Casal2 R Package Functionality

C.Marsh

02/02/2020

Introduction

This document describes and demonstrates the functionality in the Casal2 R package. This includes parsing output from deterministic and estimation *Maximum Posterior Density* (MPD) runs, and MCMC runs. This could also serve as a best practices guide.

The R package functions can interpret existing output only: Casal2's reports are user-specified, e.g., to plot derived quantities in R, the `@report` for derived quantities must be defined in the model configuration files.

There are three types of output 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`
- a MCMC run `-m`

This document will demonstrate the use of the Casal2 R package for all three of these outputs. All files that are used in this demonstration are in the `extdata` file in the `casal2` R package.

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

Single Run

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

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

# output the objects 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 warnings and the minimiser's convergence status can be viewed. This information will always be output for an estimation run.

```
# 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"
```

```
# Model convergence status
single_run$minimiser_result$Result
```

```
## NULL
```

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

```
## NULL
```

See also the section below on checking convergence for estimation for diagnosing MPD convergence. Once a model run has converged, look at the goodness of fit to the data. This can be done by looking at residuals.

Plot observed vs expected values for fits

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

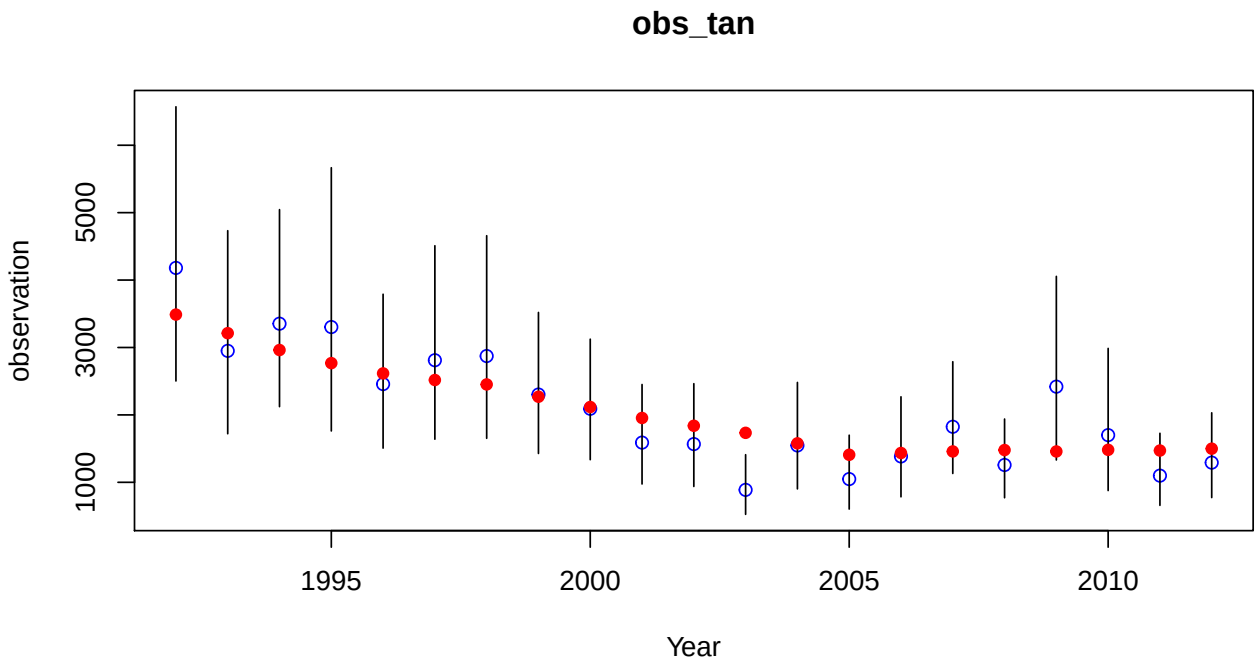


Figure 1: Observed (black) with plus or minus two standard errors, with the models fitted value (red)

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

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

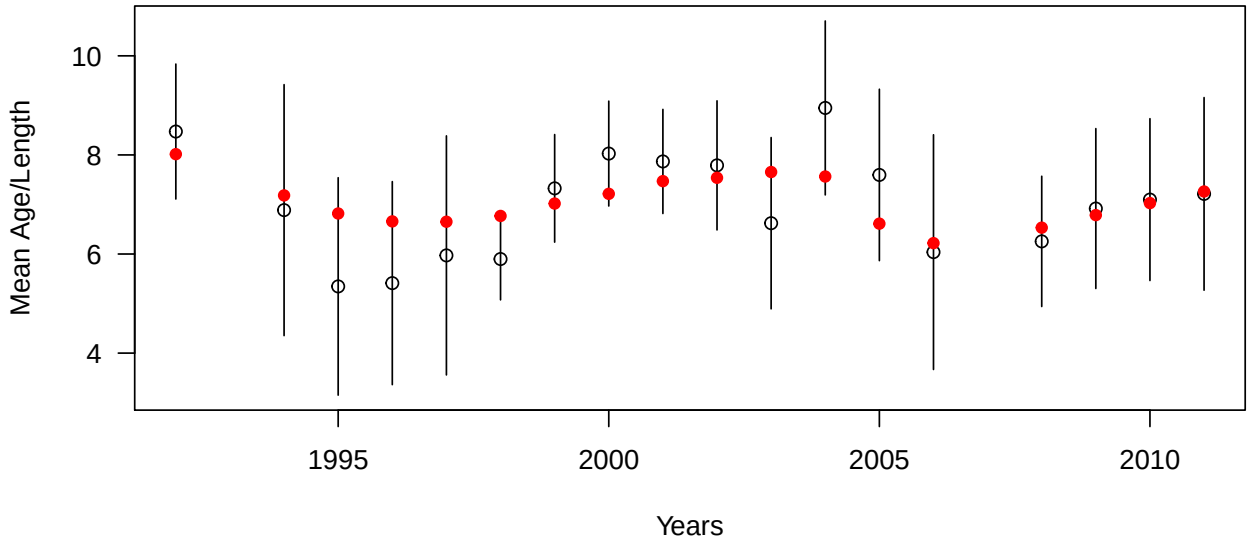


Figure 2: Observed (black) with plus or minus two standard errors, with the models fitted value (red)

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

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

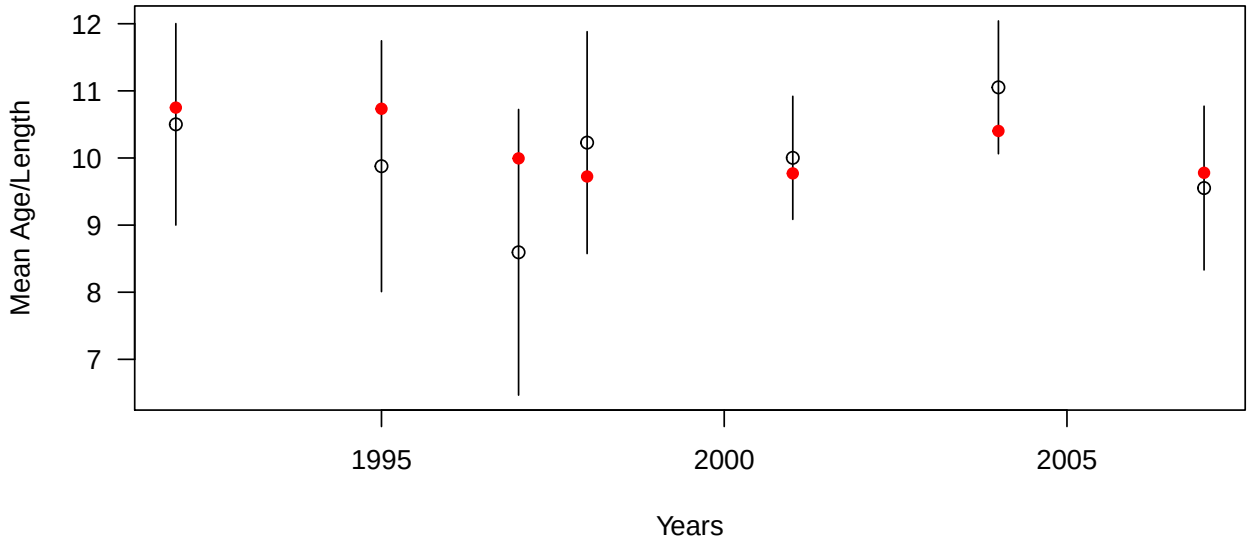


Figure 3: Observed (black) with plus or minus two standard errors, with the models fitted value (red)

If the Casal2 model configuration reports specify either normalised or Pearson's residuals for any of the obsevations, the residuals can be plotted 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)
```

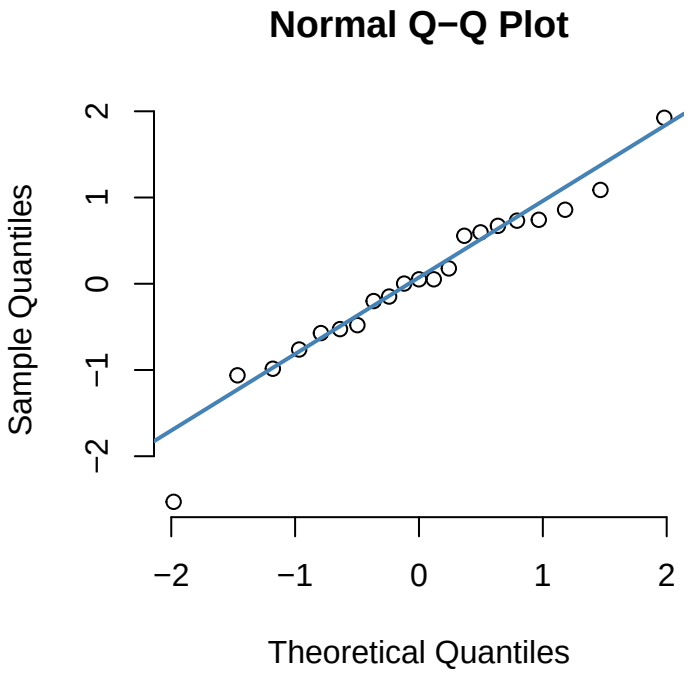


Figure 4: Sample residuals vs theoretical residuals for the biomass observation

This can also be done for multinomial Pearson's residuals, as both normalised and Pearson's residuals have distribution $\mathcal{N}(0, 1)$

Once the model fits reasonably to the data, look at the derived quantities and parameter estimates. The Casal2 R package can plot these using the function `plot.derived_quantities()`.

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

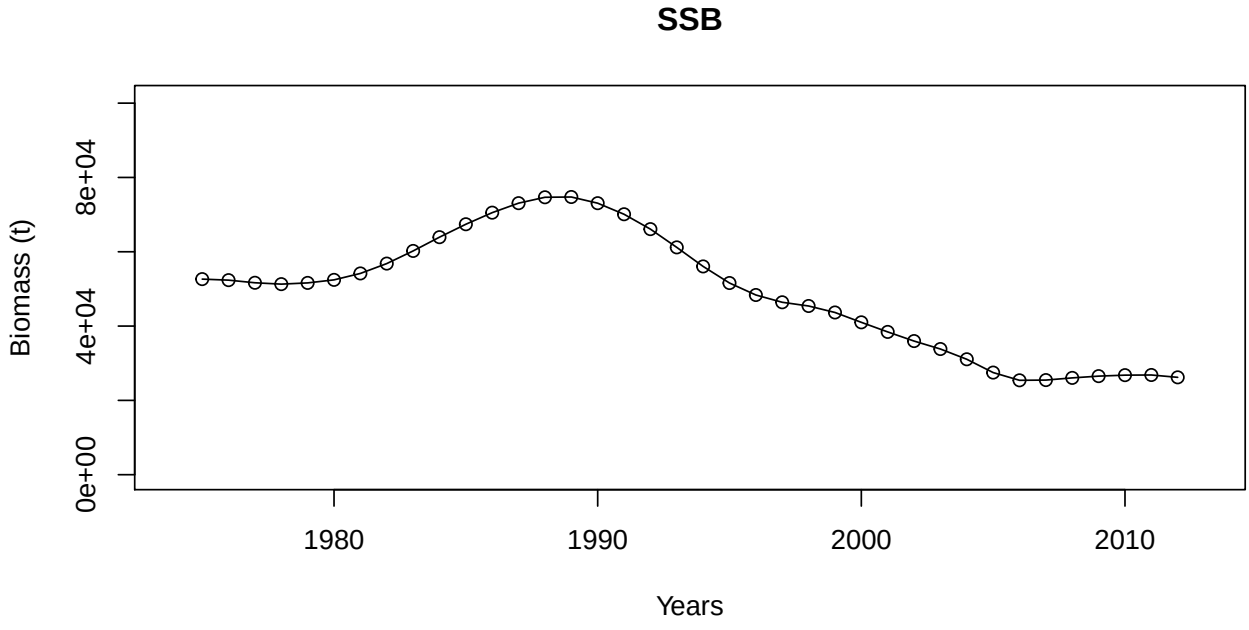


Figure 5: Estimate SSB

The function can return the SSBs only by setting the input `plot.it = F`, and then creating another 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
```

Other derived quantities include recruitment

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

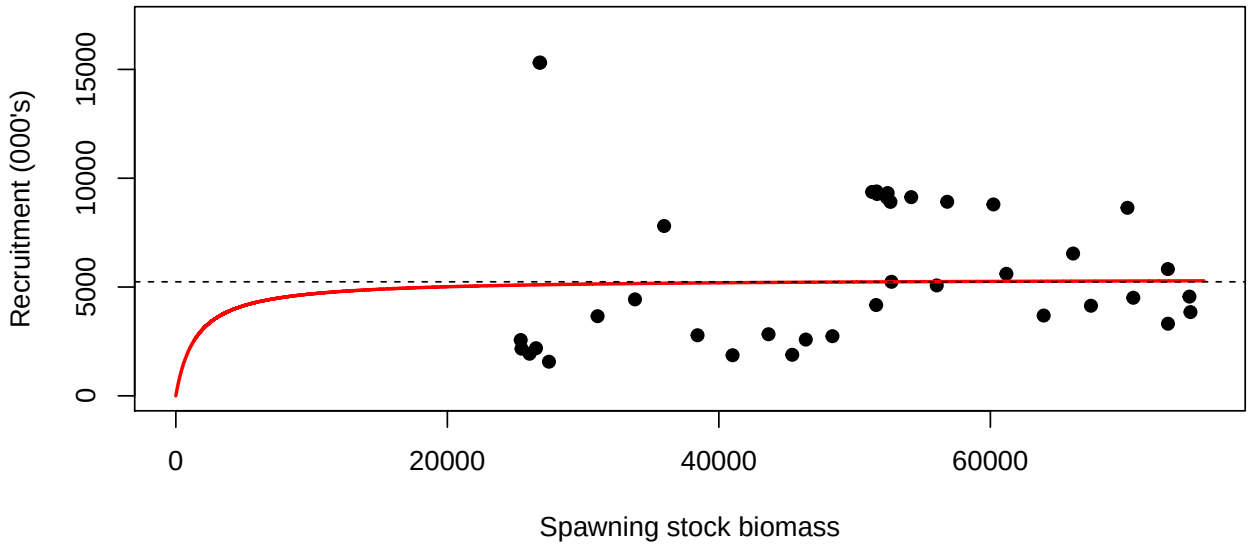


Figure 6: Estimated Recruitment (black dots), with the Beverton-Holt stock-recruitment curve (red) and R_0 (black horizontal dashed line)

YCS values

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

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

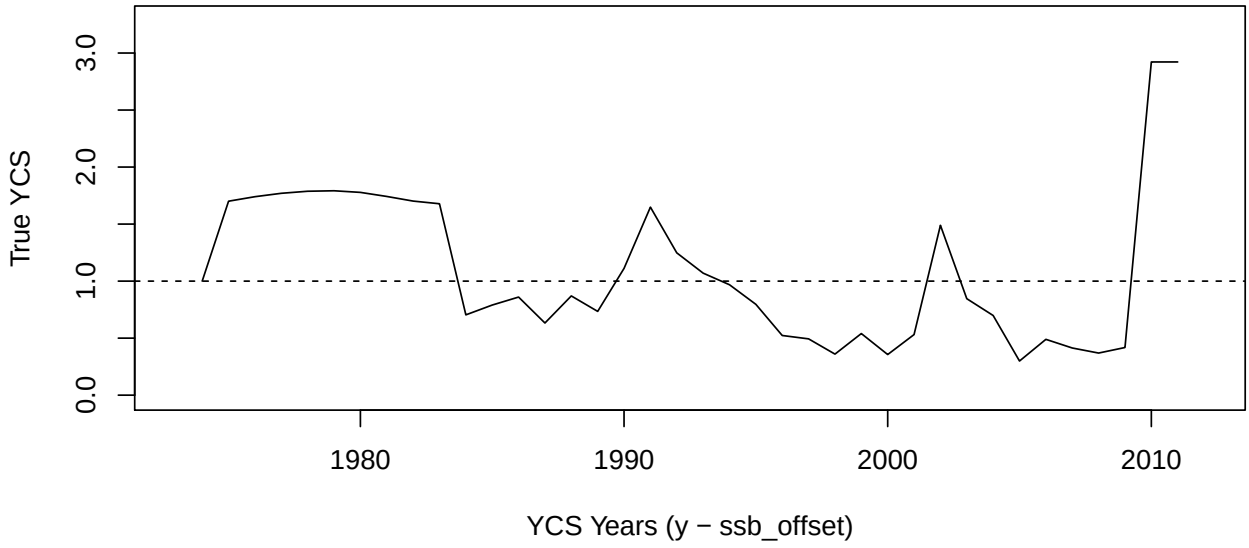


Figure 7: YCS

fishing pressure/exploitation rates for each fishery

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

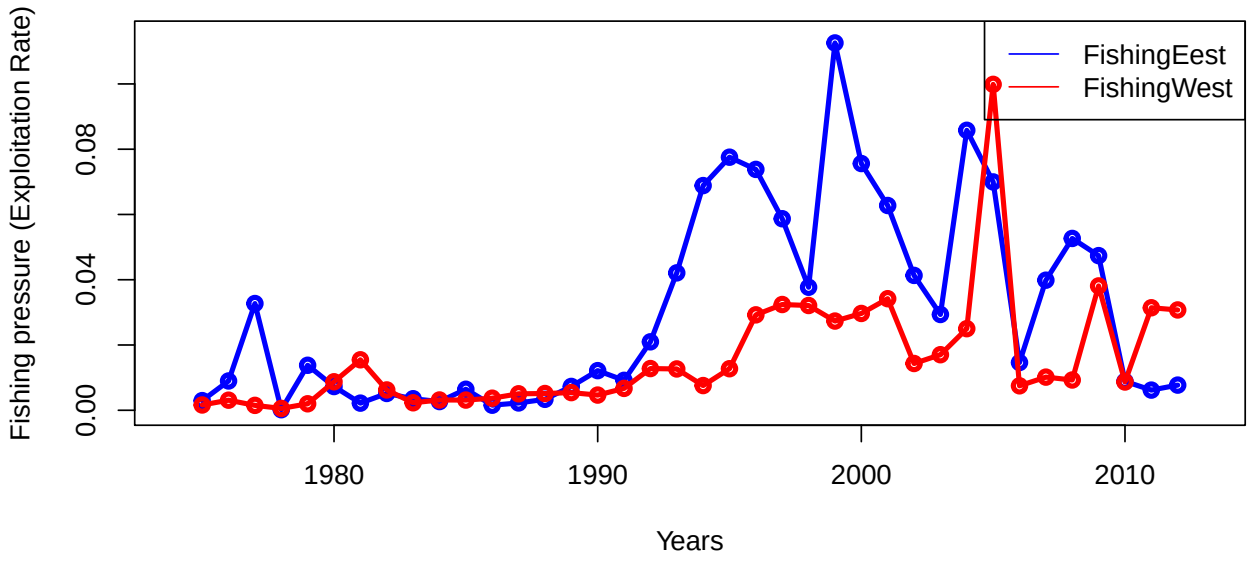


Figure 8: Estimated exploitation

Selectivity curves

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

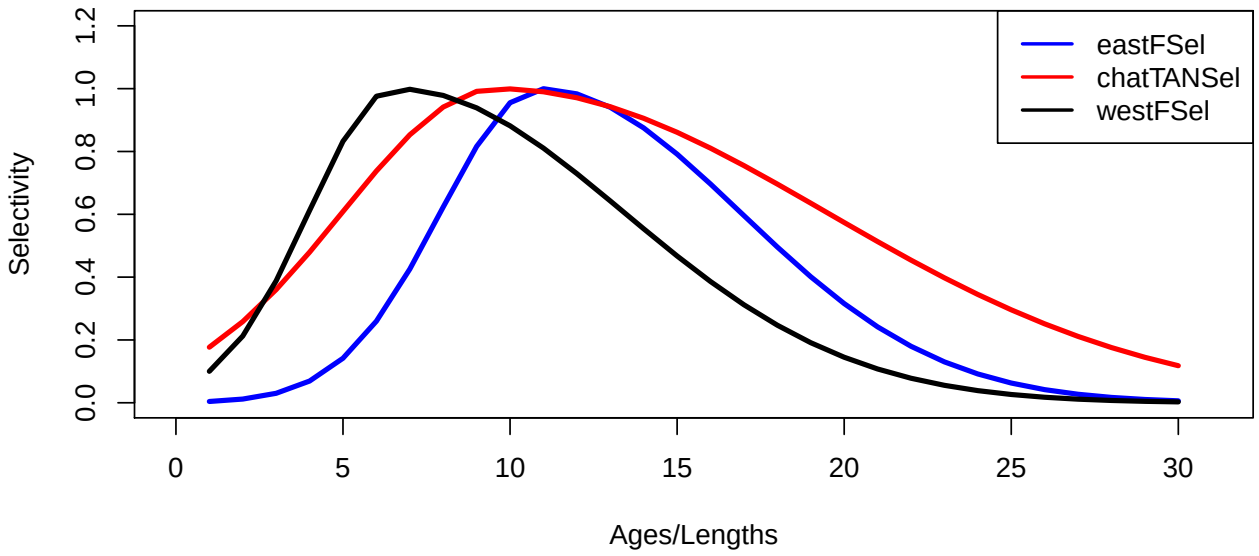


Figure 9: Estimated Selectivities

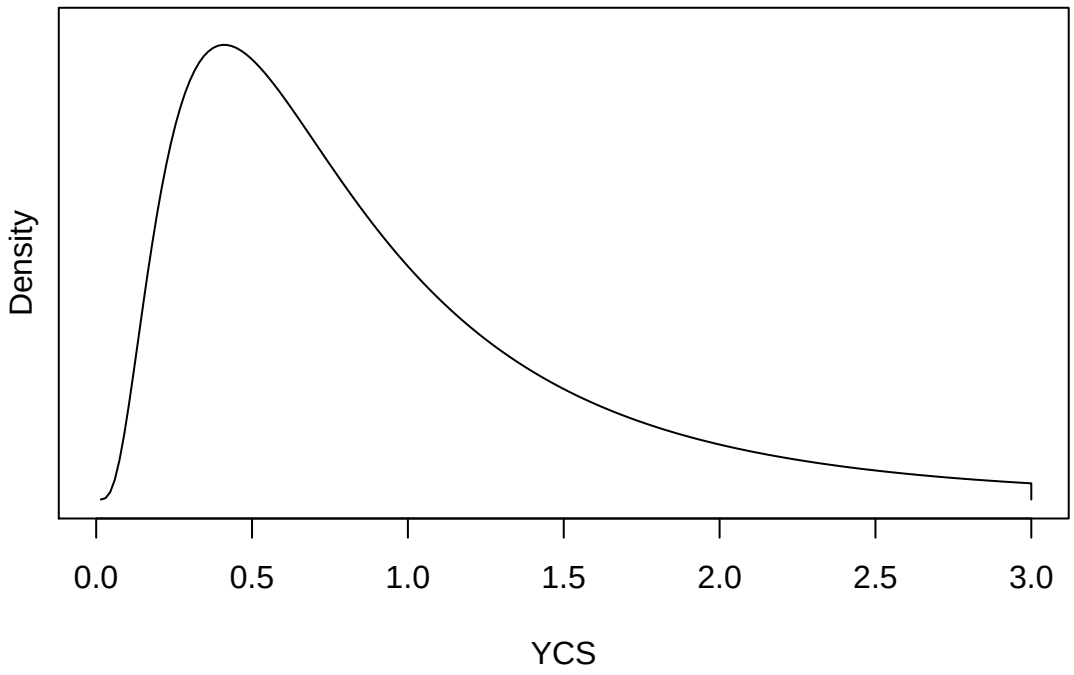
Next, check that the minimum objective function value is reached from multiple starting parameter values.

Plotting priors

It can be useful to visualise priors that are used in a Casal2 model. The `plot.prior()` function from the CASAL R package was added to the Casal2 R package.

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

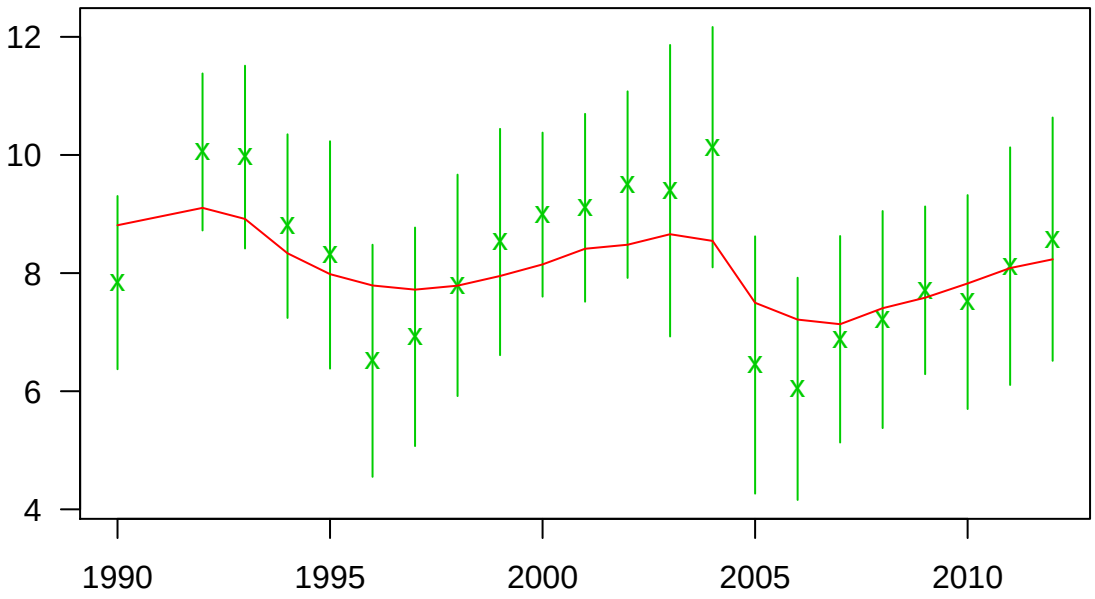
```
## Plotting lognormal prior
```



Dataweighting

Casal2 has inbuilt data weighting functions for relative indices of abundance or biomass with the function `CV.for.CPUE()`. This function generates CVs for a relative biomass index based on loess smoothing. 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 automate the dataweighting 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
```

```
## This function will also strip out all comments
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"))

## assumes that the Casal2 executable is in the path
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] 1.000682
```

```
## overwrite the re-weighed observation csl and change working directory back
setwd(this_path)
```

Assessing MPD convergence

If setting up a model for estimating the MPD, an important consideration is whether the final objective function value (the negative log likelihood) is a local or global minimum. One method available to check for a global minimum is to run multiple estimations with different starting parameter values. The function for generating multiple starting parameter values is `generate.starting.pars()`. This function takes a text configuration file as input and parses the `@estimate` blocks to get the prior and lower and upper bounds for the estimated parameters.

```
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. A note when using this function: this function may generate implausable values if the bounds are wide, so more consideration is needed than when setting bounds for a model. This will create a multi run output, with an example below.

```
# 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 the 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
```

A function to identify parameters that are unidentfiable is `check_mpd_identifiability()`. Set up the model configuration files to report the covariance matrix and the estimated parameters in order for this function to work.

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

Multi Run

Most of the functions that work for the single MPD run cannot be used with multi run reports. If there are multiple model configurations, e.g., model_BH and model_constant, create separate models for these sensitivities. This will be the most common multi model run 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 are two sets of MCMC outputs, the objective function values and parameter samples, and the derived quantitites. The first type of output is created with the `casal2 -m` command, and the latter is created with `casal2 -r -tabular -i mcmc_samples.out`.

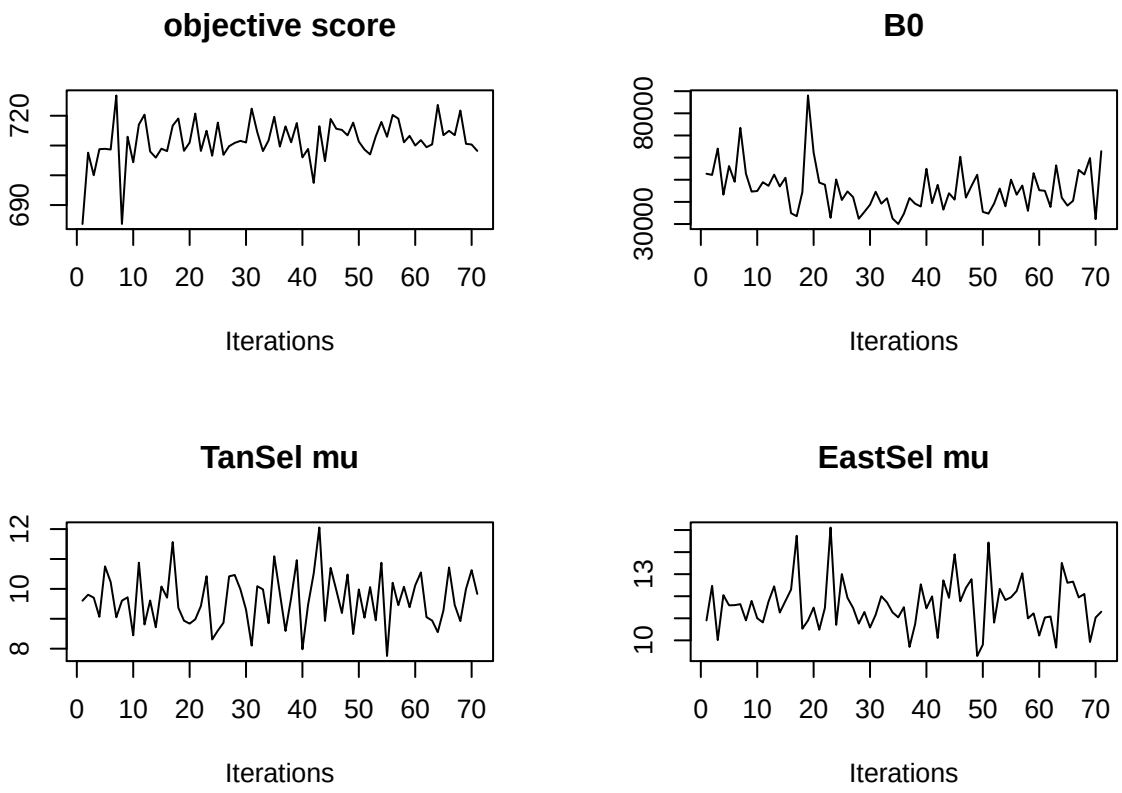
```
# this will create a list like object called single_run
mcmc_out <- extract.mcmc(samples.file = "mcmc_samples.out",
                         objectives.file = "mcmc_objectives.out",
                         return_covariance = T, path = fpath)

# output the covariance matrix, aka proposal covariance for the Metropolis-Hastings MCMC
mcmc_out$Covariance
```

Use the `coda` package for MCMC diagnostics:

```
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 this
# mcmc_chain = as.mcmc(mcmc_out)
```

Now use the diagnostics for proper Bayesian evaluation

```
# 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 include `write.csl2.file()` and `read.csl2.file()`.

Troubleshooting the Casal2 R package

TODO

fileEncoding

For runs using Windows 10, the shell outputs in UTF-16 compared to the cmd prompt which writes text files in UTF-8. The Casal2 R package is built with UTF-8 as the default. Parameters can be set to read files with format UTF-16. If the following error occurs 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
```

This issue may be resolved by using an alternative UTF format by specifying this format with the `fileEncoding` parameter:

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

References

Francis, R I C C. 2011. "Data Weighting in Statistical Fisheries Stock Assessment Models." *Canadian Journal of Fisheries and Aquatic Sciences* 68 (6): 1124–38.