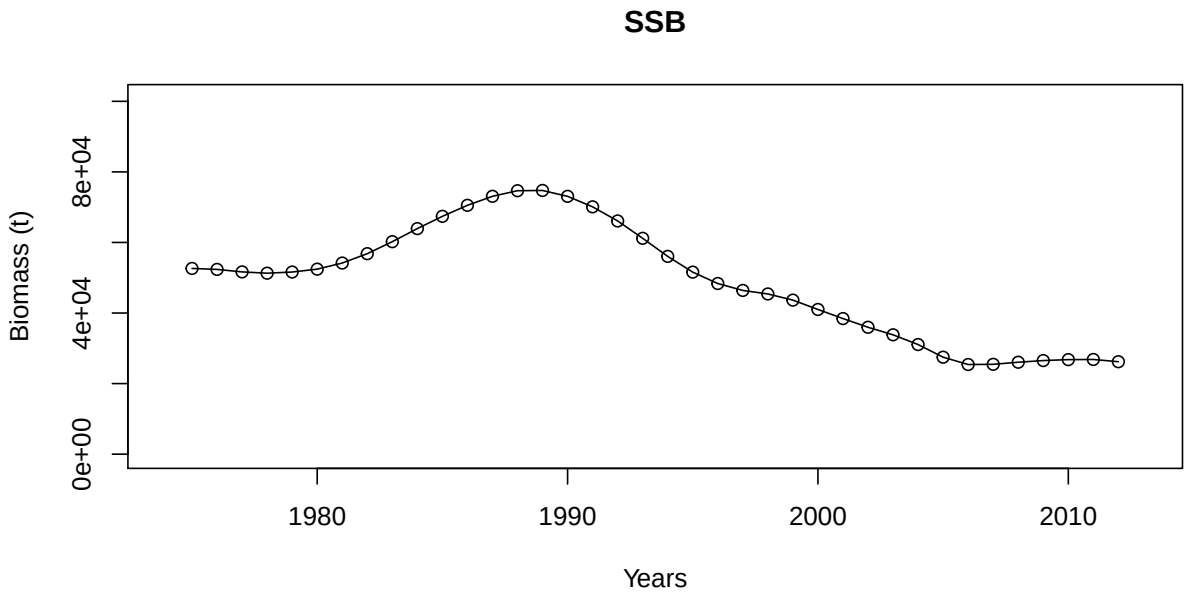



Figure 5: Estimate SSB

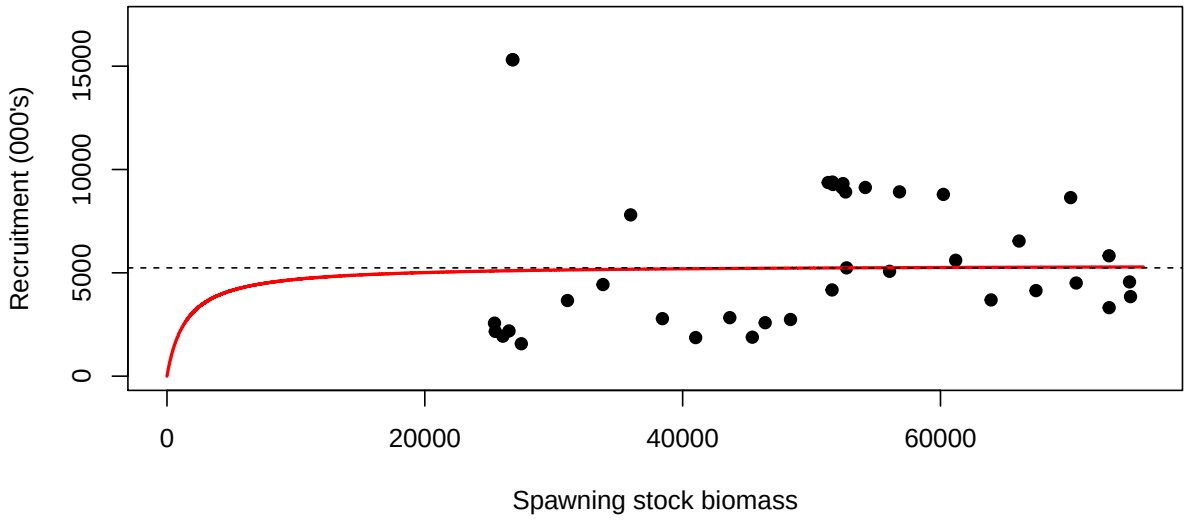
You can also, get the function to just return the SSBs, by setting the input `plot.it = F`, and then create you own plot.

```
SSBs = plot.derived_quantities(model = single_run, report_label = "SSB", plot.it = F)
head(SSBs)
```

```
##          SSB
## 1975 52639.6
## 1976 52364.3
## 1977 51655.2
## 1978 51289.1
## 1979 51619.7
## 1980 52434.6
```

The other derived quantities are selectivities,

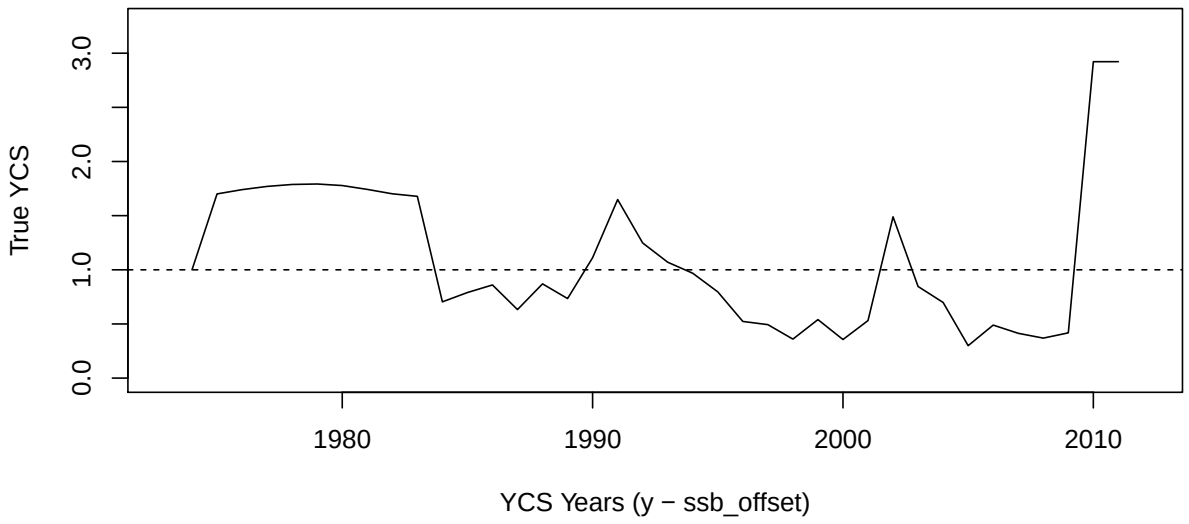
```
plot.recruitment(model = single_run, report_label = "Rec",add_BH_curve = TRUE)
```




Also YCS parameters

```
plot.ycs(model = single_run, report_label = "Rec")
```

```
## [1] "single iteration report found"
```




Fishing pressure/exploitation rates for each fishery

```
plot.pressure(model = single_run, report_label = "Mortality", plot.it = T,
              col = c("blue", "red"), lwd = 3)
```

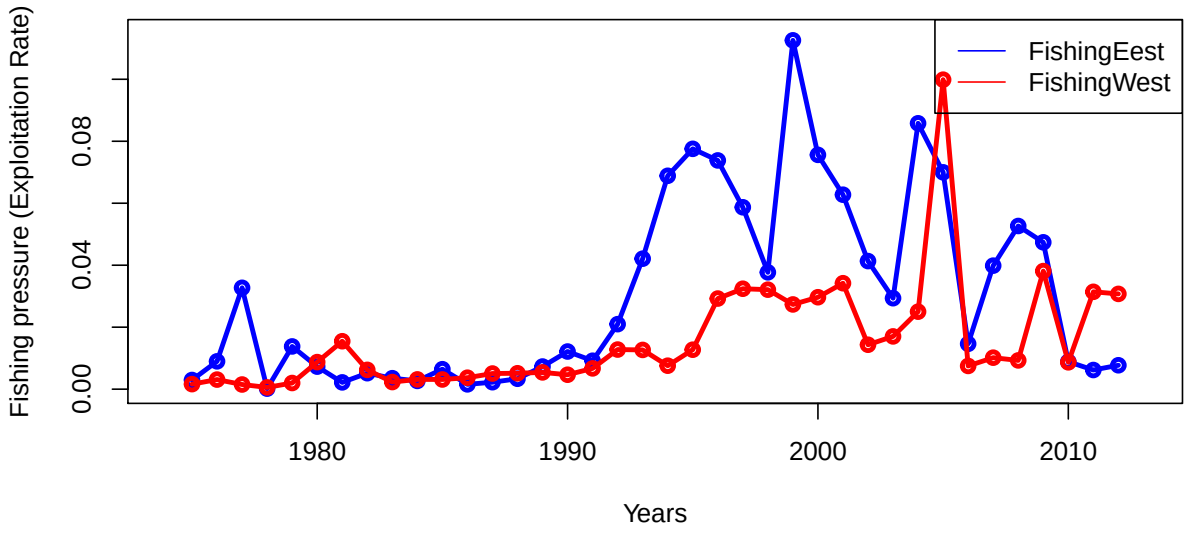



Figure 6: Estimated exploitation

plot selectivities

```
plot.selectivities(model = single_run, report_labels = c("eastFSel", "chatTANSel", "westFSel"), col = c
```

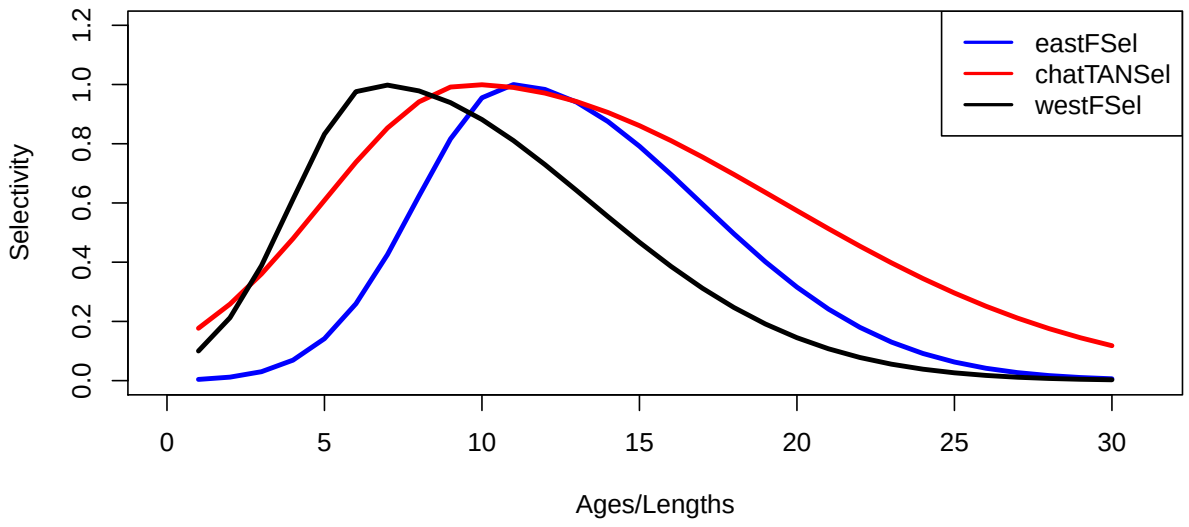
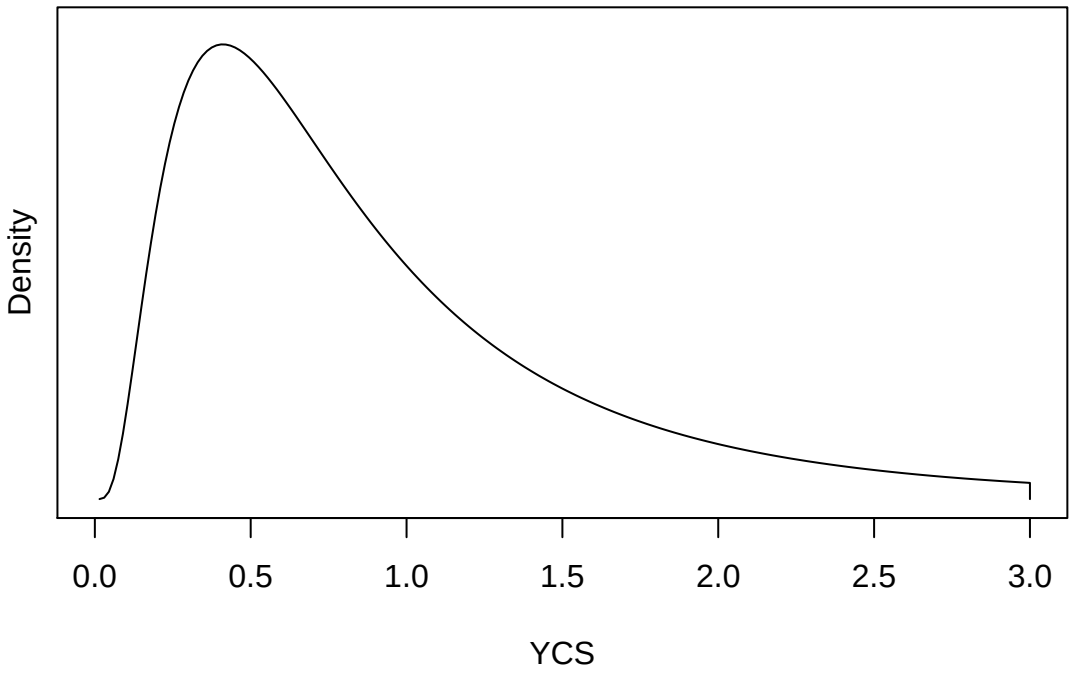



Figure 7: Estimated Selectivities

Next, we will show how to check that the minimum reported is reached with multiple parameter values. ## Plotting priors It is usefult to visulise priors, that are assumed in a Casal2 assessment model. We have added the `plot.prior()` function from the casal R package so this can be easily done.

```
plot.prior(type = "lognormal", mu = 1, cv = 0.9, xlim = c(0,3), xlab = "YCS", ylim = c(0,1), bounds = c
```

```
## Plotting lognormal prior
```




## Dataweighting

Casal2 has inbuild data weighting functions for relative index of abundance or biomass with `CV.for.CPUE` function. This function generates cv's for a relative biomass trend based on loess smoothness. There is also the Francis Method TA1.8 (Francis 2011) `Method.TA1.8()` demonstrated below.

```
Method.TA1.8(model = single_run, observation_labels = c("tan_at_age"), plot.it = T)
```


```
## [1] 1.181672
```

There are also wrapper functions that can help you automate the dataweightng procedure.

```
ModelFactor <- Method.TA1.8(single_run, observation_labels = c("eastF_at_age"))
## make a back-up copy of the original Observation.csl2 before running this section
## incase you mess it up or wipe it. This function will also strip out all comments
## which you may want to keep.
this_path = getwd();
temp_dir = tempdir();
## create a temp directory to undertake data-weighting
dir.create( file.path(temp_dir, "Simple"))
file.copy(from = file.path(fpath, "Simple", list.files(file.path(fpath, "Simple"))),
          to = file.path(temp_dir, "Simple"), recursive = T)
```

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

```
setwd(file.path(temp_dir,"Simple"))

while(abs(ModelFactor - 1.0) > 0.01) {
  if(.Platform$OS.type == "windows") {

      shell("casal2 -e > estimate.log 2> log.out")

  } else {
    # assumes linux
    system("casal2 -e > estimate.log 2> log.out")
  }
```

```
    new_mpd <- extract.mpd(file = "estimate.log")
    ModelFactor <- Method.TA1.8(new_mpd, observation_labels = c("eastF_at_age"))
    apply.dataweighting.to.csl2(weighting_factor = ModelFactor,
                                Observation_csl2_file = "observation.csl2",
                                Observation_out_filename = "observation.csl2",
                                Observation_label = c("chatOBSest"))
    print(ModelFactor)
}
```

```
## [1] "The 'csl' input parameter file has 6 commands, and
192 lines"
## [1] "chatTANq"
## [1] "chatTANbiomass"
## [1] "chatTANage"
## [1] "chatOBSwst"
## [1] "chatOBSest"
## [1] "Normal_ageing"
## [1] 1.168137
## [1] "The 'csl' input parameter file has 6 commands, and
192 lines"
## [1] "chatTANq"
## [1] "chatTANbiomass"
## [1] "chatTANage"
## [1] "chatOBSwst"
## [1] "chatOBSest"
## [1] "Normal_ageing"
## [1] 1.000688
```

```
## overwrite the re-weighed observation csl and chang back wd
setwd(this_path)
```

## Assessing MPD convergence

If setting up a model for estimating the *Maximum Posterior Density* (MPD), an important consideration is whether the model has been estimated at a local or global minmum. One method available is doing multiple estimations from different starting values. One function available for generating multiple starting values is the `generate.starting.pars()` function. This function takes a text configuration file, and searches for `@estimate` blocks to find prior and lower and upper bounds to simulate values for.

```
generate.starting.pars(path = fpath, Estimation_csl2_file = "Estimation.csl2",N = 10,
                       par_file_name = "random_start.out")
```

This function will create the file '$random\_start.out$' and format the values so they are compatible with '$casal2 -e -i random_start.out > multi_start_mpd.out$' format. This will create a multi run output which is given below for an example of using

```
# this will create a list like object called single_run
multi_run = extract.mpd(file = "multi_start_mpd.out", path = fpath)
```

```
## loading a run from -i format
```

```
# look at what objects are in the list
names(multi_run)
```

```
## [1] "Init" "Sep_Feb" "Mar_May"
```

```
## [4] "Jun_Aug" "summary" "objective"
## [7] "Rec" "Mortality" "SSB"
## [10] "obs_tan" "tan_at_age" "eastF_at_age"
## [13] "westF_at_age" "minimiser_result"
"warnings_encounted"
```

```
n_runs = length(multi_run$Init)
objective = vector()
convergence = vector()

for (i in 1:n_runs) {
  objective[i] = multi_run$objective[[i]]$values["total_score"]
  convergence[i] = multi_run$minimiser_result[[i]]$Result
}
convergence
```

```
## [1] "Success" "Success" "Success" "Failed" "Failed"
"Success" "Failed"
## [8] "Success" "Failed" "Failed"
```

```
objective[convergence == "Success"]
```

```
## [1] 683.638 683.638 683.638 683.638 829.424
```

Another helpful function that is available to identify parameters that are unidentfiable is '$check_mpd_identifiability()$'. You will need to report the covariance matrix and the estimate values in order for this function to work.

```
# Give this is an example file, the covariance is not invertable
# check_mpd_identifiability(single_run)
```

## Multi Run

Most of the functions that work for the Single MPD aren't written for multi run reports =(. The next step will be if there are multiple model configurations, say model_BH and model_constant, you would create sensitivities for these. This will be the most common multi model run the users will want to summarise

```
#model_bH = extract.mpd(file = "BevertonHoltRecruitment.out", path = fpath)
#model_const = extract.mpd(file = "ConstantRecruitment", path = fpath)
#multi_mpds = list(model_bH, model_const)
```

## MCMC

There is two formats for MCMC outputs, the objectives and samples, and the derived quantitites. The first are created with a `casal2 -m` command, and the latter is created via `casal2 -r -tabular -i mcmc_samples.out`

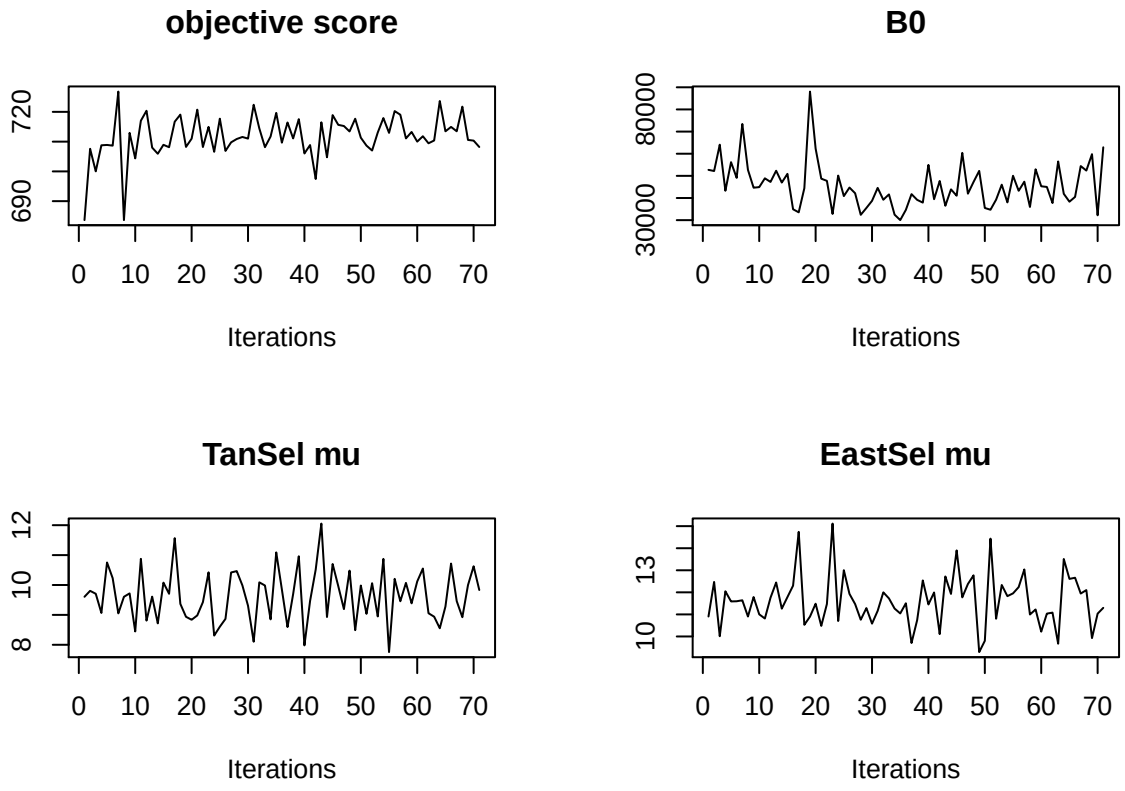
```
# this will create a list like object called single_run
mcmc_out = extract.mcmc(samples.file = "mcmc_samples.out",objectives.file =
            "mcmc_objectives.out", return_covariance = T, path = fpath)
# look at the covariance, aka proposal covariance for the MH MCMC
mcmc_out$Covariance
```

Because we don't want to re-invent the well, the next thing we want to do is bring in a thirdparty library i.e. `coda`.

```
library(coda) # TODO make this a dependency
mcmc_out = extract.mcmc(samples.file = "mcmc_samples.out",objectives.file =
            "mcmc_objectives.out", return_covariance = T, path = fpath)
# convert to coda mcmc object
# drop out sample jacobians, step_size, acceptance_rate, acceptance_rate_since_adapt
mcmc_chain = as.mcmc(mcmc_out$Data[, !colnames(mcmc_out$Data) %in%
     c("sample", "jacobians","step_size", "acceptance_rate",
     "penalties", "acceptance_rate_since_adapt")])
# if return_covariance = F, replace the above with below
#mcmc_chain = as.mcmc(mcmc_out)
```

Now use all the diagnostics for proper Bayesian evaluation

```
# once this occurs you can use all the Coda MCMC code
#geweke.diag(mcmc_chain)
par(mfrow = c(2,2))
traceplot(mcmc_chain[,"objective_score"], main = "objective score")
traceplot(mcmc_chain[,"process[Recruitment].b0"], main = "B0")
traceplot(mcmc_chain[,"selectivity[chatTANSel].mu"], main = "TanSel mu")
traceplot(mcmc_chain[,"selectivity[eastFSel].mu"], main = "EastSel mu")
```




## Using R to read and write Casal2 configuration files for simulation/model exploration

Functions that exist `write.csl2.file()` and `read.csl2.file()`

## TroubleShooting The R package

### fileEncoding

Since Windows 10, most people will run Casal2 out of the shell, this outputs in UTF-16 compared to the cmd prompt which writes text files in UTF-8. The R-package is build around UTF-8 but has parameters to read files of formate UTF-16. If you get the following error below when using one of the `extract()` functions

```
#  Read 1 item
#  Warning messages:
#  1: In scan(filename, what = "", sep = "\n", fileEncoding = fileEncoding) :
#  embedded nul(s) found in input
#  2: In extract.mpd(file = "results.txt", fileEncoding = "") :
#  File is empty, no reports found
```

You may be able to resolve this issue by using an alternative UTF format by specifying this format with the `fileEncoding` parameter

```
# MyOutput <- extract.mpd(... , fileEncoding = "UTF-16")
```

Francis, RIC Chris. 2011. "Data Weighting in Statistical Fisheries Stock Assessment Models." *Canadian Journal of Fisheries and Aquatic Sciences* 68 (6). NRC Research Press: 1124–38.