User Guidelines for Advanced Model Diagnostics with ss3diags

Henning Winker (JRC-EC) Felipe Carvalho (NOAA)
Massimiliano Cardinale (SLU) Laurence Kell (Sea++)

20 April, 2022

Contents

1 Getting started

This vignette introduces you to the `ss3diags` R package, which accompanies the paper "A cookbook for using model diagnostics in integrated stock assessments" by Carvalho, Winker et al. (2021).

The `ss3diags` comprises a set of functions for applying advanced model diagnostics to Stock Synthesis models. The package builds on the widely used R package `r4ss` (Taylor et al. 2021), which is designed to support the use of the Stock Synthesis software modeling framework (Methot and Wetzel, 2013). This vignette is divided into four sections. Section 1 consists of installing `ss3diags` and loading the example data from a simulated, cod-like stock that is included with the package. Section 2 describes the plotting of various model diagnostics as described in the Cookbook. Section 3 provides a detailed explanation on how to assess model uncertainty using `ss3diags`. In Section 4 we provide a series of "cookbook recipes" on how to implement selected model diagnostics on Stock Synthesis models.

1.1 Installation

Both `ss3diags` and `r4ss` can be installed from gihtub using the `remotes` package:

```
install.packages("remotes")

remotes::install_github("r4ss/r4ss")

remotes::install_github("PIFSCstockassessments/ss3diags")
```

Once the packages are installed they can be loaded by:

```
library(r4ss)
library(ss3diags)
```

1.2 Loading built-in example data

The package contains output from a simple, cod-like SS model that was simulated using ss3sim. The model includes 2 fleets, one fishery and one survey. Catch data is available from year 26 to year 100 (final year of model). An index of abundance is available from the survey fleet for years 62 - 100. No discard data was simulated. Simulated composition data includes length (fleets 1 and 2), age (fleets 1 and 2), and conditional age-at-length (fleet 1). Examples of the the output of a single run (as read by `r4ss::SS_output()`) of the model as well as the output from a retrospective analysis with 5 year peels (as read by `r4ss::SSgetout()`) are available with the package.

1.2.1 "Simple" model

The example outputs can be loaded into R by:

```
# Single run output
data("simple")

# retrospective analysis output
data("retroSimple")

# mcmc estimation
data("mcmcSimple")
```

- `simple`: list of stock synthesis objects created with `r4ss::SS_output()`
- `retroSimple`: list of retrospective runs created with `r4ss:SS_doRetro()` and read by `r4ss::SSgetoutput()`.
  - The first object in the list is the reference run and the other 5 objects are the 5 1-year peels.
- `mcmcSimple`: dataframe of MCMC posterior distributions

2 Model Diagnostics with ss3diags

The plotting options are kept mainly to those provided by r4ss. Like with r4ss, if, for example, `SSplotRunstest()` is called with no further specifications several windows will open, the number of windows depends on the number abundance indices.

2.1 Residual diagnostics

The runs test is a nonparametric hypothesis test for randomness in a data sequence that calculates the 2-sided p-value to estimate the number of runs (i.e., sequences of values of the same sign) above and below a reference value. The runs test can diagnose model misspecification using residuals from fits to abundance indices (Carvalho et al. 2017) by testing if there are non-random patterns in the residuals. It can also be applied to other data components in assessment models such as the mean-length residuals and mean-age residuals. In addition, the three-sigma limits can be considered to identify potential outliers as any data point would be unlikely given a random process error in the observed residual distribution if it is further than three standard deviations away from the expected residual process average of zero.

The output for `SSplotRunstest()` includes a plot of the residuals by fleet and a table with the results from the runs test and 'three-sigma limit' values. In the plots below, the shaded area represents the 'three-sigma limit', or three residual standard deviations from zero. If any of the individual residual points fall outside of the three-sigma limit, they are colored red as in the fishery length-composition in the example below. Green shaded area indicates the residuals are randomly distributed (p-value >= 0.05) and red shaded area indicates the residuals are not randomly distributed and there is some misspecification with the indices or composition data (p-value < 0.05). To visualize the runs test for multiple indices, it is recommended to use the function `r4ss::sspar()` to specify row and column layout and any other plotting parameters. The option `add=TRUE` included in any of the ss3diags plotting functions prevents the functions from over-writing `sspar()`.

```
r4ss::sspar(mfrow = c(2, 2))
SSplotRunstest(simple, subplots = "cpue", add = TRUE)
   Running Runs Test Diagnostics w/ plots forIndex
   Plotting Residual Runs Tests
   Residual Runs Test (/w plot) stats by Index:
      Index runs.p   test  sigma3.lo sigma3.hi type
   1 Survey  0.033 Failed -0.4320694 0.4320694 cpue
SSplotRunstest(simple, subplots = "len", add = TRUE)
   Running Runs Test Diagnostics w/ plots forMean length
   Plotting Residual Runs Tests
   Residual Runs Test (/w plot) stats by Mean length:
       Index runs.p   test  sigma3.lo sigma3.hi type
   1 Fishery  0.724 Passed -0.1454301 0.1454301  len
   2  Survey  0.338 Passed -0.1105796 0.1105796  len
SSplotRunstest(simple, subplots = "con", add = TRUE)
   Running Runs Test Diagnostics w/ plots forConditional age-at-length
   Plotting Residual Runs Tests
```

```
   Residual Runs Test (/w plot) stats by Conditional age-at-length:
       Index runs.p   test  sigma3.lo sigma3.hi type
   1 Fishery    0.5 Passed -0.1491212 0.1491212  con
```

It is also possible to select the indices that should be plotted. For example, if we only want to plot the fishery length composition residuals, we can specify this with the `indexselect` argument.

```
r4ss::sspar()
SSplotRunstest(simple, subplots = "len", indexselect = 1,
    add = TRUE)
   Running Runs Test Diagnostics w/ plots forMean length
   Plotting Residual Runs Tests
```

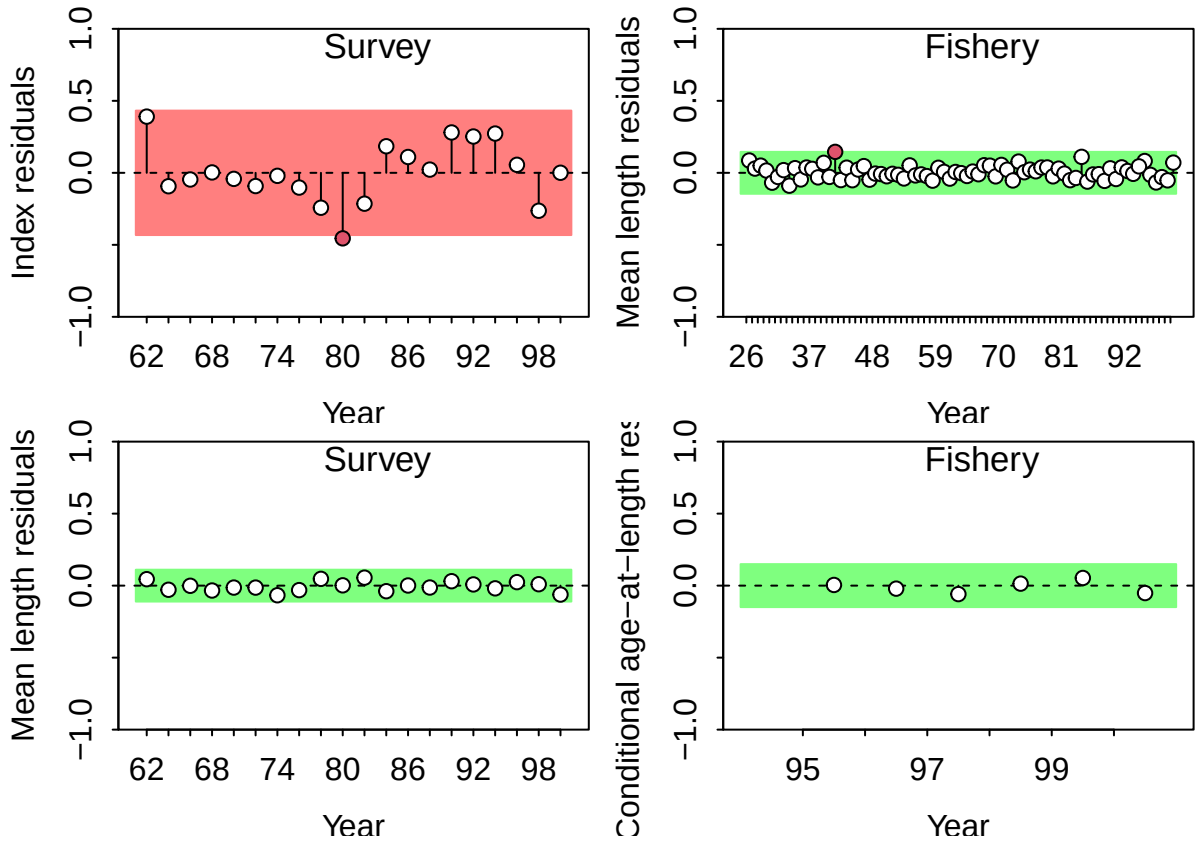


Figure 1: Runs test plots for CPUE index, length-composition, and conditional-age-at-length data fits. Green shading indicates no evidence (p = 0.05) and red shading indicates evidence (p < 0.05) to reject the hypothesis of a randomly distributed time-series of residuals. The shaded (green/red) area spans three residual standard deviations to either side from zero, and the red points outside of the shading violate the 'three-sigma limit' for that series.

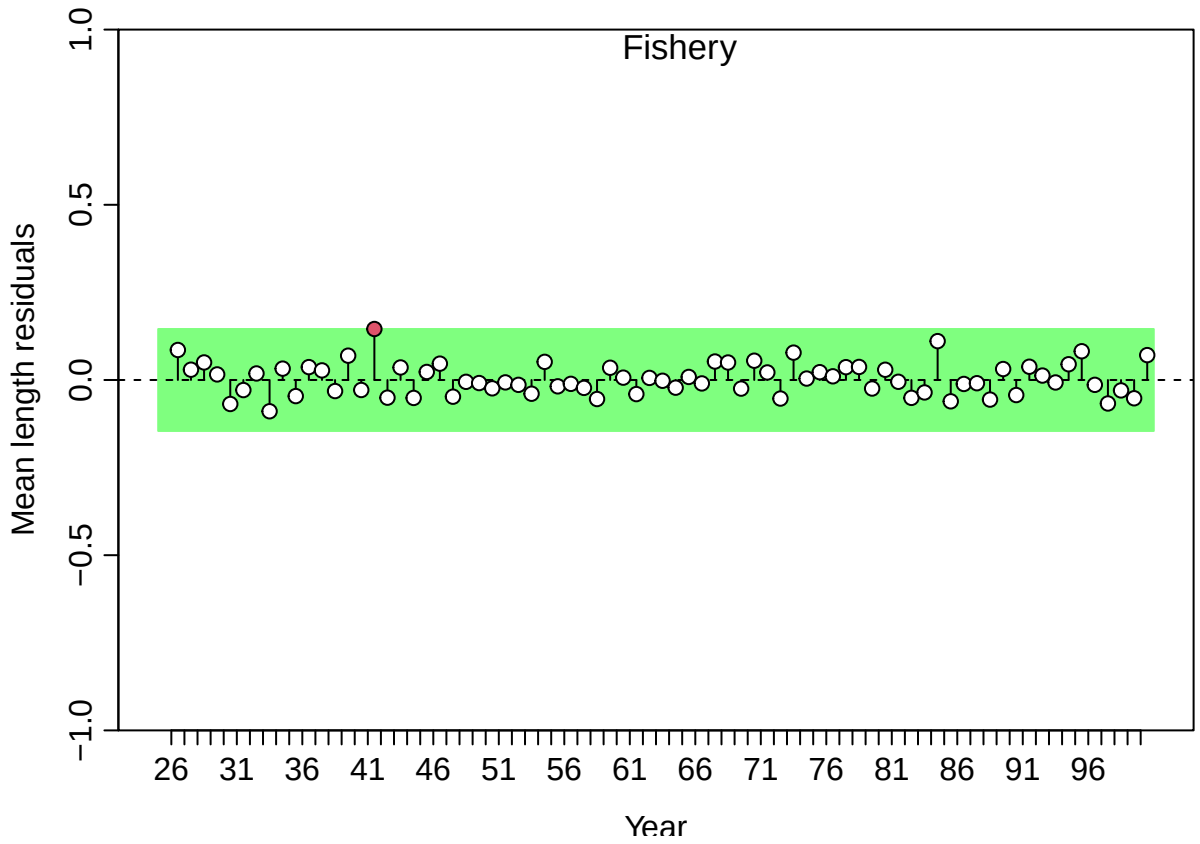


Figure 2: Runs test plot for fits to fishery length composition data.

```
   Residual Runs Test (/w plot) stats by Mean length:
       Index runs.p   test  sigma3.lo sigma3.hi type
   1 Fishery  0.724 Passed -0.1454301 0.1454301  len
```

In addition to the residual plots, `SSplotRunstest()` produces a summary table of the runs test output values, including:

- p-value for the runs test
- if the test passed or failed (indicated by green or red shading in the plot)
- upper and lower limits for the 3-sigma interval
- type of data tested (CPUE, length-composition, age-composition, or conditional age-at-length)

To only produce the summary table and skip the plot, e.g. to faciliate automated processing, use `SSrunstest()`.

```
rcpue <- SSrunstest(simple, quants = "cpue")
   Running Runs Test Diagnosics for Index
   Computing Residual Runs Tests
   Residual Runs Test stats by Index:
rlen <- SSrunstest(simple, quants = "len")
   Running Runs Test Diagnosics for Mean length
   Computing Residual Runs Tests
   Residual Runs Test stats by Mean length:
rbind(rcpue, rlen)
       Index runs.p   test  sigma3.lo sigma3.hi type
   1  Survey  0.033 Failed -0.4320694 0.4320694 cpue
   2 Fishery  0.724 Passed -0.1454301 0.1454301  len
   3  Survey  0.338 Passed -0.1105796 0.1105796  len
```

The second function for residual diagnostics is the function `SSplotJABBAres()`. This function is from the R package JABBA and plots a time series of residuals for all fleets of the indicated data (CPUE or composition). In the example below, we plot the residuals for the mean age (age-composition) and mean length (length-composition) for both fleets.

```
r4ss::sspar(mfrow = c(1, 2), plot.cex = 0.8)
SSplotJABBAres(simple, subplots = "age", add = TRUE)
   RMSE stats by Index:
      indices RMSE.perc nobs
   1  Fishery       9.3   69
   2   Survey       5.1   20
   3 Combined       8.5   89
SSplotJABBAres(simple, subplots = "len", add = TRUE)
```

```
   RMSE stats by Index:
      indices RMSE.perc nobs
   1  Fishery       4.5   75
   2   Survey       3.4   20
   3 Combined       4.3   95
```

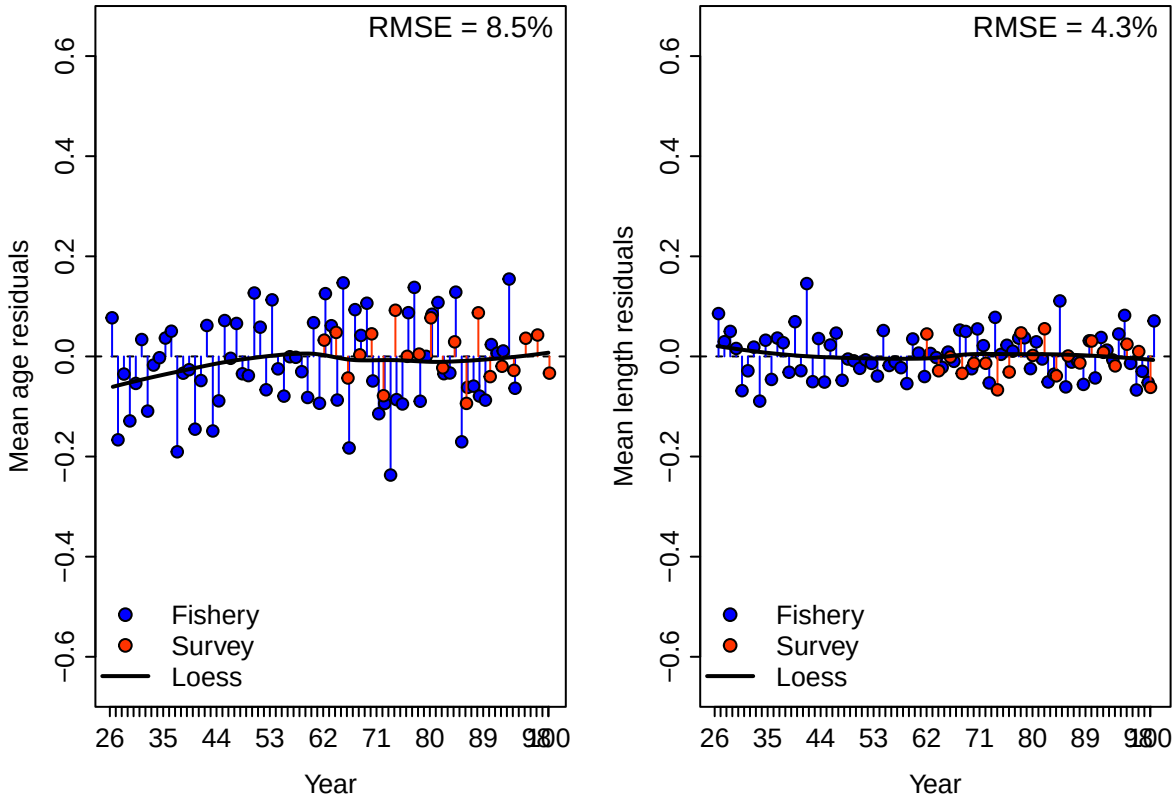


Figure 3: Joint residual plots for fits to age and length compositions, where the vertical lines with points show the residuals, and solid black lines show loess smoother through all residuals. Boxplots indicate the median and quantiles in cases where residuals from the multiple indices are available for any given year. Root-mean squared errors (RMSE) are included in the upper right-hand corner of each plot.

2.2 Retrospective and Forecast bias

Retrospective analysis is commonly used to check the consistency of model estimates, i.e., the invariance in spawning stock biomass (SSB) and fishing mortality (F) as the model is updated with new data in retrospect. The retrospective analysis involves sequentially removing observations from the terminal year (i.e., peels), fitting the model to the truncated series, and then comparing the relative difference between model estimates from the full-time series with the truncated time-series.

In Stock Synthesis, retrospective analysis can be routinely implemented using `r4ss:SS_doRetro()` (see Section 3.1). `ss3diags` provides the function `SSplotRetro()` to visualize the retrospective patterns of SBB and F and compute the associated Mohn's rho value (i.e. retrospective bias). This first requires loading the retrospective runs (Section 1.2), which are already built into `ss3diags` in this case. The next step is to summarize the list of retrospective runs using `r4ss::SSsummarize()`.

```
retroI.simple <- r4ss::SSsummarize(retroSimple, verbose = F)
```

We use the notation "retroI" because `r4ss::SSsummarize()` summarizes the modeled quantities and abundance indices, but not length or age composition data. Using `retroI.simple` it is possible to produce some basic retrospective plots.

```
r4ss::sspar(mfrow = c(1, 2), plot.cex = 0.8)
rssb <- SSplotRetro(retroI.simple, add = TRUE, subplots = "SSB",
    forecast = F, legend = F, verbose = F)
rf <- SSplotRetro(retroI.simple, add = TRUE, subplots = "F",
    ylim = c(0.05, 0.2), forecast = F, legendloc = "topright",
    legendcex = 0.8, verbose = F)
```

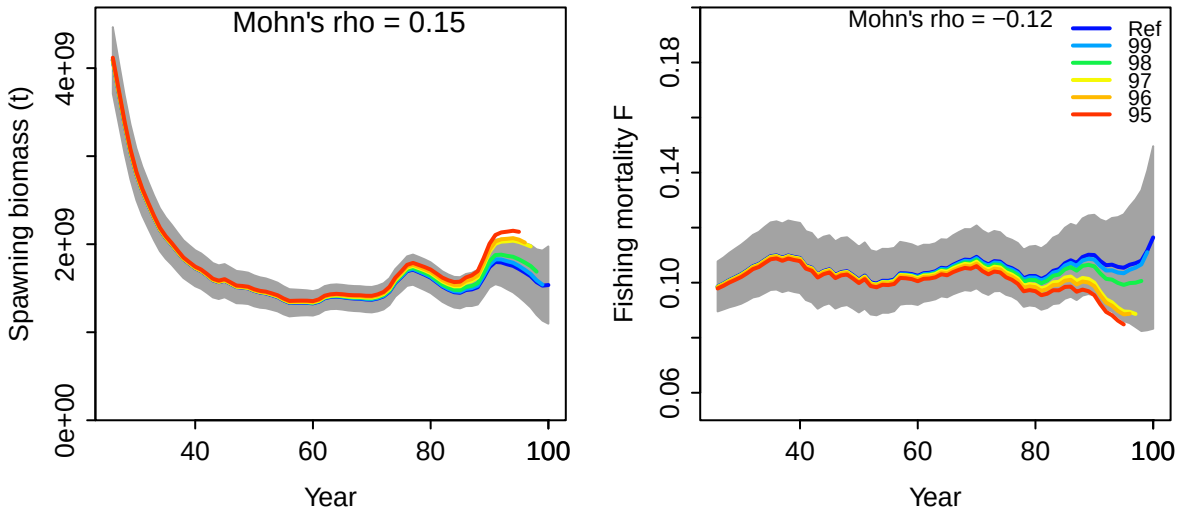


Figure 4: Retrospective analysis of spawning stock biomass (SSB) and fishing mortality estimates for cod-like stock conducted by re-fitting the reference model (Ref) after five years of data were removed, one year at a time sequentially. Mohn's rho statistic are denoted on top of the panels. Grey shaded areas are the 95 % confidence intervals from the reference model in cases where the analysis was run with Hessian.

An intuitive extension of the retrospective analysis is to assess potential forecast bias by adding the additional step of forward projecting quantities, such as SSB, over the truncated years. In Stock Synthesis the forecasts

are automatically done when using `r4ss:SS_doRetro()`.The forecasts are based on the settings specified in 'forecast.ss', which are also evoked when conducting future projections with the same model. The observed catches are used for the retrospective forecasts. Retrospective forecasts with Stock Synthesis are therefore only a matter of visualization, which can be done by setting the `SSplotRetro()` option `forecast=TRUE`.

```
r4ss::sspar(mfrow = c(1, 2), plot.cex = 0.8)
rssb <- SSplotRetro(retroI.simple, add = T, subplots = "SSB",
    forecast = T, legend = F, verbose = F, xmin = 2000,
    ylim = c(5e+08, 2.5e+09))
rf <- SSplotRetro(retroI.simple, add = T, subplots = "F",
    ylim = c(0.05, 0.25), forecast = T, legendloc = "topleft",
    legendcex = 0.8, verbose = F, xmin = 2000)
```

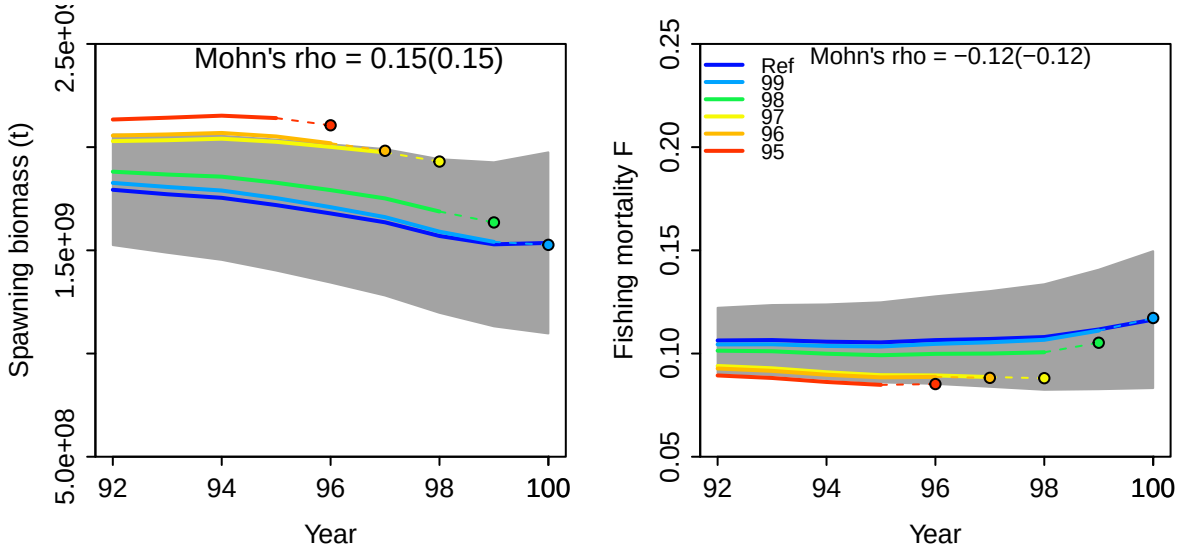


Figure 5: Retrospective results shown for the most recent years only. Mohn's rho statistic and the corresponding 'hindcast rho' values (in brackets) are now printed at the top of the panels. One-year-ahead projections denoted by color-coded dashed lines with terminal points are shown for each model.

The statistics from the retrospective analysis with forecasting, Mohn's rho and forecast bias, can be called without plotting using the function `SShcbias()`

```
SShcbias(retroI.simple, quant = "SSB", verbose = F)
     type     peel         Rho   ForcastRho
   1  SSB       99 0.007769174 -0.006152424
   2  SSB       98 0.075590953  0.069386314
   3  SSB       97 0.207121898  0.229780185
   4  SSB       96 0.202493492  0.211816848
   5  SSB       95 0.245173711  0.254376716
   6  SSB Combined 0.147629846  0.151841528

SShcbias(retroI.simple, quant = "F", verbose = F)
     type     peel         Rho   ForcastRho
   1    F       99 -0.00509569  0.006707778
   2    F       98 -0.06829083 -0.057673536
   3    F       97 -0.17225678 -0.184649147
   4    F       96 -0.16735016 -0.175990996
   5    F       95 -0.19535279 -0.199995306
   6    F Combined -0.12166925 -0.122320241
```

2.3 Hindcast Cross-Validation and prediction skill

Implementing the Hindcast Cross-Validation (HCxval) diagnostic in Stock Synthesis requires the same model outputs generated by `r4ss:SS_doRetro()` as described in Section 3.1. Therefore, no additional step is needed for HCxval if conducted in conjunction with retrospective analysis. As a robust measure of prediction skill, we implemented the mean absolute scaled error (MASE). In brief, the MASE score scales the mean absolute error (MAE) of forecasts (i.e., prediction residuals) to MAE of a naïve in-sample prediction, which is realized in the form of a simple 'persistence algorithm', i.e. tomorrow's weather will be the same as today's (see Eq. 3, p.5 in Carvalho and Winker et al. 2021). A MASE score > 1 indicates that the average model forecasts are worse than a random walk. Conversely, a MASE score of 0.5 indicates that the model forecasts twice as accurately as a naïve baseline prediction; thus, the model has prediction skill.

HCxval is implemented using function `SSplotHCxval()`, which produces the novel HCxval diagnostic plot and computes the MASE scores for CPUE indices, mean lengths or mean ages that have observations falling within the hindcast evaluation period.

Plotting HCxval for abundance indices requires the same step of summarizing the list of retrospective runs as for the retrospective analysis, which therefore only needs be done once.

```
hci <- SSplotHCxval(retroI.simple, add = T, verbose = F,
    legendcex = 0.7)
```

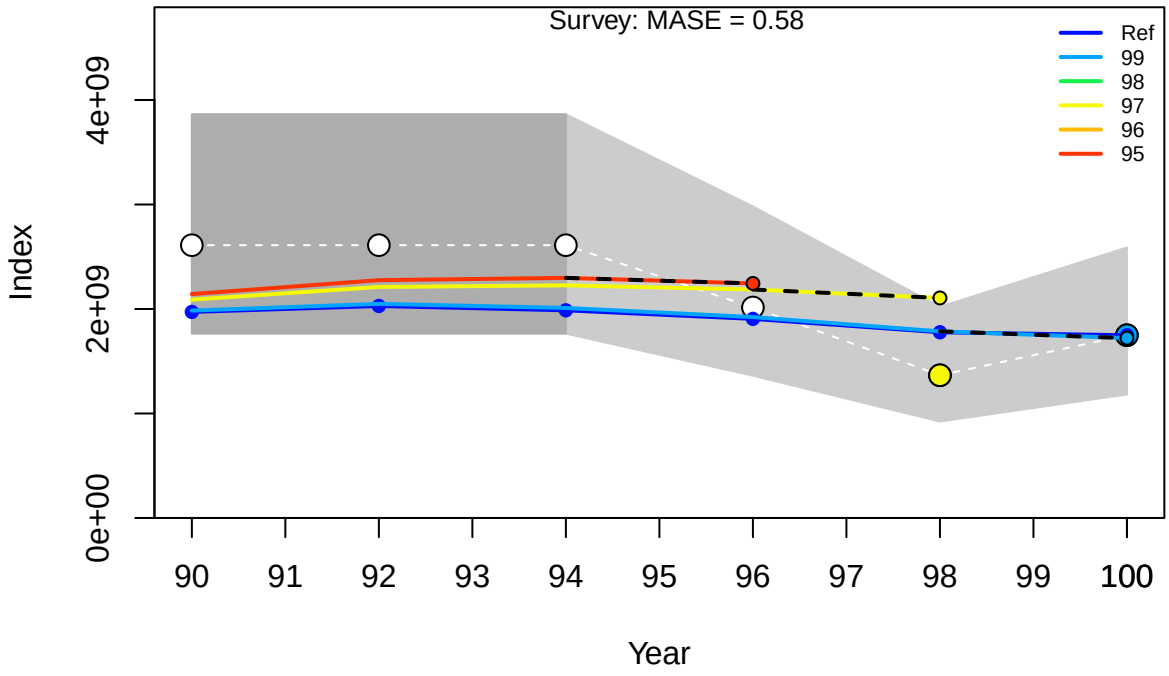


Figure 6: Hindcasting cross-validation (HCxval) results from CPUE fit, showing observed (large points connected with dashed line), fitted (solid lines) and one-year ahead forecast values (small terminal points). HCxval was performed using one reference model (Ref) and five hindcast model runs (solid lines) relative to the expected CPUE. The observations used for cross validation are highlighted as color-coded solid circles with associated 95 % confidence intervals. The model reference year refers to the endpoints of each one-year-ahead forecast and the corresponding observation (i.e., year of peel + 1). The mean absolute scaled error (MASE) score for the survey index is shown at the top of the plot.

The forecast length- and age-composition are located in the Stock Synthesis report.sso as "ghost files". To extract and summarize the composition data in the form of observed and expected mean lengths and age `ss3diags` provides the function `SSretroComps()`.

```
retroC.simple <- SSretroComps(retroSimple)
```

```
r4ss::sspar(mfrow = c(1, 2), plot.cex = 0.8)
hcl <- SSplotHCxval(retroC.simple, subplots = "len",
    add = T, verbose = F, legendcex = 0.7, ylim = c(50,
        100))
```

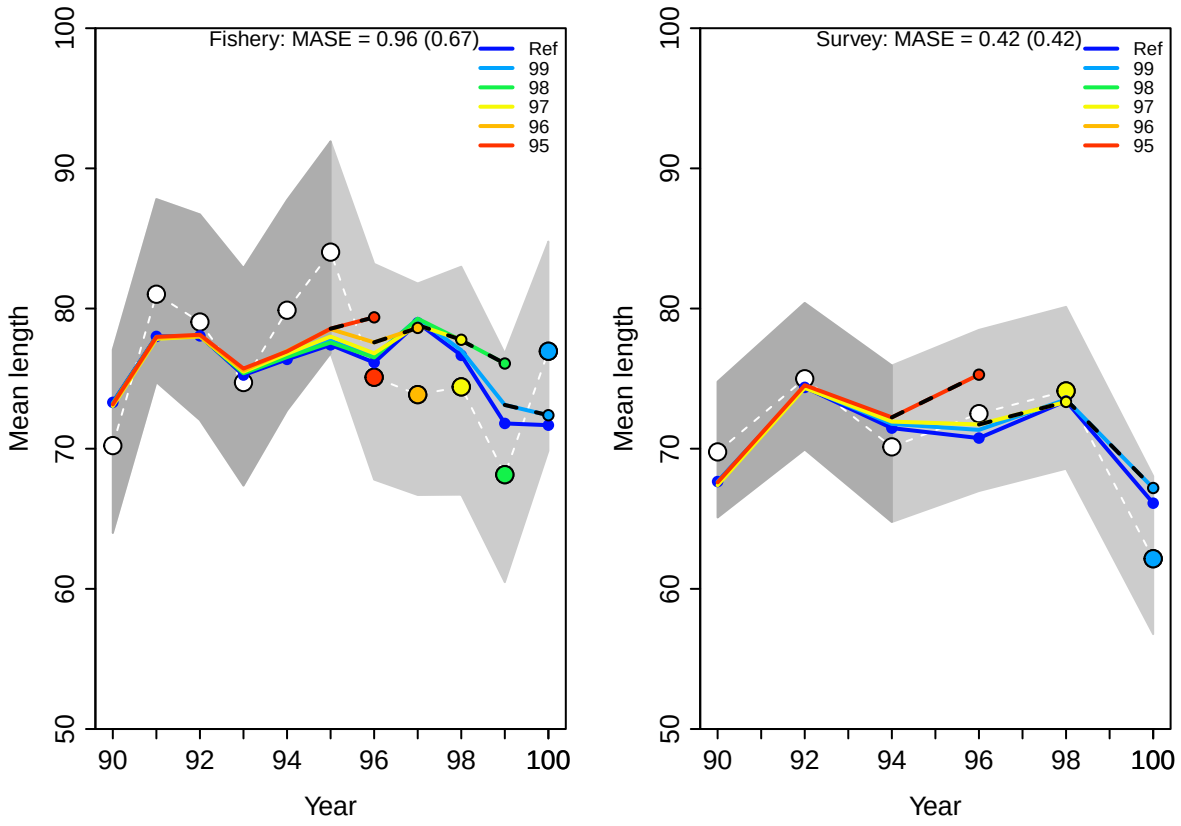


Figure 7: Hindcasting cross-validation (HCxval) results for mean lengths. Note that MASE values in brackets are adjusted MASE values for cases where naive predictions have a Mean-Absolute-Error below 0.1

The figure above provides some additional, so called adjusted MASE values, in parentheses. This gets invoked in cases where the inter-annual variation in the observed values is very small (default MAE < 0.1 for naive predictions log(y[t+1])-log(y[t])). The reasoning is that prediction residuals must be already very accurate to fall below this threshold. The adjusted MASE essential keep the naive prediction MAE denominator of the MASE to a maximum. Below we show the effect of changing adjustment threshold from the default `MAE.base.adj = 0.1`

```
mase1 = SSmase(retroC.simple, quant = "len", MAE.base.adj = 0.1)
mase1
      Index Season      MASE     MAE.PR   MAE.base  MASE.adj n.eval
```

```
   1 Fishery      1 0.9635032 0.06664560 0.06917009 0.6664560      5
   2  Survey      1 0.2433708 0.02412211 0.09911671 0.2412211      2
   3   joint        0.7011276 0.05449603 0.07772627 0.5449603      7
```

to a larger value `MAE.base.adj = 0.15`

```
SSmase(retroC.simple, quant = "len", MAE.base.adj = 0.15)
       Index Season      MASE     MAE.PR   MAE.base  MASE.adj n.eval
   1 Fishery      1 0.9635032 0.06664560 0.06917009 0.4443040      5
   2  Survey      1 0.2433708 0.02412211 0.09911671 0.1608141      2
   3   joint        0.7011276 0.05449603 0.07772627 0.3633069      7
```

where `MASE` is the ratio of the mean absolute error of the prediction residuals `MAE.PR` to the residuals of the naive predictions `MAE.base`

```
mase1$MAE.PR/mase1$MAE.base
   [1] 0.9635032 0.2433708 0.7011276
mase1$MASE
   [1] 0.9635032 0.2433708 0.7011276
```

and MASE.adj

```
mase1$MAE.PR/pmax(mase1$MAE.base, 0.1)
   [1] 0.6664560 0.2412211 0.5449603
mase1$MASE.adj
   [1] 0.6664560 0.2412211 0.5449603
```

Note that applying HCxval for composition data requires correctly specifying the composition data type fitted in the model. For example, age composition data need to be specified as "age" in `SSplotHCxval` and `SSmase`, as shown below.

```
hcl <- SSplotHCxval(retroC.simple, subplots = "age",
    add = TRUE, verbose = F, legendcex = 0.7)
```

```
SSmase(retroC.simple, quants = "age")
       Index Season      MASE     MAE.PR  MAE.base  MASE.adj n.eval
   1 Fishery      1        NA         NA        NA        NA      0
   2  Survey      1 0.3171172 0.04623051 0.1457836 0.3171172      2
   3   joint        0.3171172 0.04623051 0.1457836 0.3171172      2
```

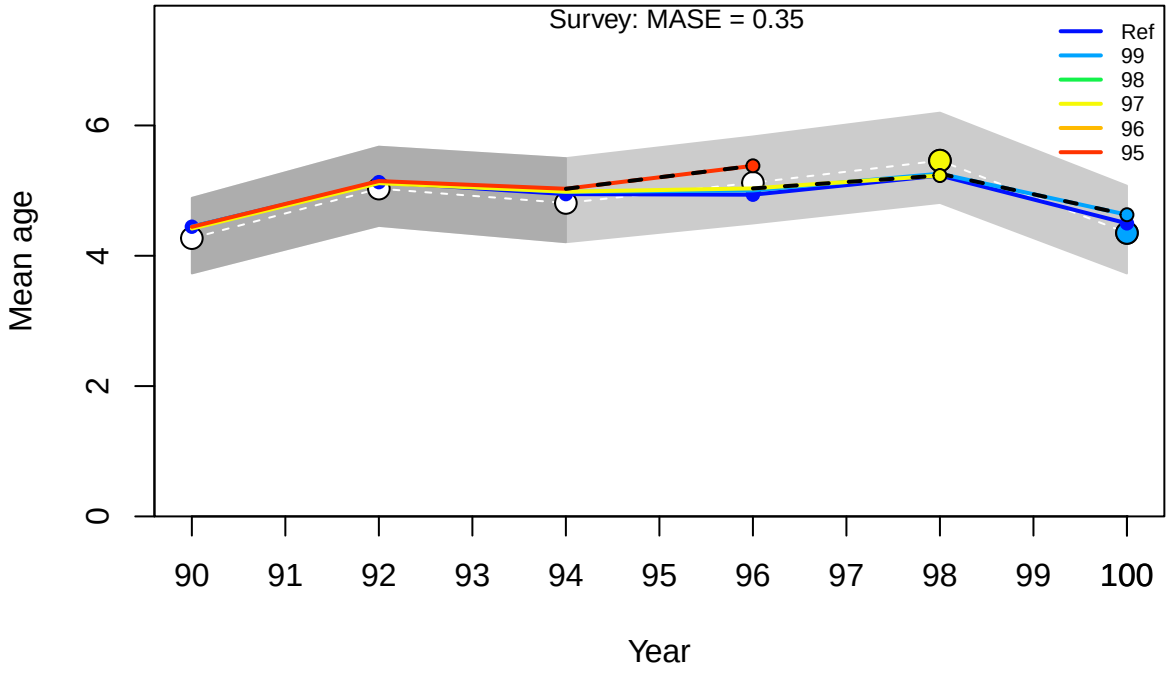


Figure 8: Hindcasting cross-validation (HCxval) results for mean ages. Note that MASE values in brackets are adjusted MASE values for cases where naive predictions have a Mean-Absolute-Error below 0.1

3 Model uncertainty

The management advice frameworks increasingly require translating the estimated uncertainty about the stock status into probabilistic statements (Kell et al. 2016). A classic example is the Kobe framework used in tuna Regional Fisheries Management Organisations (tRFMOs) around the world. The key quantities of interest are typically the ratios SSB/SSB_{MSY} and F/F_{MSY}. While it is reasonably straight forward in Stock Synthesis to approximate uncertainty of individual quantities (e.g. SSB) from the asymptotic standard errors (SE) derived from the Hessian matrix using the delta method, the joint distribution of SSB/SSB_{MSY} and F/F_{MSY} requires adequately accounting for the covariance structure between these two derived quantities. Joint distributions are typically constructed using bootstrap or Markov Chain Monte-Carlo (MCMC) methods. However, these methods can be computationally intense and time-consuming in integrated assessments.

As an alternative, `ss3diags` implements a rapid delta-Multivariate lognormal approximation with `SSdeltaMVLN()` to generate joint error distributions for SSB/SSB_{ref} and F/F_{ref}, where the ref may refer to MSY, but also other reference points (e.g., SSB_{40} and F_{40}). In Stock Synthesis, these ratios are determined by the derived quantities `Bratio` and `F`, where either can take the form of ratios (e.g. F/F_{ref}) or absolute value (e.g. `absF`) depending on settings in the `starter.ss` file.

Let `Bratio` be $u = SSB/SSB_{ref}$, `F` be $v = F$, and $w = F_{ref}$ be the F reference point of interest (e.g. F_{MSY}), with $x = \log(u)$, $y = \log(v)$ and $z = \log(w)$, then the variance-covariance matrix VCM has the form:

$$VCM_{x,y,z} = \begin{pmatrix} \sigma_x^2 & cov_{x,y} & cov_{x,y} \\ cov_{x,y} & \sigma_y^2 & cov_{y,z} \\ cov_{x,z} & cov_{y,z} & \sigma_z^2 \end{pmatrix}$$

where, e.g., σ_x^2 is the variance of x and $cov_{x,y}$ is the covariance of x and y. Deriving those requires conducting a few normal to lognormal transformations. First, the variances are approximated as:

$$\sigma_x^2 = \log\left(1 + \left(\frac{SE_u}{u}\right)^2\right)$$

where SE_u, SE_v and SE_z are the asymptotic standard error estimates for $u = SSB/SSB_{ref}$, $v = F$ and $z = F_{ref}$.

The corresponding covariance for x and y, can then be approximated on the log-scale by:

$$COV_{x,y} = \log\left(1 + \rho_{u,v}\sqrt{\sigma_x^2\sigma_y^2}\right)$$

where $rho_{u,v}$ denotes the correlation of u and v.

To generate a joint distribution of $\tilde{u} = SSB/SSB_{ref}$, $\tilde{v} = F$ and $\tilde{z} = F_{ref}$, a multivariate random generator is used, which is available in the R package 'mvtnorm', to obtain a large number (e.g. nsim = 10,000) iterations, such that

$$JD(\tilde{u}, \tilde{v}, \tilde{w}) = \exp(MVN(\mu_{x,y,z}, VCM_{x,y,z}))$$

so that

$$\tilde{SSB}/\tilde{SSB}_{MSY} = \tilde{u}$$

and

$$\tilde{F}/\tilde{F}_{MSY} = \tilde{v}/\tilde{w}$$

The reference points depend on the settings in the `starter.ss` file that determine the derived quantities `Bratio` and `Fvalue`.

We provide the function `SSsettingsBratioF(simple)` to view the `starter.ss` settings:

```
SSsettingsBratioF(simple)
   $Bratio
   [1] "SSB/SSB0"

   $F
   [1] "_abs_F"

   $Bref
   [1] 0.4
```

This function is also inbuilt in `SSdeltaMVLN()` to prevent misleading results. The `SSdeltaMVLN()` output includes the maximum likelihood estimates (MLEs) and the MVLN Monte-Carlo distributions `$kb` of SSB/SSB_{MSY}, F/F_{MSY} and F. Note the additional quantities SSB and Rec are generated independently from lognormal distributions for practical reasons. These can be plotted by `SSplotEnsemble()`.

The `SSdeltaMVLN()` provides the option to set alternative `Fref` values, but this is only possible for the recommended `starter.ss` option 0 for `F_report_basis`. For option 2, `SSdeltaMVLN()` prompts an error if `Fref` is changed.

The `simple` model is run with settings that are common in NOAA assessments, with `Bratio` set to SSB/SSB_0 and `F` is typically kept at absolute quantity.

```
1 # Depletion basis: 1=rel X*SB0; 2=rel SPBmsy; 3=rel X*SPB_styr; 4=rel X*SPB_endyr
```

```
0 # F_report_basis: 0=raw_F_report; 1=F/Fspr; 2=F/Fmsy ; 3=F/Fbtgt
```

The management quantities in this case are SSB/SSB_{40} and F/F_{spr40}, where the target of 40% is specified in the `forecast.ss` file.

```
0.4 # SPR target (e.g. 0.40)
```

```
0.4 # Biomass target (e.g. 0.40)
```

```
mvln <- SSdeltaMVLN(simple, run = "Simple", Fref = "SPR",
    plot = TRUE)

    starter.sso with Bratio: SSB/SSB0 and F: _abs_F
```

```
r4ss::sspar(mfrow = c(3, 2), plot.cex = 0.7)
SSplotEnsemble(mvln$kb, ylabs = mvln$labels, add = T,
    verbose = F)
```

In some instances, mismatches between the`SSdeltaMVLN` and MCMC may be caused by the latter's poor performance due to poor regularization (Monnahan et al., 2019) or in cases where key parameters such as steepness h or natural M are estimated using informative priors, which can result in left skewed (non-lognormal) distributions of the benchmarks F_{ref} and B_{ref} (Stewart et al. (2013) and Taylor et al. (2021)).

To facilitate a comparison between the `SSdeltaMLVN()` and MCMC outputs, we provide the function `SSdiagsMCMC()`, which is illustrated on the example of the Simple cod-like Stock Synthesis model.

`SSdiagsMCMC()` requires loading both the `report.sso` and MCMC output in the `posterior.sso` file, where the MCMC was in this case run in the subfolder of the assessment file `/mcmc`. For an example, we have provided MCMC output for the simple model which can be loaded in by `data("mcmcSimple")`.

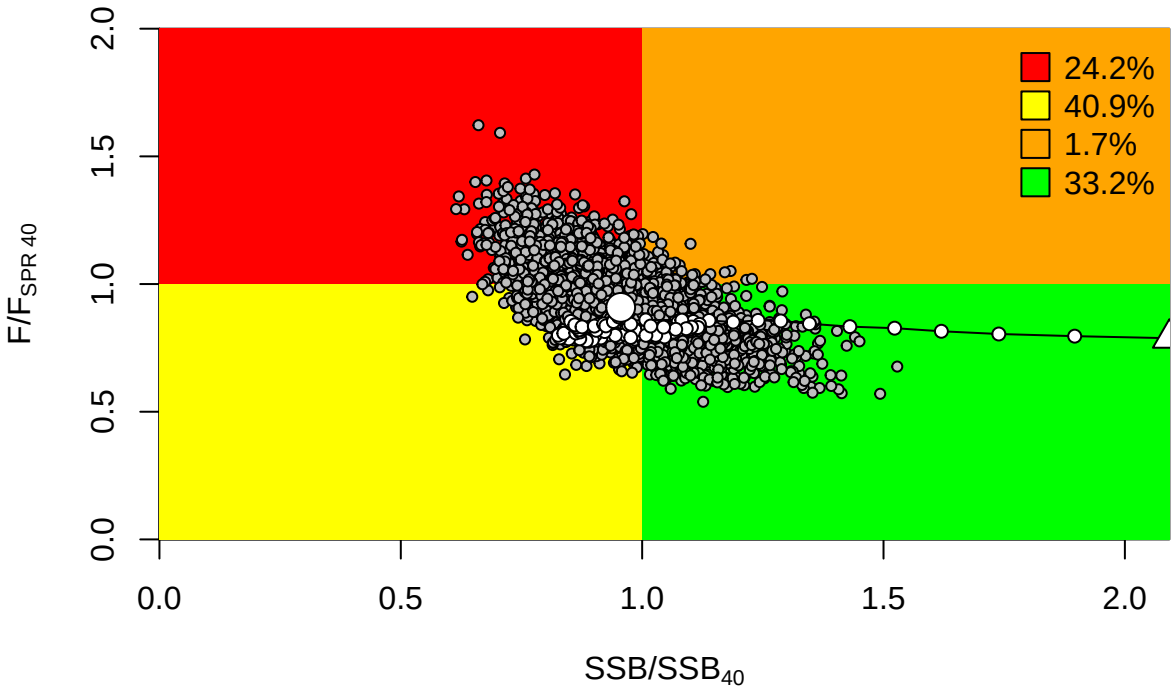


Figure 9: Kobe phase plots showing MVLN Kobe probability distributions of SSB/SSB_{40} and F/F_{SPR40} for the simple SS3 model.

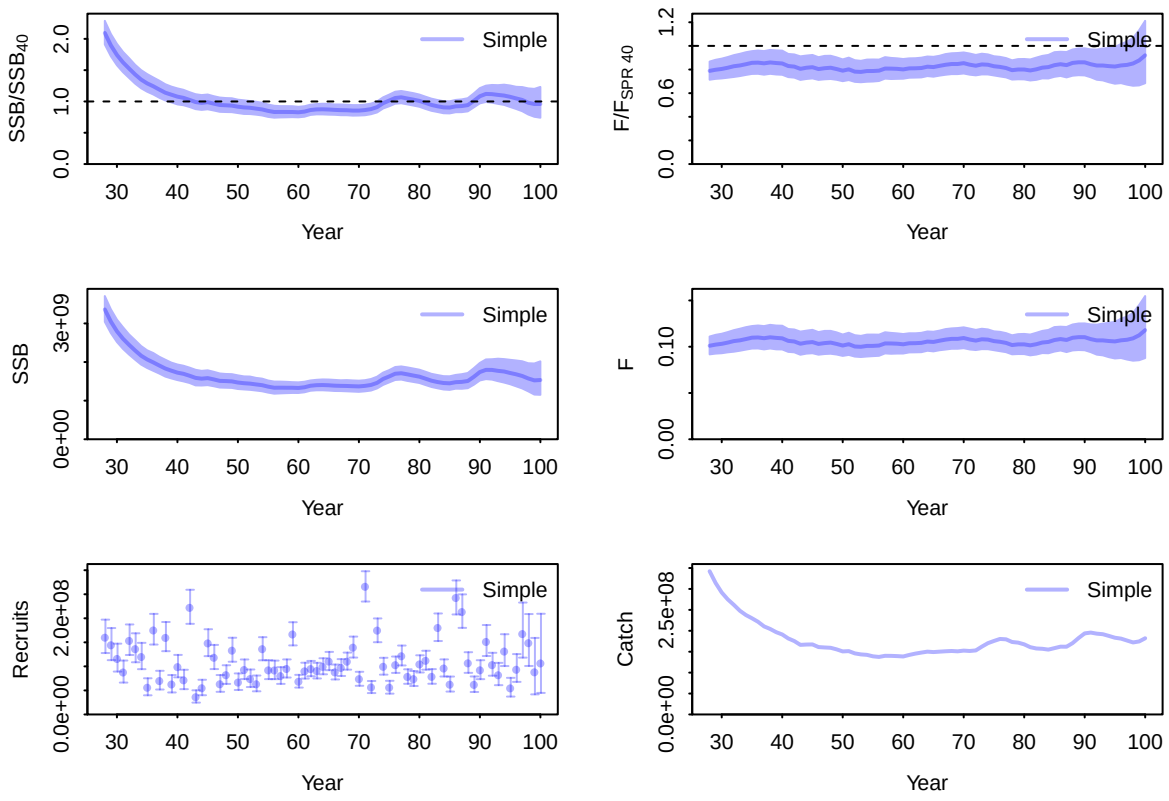


Figure 10: Distributions for SSB/SSB_{40}, F/F_{SPR40}, SSB, F, Recruitment and Catch trajectories for the simple SS3 model

```
# report file
report <- SS_output("./reference_run")
mcmc <- SSgetMCMC("./reference_run/mcmc")

# MCMC example included in ss3diags
data("mcmcSimple")
```

The options and output of `SSdiagsMCMC()` are largely identical to `SSdeltaMVLN`.

```
mvln <- SSdeltaMVLN(simple, plot = F, run = "mvln")

    starter.sso with Bratio: SSB/SSB0 and F: _abs_F

mcmc <- SSdiagsMCMC(mcmcSimple, simple, plot = F, run = "mcmc")

    starter.sso with Bratio: SSB/SSB0 and F: _abs_F

mcmc$kb <- dplyr::filter(mcmc$kb, year > 27)
```

Comparing delta-MVLN with MCMC simply requires combining the `$kb` outputs by, e.g.,

```
r4ss::sspar(mfrow = c(1, 1), plot.cex = 0.8)
SSplotKobe(rbind(mvln$kb, mcmc$kb), joint = F, xlab = mvln$labels[1],
    ylab = mvln$labels[2], fill = F)
```

```
     Quadrant   Percent
1         Red 47.136132
2      Orange  9.003531
3      Yellow 17.595135
4       Green 26.265202
```

```
r4ss::sspar(mfrow = c(2, 2), plot.cex = 0.7)
SSplotEnsemble(rbind(mvln$kb, mcmc$kb), ylabs = mvln$labels,
    add = T, subplots = c("stock", "harvest", "SSB",
        "F"), verbose = F)
```

This function works equally for joining a model ensemble.

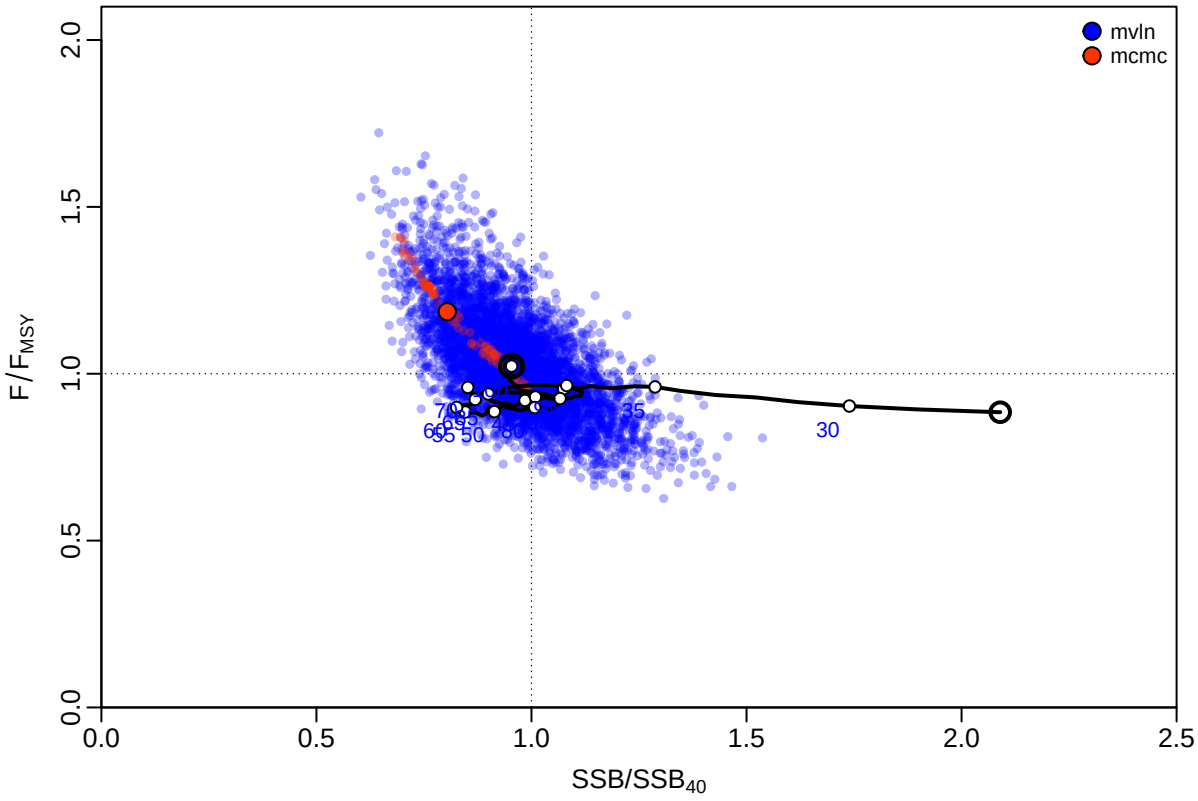


Figure 11: Kobe phase plot comparing MVLN and MCMC posterior distributions of SSB/SSB_{40} and F/F_{40} for the simple SS3 model

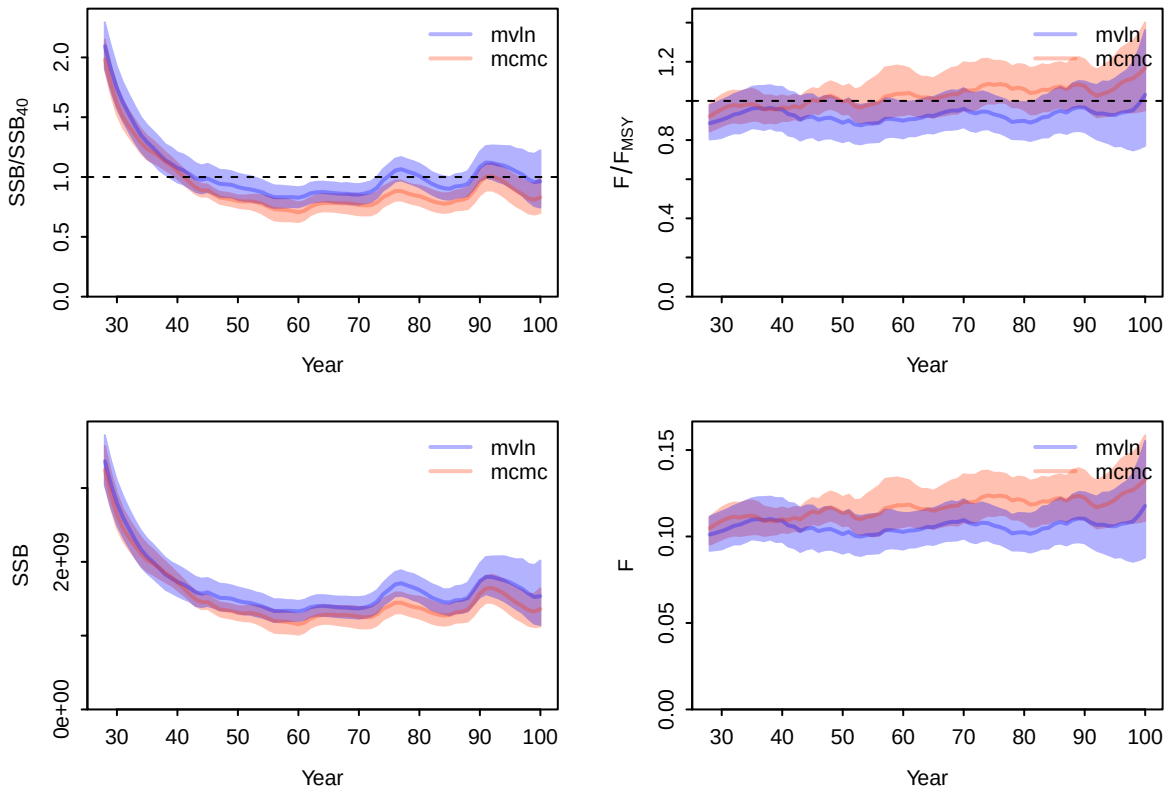


Figure 12: Comparison of MVLN and MCMC posterior distributions for SSB/SSB_{40}, F/F_{SB40}, SSB and F for the simple SS3 model

4 Cookbook Recipies

4.1 Retrospectives with hindcasts

Retrospective analysis can be run for Stock Synthesis using the function `r4ss::SS_doRetro()` available in r4ss. This setup of the retrospective analysis has the advantage that forecasts are conducted automatically given the catch. This makes it possible to apply retrospective forecasting and hindcast cross-validations of observations based on the same output.

Below is a step-by-step cookbook recipe for retrospective analysis in Stock Synthesis which can also be found in the model recipes.

4.1.1 Step 1: Identify restrospective period

Specify the range pf peels that will then determine the `end.yr.vec` of runs in `r4ss::SS_doRetro()`

```
start.retro <- 0  # end year of reference year
end.retro <- 5  # number of years for retrospective e.g.,
```

4.1.2 Step 2: Identify the base directory

Specify the path directory that holds the folder with the base case run. In this case the 'Simple' folder with a model folder 'Reference_Run"

```
dirname.base = "./Simple"
run = "Reference_Run"
model.run <- file.path(dirname.base, run)
```

4.1.3 Step 3: `DAT` and `CONTROL` files

Specify the names of the data and control files. Note these files are named differently from the `DATA.ss` and `CONTROL.ss`. In this case

```
DAT = "simple.dat"
CTL = "simple.ctl"
```

The names of the DAT and CONTROL are declared on the top of the 'starter.ss`, e.g.`#C Simple starter filesimple.datsimple.ctl'

4.1.4 Step 4: Create a subdirectory for the Retrospectives

There are several ways to organize the retrospective output structure. First create a new subfolder for the retrospective runs output

```
dir.retro <- paste0(dirname.base, "/Retro_", run)
dir.create(path = dir.retro, showWarnings = F)
```

Also create a subdirectory for the retrospective model folders

```
dir.create(path = file.path(dir.retro, "retros"), showWarnings = F)
```

then copy model run files to the new retrospective folder

```
file.copy(file.path(model.run, "starter.ss_new"), file.path(dir.retro,
    "starter.ss"))
file.copy(file.path(model.run, "control.ss_new"), file.path(dir.retro,
    CTL))
file.copy(file.path(model.run, "data.ss_new"), file.path(dir.retro,
    DAT))
file.copy(file.path(model.run, "forecast.ss"), file.path(dir.retro,
    "forecast.ss"))
file.copy(file.path(model.run, "SS.exe"), file.path(dir.retro,
    "SS.exe"))
# Automatically ignored for models without
# wtatage.ss
file.copy(file.path(model.run, "wtatage.ss"), file.path(dir.retro,
    "wtatage.ss"))
```

4.1.5 Step 5: Modify Starter.ss file

Modifying the Starter File helps to speed up model runs

```
starter <- r4ss::SS_readstarter(file = file.path(dir.retro,
    "starter.ss"))

starter$run_display_detail <- 1
# write modified starter.ss
r4ss::SS_writestarter(starter, file.path(dir.retro,
    "starter.ss"))
```

4.1.6 Step 6: Execute retrospective runs

Run the retrospective analyses using r4SS function `r4ss::SS_doRetro`.Ideally, the runs should be done with Hessian to evaluate the retrospective trajectories with respect to the confidence interval coverage of the reference model.

```
r4ss::SS_doRetro(masterdir = dir.retro, oldsubdir = "",
    newsubdir = "", years = start.retro:-end.retro)
```

However, for larger models it might be desirable to shorten run times, by not inverting the hessian, using the option `extras = "-nohess"`. This runs the model much faster.

```
r4ss::SS_doRetro(masterdir = dir.retro, oldsubdir = "",
    newsubdir = "", years = start.retro:-end.retro,
    extras = "-nohess")
```

4.1.7 Step 7: Read `SS_doRetro()` output

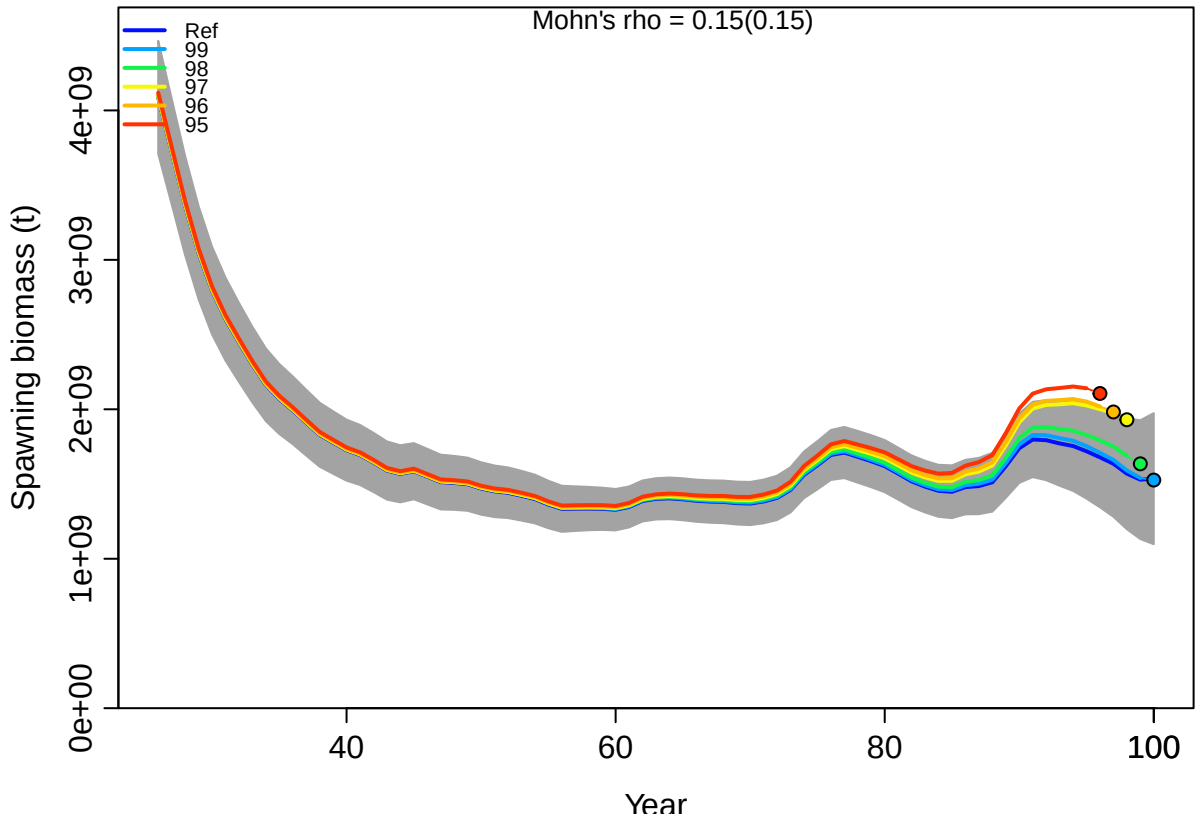
```
retroSimple <- r4ss::SSgetoutput(dirvec = file.path(dir.retro,
    paste0("retro", start.retro:-end.retro)))
```

It is often useful to save the retro model runs as `.rdata` file for further processing with **ss3diags**, considering that reading the models with `r4ss::SSgetoutput()` can be quite time-consuming for more complex models.

```
save(retroSimple, file = file.path(dir.retro, "retroSimple.rdata"))
```

4.1.8 Step 8: Check

```
library(ss3diags)
data("retroSimple")
check.retro = r4ss::SSsummarize(retroSimple)
   Summarizing 6 models:
   imodel=1/6
     N active pars = 112
   imodel=2/6
     N active pars = 112
   imodel=3/6
     N active pars = 112
   imodel=4/6
     N active pars = 112
   imodel=5/6
     N active pars = 112
   imodel=6/6
     N active pars = 112
   Summary finished. To avoid printing details above, use 'verbose = FALSE'.
r4ss::sspar(mfrow = c(1, 1))
SSplotRetro(check.retro, forecast = T, add = T, legendcex = 0.7,
    legendloc = "topleft")
   Plotting Retrospective pattern
```




```
Mohn's Rho stats, including one step ahead forecasts:
  type     peel         Rho  ForecastRho
1  SSB       99 0.007769174 -0.006152424
2  SSB       98 0.075590953  0.069386314
3  SSB       97 0.207121898  0.229780185
4  SSB       96 0.202493492  0.211816848
5  SSB       95 0.245173711  0.254376716
6  SSB Combined 0.147629846  0.151841528
```

4.2 R_0 Profiling

Likelihood profiling is a key model diagnostic that helps identify the influence of information sources on model estimates. R_0 is a commonly profiled parameter because it represents a global scaling parameter. To conduct a profile, values of the parameter over a given range are fixed and the model re-run and then changes in total likelihood and data-component likelihoods are examined.

Below is a step-by-step cookbook recipe for R_0 profiling in Stock Synthesis which can also be found in the model recipes.

4.2.1 Step 1: Base model directory

Identify the directory where completed base model was run.

```
completed.model.run <- "./reference_run_orig"
```

4.2.2 Step 2: Create R0_profile subdirectory

```
dirname.R0.profile <- paste0(completed.model.run, "/Likelihood profiles/R0")
dir.create(path = dirname.R0.profile, showWarnings = TRUE,
    recursive = TRUE)
```

4.2.3 Step 3: Create a "Figures_Tables" subdirectory

```
plotdir = paste0(dirname.R0.profile, "/Figures & Tables")
dir.create(path = plotdir, showWarnings = TRUE, recursive = TRUE)
```

4.2.4 Step 4: Copy completed base model output to "R0_profile" directory

```
list_of_files <- list.files(completed.model.run)
file.copy(file.path(completed.model.run, list_of_files),
    dirname.R0.profile)
```

4.2.5 Step 5: Edit "control.ss" in the "R0_profile" working directory to estimate at least one parameter in each phase

```
control.file <- SS_readctl(file = file.path(dirname.R0.profile,
    "control.ss_new"), datlist = file.path(dirname.R0.profile,
    "data.ss_new"))
control.file$recdev_phase <- 1
SS_writectl(control.file, outfile = file.path(dirname.R0.profile,
    "control.ss_new"))
```

4.2.6 Step 6: Edit "starter.ss" file

Edit the starter file in the "R0_profile" working directory to read from init values from control_modified.ss.

```
starter.file <- SS_readstarter(file.path(dirname.R0.profile,
    "/starter.ss", sep = ""))
# make sure names for control and data file are
# correct
starter.file$ctlfile <- "control_modified.ss"
starter.file$datfile <- "data.ss_new"
# for non-estimated quantities
starter.file$init_values_src <- 0
# Make sure the prior likelihood is calculated
# for non-estimated quantities
starter.file$prior_like <- 1
SS_writestarter(starter.file, dir = dirname.R0.profile,
    overwrite = TRUE)
```

4.2.7 Step 7: Begin likelihood profiling

Create a vector of values to profile over. Make sure the values span both sides of the estimated parameter value (found in the control.ss_new file). Then use the `r4ss::SS_profile()` function to run the models.

```
# vector of values to profile over
R0.vec <- seq(16, 20, 0.1)
Nprof.R0 <- length(R0.vec)

profile <- SS_profile(dir = dirname.R0.profile, model = "ss",
    masterctlfile = "control.ss_new", newctlfile = "control_modified.ss",
    string = "SR_LN(R0)", profilevec = R0.vec)
```

4.2.8 Step 8: Summarize and analyze output

```
# read the output files (with names like
# Report1.sso, Report2.sso, etc.)
prof.R0.models <- SSgetoutput(dirvec = dirname.R0.profile,
    keyvec = 1:Nprof.R0, getcovar = FALSE)  #

# summarize output
prof.R0.summary <- SSsummarize(prof.R0.models)

# add base model into summary
MLEmodel <- SS_output(dirname.completed.model.run)
prof.R0.models$MLE <- MLEmodel
prof.R0.summary <- SSsummarize(prof.R0.models)

# Likelihood components
mainlike_components <- c("TOTAL", "Survey", "Discard",
    "Length_comp", "Age_comp", "Recruitment")

mainlike_components_labels <- c("Total likelihood",
    "Index likelihood", "Discard", "Length likelihood",
    "Age likelihood", "Recruitment likelihood")

# plot profile using summary created above
SSplotProfile(prof.R0.summary, profile.string = "R0",
    profile.label = expression(log(italic(R)[0])),
    print = TRUE, plotdir = plotdir)
Baseval <- prof.R0.models$MLE$parameters %>%
    filter(str_detect(Label, "SR_LN")) %>%
    pull(Value)
abline(v = Baseval, lty = 2)
```

4.3 ASPM diagnostic

The application of the Age-Structured Production Model (ASPM) approach as a diagnostic can help identify misspecification of the production function. If, in the absence of composition data (likelihood weighting set to 0), the ASPM fits well to the indices of abundance that have good contrast, then the production function is likely to drive the stock dynamics and indices will provide information about the absolute abundance

(Carvalho et al. 2017). If there is not a good fit to the indices, then the catch data and production function alone cannot explain the trajectories of the indices of relative abundance. Below is a step-by-step cookbook recipe for the ASPM analysis in Stock Synthesis which can also be found in the model recipes.

4.3.1 Step 1: Create a "ASPM" subdirectory

Create a new directory for the ASPM run and a subdirectory for figures and tables

```
dirname.aspm <- "./ASPM"
dir.create(path = dirname.aspm, showWarnings = TRUE,
    recursive = TRUE)

plotdir <- file.path(dirname.aspm, "Figures_Tables")
dir.create(path = plotdir, showWarnings = TRUE, recursive = TRUE)
```

4.3.2 Step 2: Copy and paste files from reference run to ASPM directory

```
completed.model.run <- "./reference_run_orig"

list_of_files <- list.files(completed.model.run)
file.copy(file.path(completed.model.run, list_of_files),
    dirname.aspm)
```

4.3.3 Step 3: Change rec devs in ss3.pars to 0

```
pars <- SS_readpar_3.30(file.path(dirname.aspm, "ss3.par"),
    datsource = file.path(dirname.aspm, "data.ss_new"),
    ctlsource = file.path(dirname.aspm, "control.ss_new"))

pars$recdev_early[, 2] <- 0
pars$recdev1[, 2] <- 0
pars$recdev_forecast[, 2] <- 0
SS_writepar_3.30(pars, outfile = file.path(dirname.aspm,
    "ss3.par"))
```

4.3.4 Step 4: Change starter

Change the settings in the `starter.ss` file to read from par file and update the dat and ctl file names to .ss_new versions. Also, change the phase that steepness and sigmaR are estimated in.

```
starter <- SS_readstarter(file = file.path(dirname.aspm,
    "starter.ss"))
starter$init_values_src <- 1
starter$datfile <- "data.ss_new"
starter$ctlfile <- "control.ss_new"
SS_writestarter(starter, dir = dirname.aspm, overwrite = TRUE)

SS_changepars(dir = dirname.aspm, strings = c("steep",
    "sigmaR"), newphs = c(-4, -5))
```

4.3.5 Step 5: Change control file

Adjust the control file to fix rec devs at the value read from par file. Also be sure to change the phase to negative (recdev phase =-3, recdev_early_phase = -4). If there are any length or age composition data data, change the likelihood lambda to 0 (to turn off) and penalty for rec dev estimation in likelihood (lambda = 0 for recruitment). Also, manually fix the selectivity parameters to the values estimated (change phase to negative value in control.ss_new file).

```
control <- SS_readctl(file = file.path(dirname.aspm,
    "control.ss_new"), datlist = file.path(dirname.aspm,
    "data.ss_new"))


control$recdev_early_phase <- -4
control$recdev_phase <- -3

# Manually turn off all length comp data
# (likelihood lambda to 0) and penalty for rec
# dev estimation in likelihood (lambda = 0 for
# recruitment)

# 4 1 1 0 1

# 4 2 1 0 1

# 10 1 1 0 1

# If there are already lambda adjustments you can
# do this through R by:

control$lambdas$value[which(control$lambdas$like_comp ==
    4)] <- 0
control$lambdas$value[which(control$lambdas$like_comp ==
    10)] <- 0

SS_writectl_3.30(control, outfile = file.path(dirname.aspm,
    "control.ss_new"), overwrite = TRUE)
```

4.3.6 Step 6: Run original and ASPM models

4.3.7 Step 7: Summarize results and plot comparisons

```
aspm.mods <- SSgetoutput(dirvec = c(completed.model.run,
    dirname.aspm))
aspm.summary <- SSsummarize(aspm.mods)

SSplotComparisons(aspm.summary, legendlabels = c("Reference",
    "ASPM"), print = TRUE, plotdir = plotdir)

SSplotModelcomp(aspm.summary, subplots = "Index", add = TRUE,
    legendlabels = c("Full Model", "ASPM"))
SSplotModelcomp(aspm.summary, subplots = "SSB", add = TRUE)
```

```
SSplotModelcomp(aspm.summary, subplots = "RecDevs",
    add = TRUE)
```

4.4 Jittering

Jitter tests are commonly implemented in Stock Synthesis to ensure global convergence is reached. Jitter can be run using `r4ss::SS_RunJitter()`.

Below is a step-by-step cookbook recipe for the jitter analysis in Stock Synthesis which can also be found in the model recipes.

4.4.1 Step 1: Define the directory where a completed "base" model run is located

```
dirname.base <- "./reference_run"
```

4.4.2 Step 2: Create a "Jitter" subdirectory

Also create a subdirectory for the output plots.

```
dirname.jitter <- "./jitter"
dir.create(path = dirname.jitter, showWarnings = TRUE,
    recursive = TRUE)

dirname.plots <- paste0(dirname.jitter, "/plots")
dir.create(dirname.plots)
```

4.4.3 Step 3: Copy base model files to jitter directory

```
file.copy(paste(dirname.base, "starter.ss", sep = "/"),
    paste(dirname.jitter, "starter.ss", sep = "/"))
file.copy(paste(dirname.base, "em.CTL", sep = "/"),
    paste(dirname.jitter, "em.CTL", sep = "/"))
file.copy(paste(dirname.base, "ss3.DAT", sep = "/"),
    paste(dirname.jitter, "ss3.DAT", sep = "/"))
file.copy(paste(dirname.base, "forecast.ss", sep = "/"),
    paste(dirname.jitter, "forecast.ss", sep = "/"))
file.copy(paste(dirname.base, "ss.exe", sep = "/"),
    paste(dirname.jitter, "ss.exe", sep = "/"))
file.copy(paste(dirname.base, "ss.par", sep = "/"),
    paste(dirname.jitter, "ss.par", sep = "/"))
```

4.4.4 Step 4: Run Jitter

```
# set number of iterations
Njitter = 200
```

```
jit.likes <- r4ss::SS_RunJitter(mydir = dirname.base,
    Njitter = Njitter, jitter_fraction = 0.1, init_values_src = 1)
```

4.4.5 Step 5: Summarize jitter results

Save total likelihoods necessary to assess global convergence

```
x <- as.numeric(jit.likes)
global.convergence.check <- table(x, exclude = NULL)
write.table(jit.likes, paste0(dirname.plots, "/jit_like.csv"))
write.table(global.convergence.check, paste0(dirname.plots,
    "/global_convergence_check.csv"))
```

Summarize output from all runs using `r4ss::SSsummarize()`

```
jit_mods <- SSgetoutput(keyvec = 0:Njitter, getcomp = FALSE,
    dirvec = dirname.base, getcovar = FALSE)

jit_summary <- SSsummarize(jit_mods)
```

Key outputs from summarized object

```
# Likelihood across runs
likes <- jit_summary$likelihoods

# Derived quants across runs
quants <- jit_summary$quants

# Estimated parameters across runs
pars <- jit_summary$pars

# Write output tables to jitter directory
write.table(quants, paste0(dirname.plots, "/Quants.csv"))
write.table(pars, paste0(dirname.plots, "/Pars.csv"))
write.table(likes, paste0(dirname.plots, "/Likelihoods.csv"))
```

Re-tabulate total likelihoods necessary to assess global convergence and compare to jit.likes from above

```
library(magrittr)
library(dplyr)

x <- likes %>%
    filter(str_detect(Label, "TOTAL")) %>%
    select(-Label) %>%
    mutate_all(~as.numeric(.)) %>%
    unlist(use.names = FALSE)

global.convergence <- table(x, exclude = NULL)
write.table(global.convergence, paste0(dirname.plots,
    "/global_convergence.csv"))
```

Check convergence by seeing if the estimated spawning biomass is really big (+2x base spawning biomass) or really small (<0.5x base spawning biomass). *Note: this code is based on* ***check_convergence()*** *from SSMSE.*

```
converged_ssb <- jit_summary$SpawnBio %>%
    mutate(across(c(1:201), .fns = ~./replist0)) %>%
    select(-Label) %>%
    pivot_longer(col = c(1:201), names_to = "jitter",
        values_to = "SSB") %>%
    pivot_wider(names_from = Yr, values_from = SSB) %>%
    mutate(rownumber = seq(1, nrow(.))) %>%
    column_to_rownames("jitter") %>%
    filter_at(vars(1:78), all_vars((.) < 2 & (.) >
        0.5)) %>%
    select(rownumber) %>%
    pull(rownumber)
```

Check to make sure max gradient is small

```
converged_grad <- which(jit_summary$maxgrad < 0.001)
converged_jitters <- jit_mods[converged_grad]
converged_sum <- SSsummarize(converged_jitters)
```

4.4.6 Step 6: Make plots with r4ss for runs ending at a converged solution

Plot of likelihood for all jitter runs, regardless of convergence

```
library(ggplot2)

jit_summary$likelihoods %>%
    filter(str_detect(Label, "TOTAL")) %>%
    select(-Label) %>%
    pivot_longer(cols = everything(), names_to = "jitter",
        values_to = "likelihood") %>%
    separate(jitter, into = c("replist", "jitter"),
        sep = "(?<=[A-Za-z])(?=[0-9])") %>%
    mutate(jitter = as.numeric(jitter)) %>%
    ggplot(aes(x = jitter, y = likelihood)) + geom_point(size = 2) +
    geom_hline(aes(yintercept = likelihood[1]), color = "red") +
    theme_classic() + labs(y = "Total Likelihood",
    x = "Jitter runs")
ggsave(paste0(dirname.plots, "/all_likelihoods.png"))


SSplotComparisons(jit_summary, subplots = 2, pch = "",
    legend = FALSE, lwd = 1, new = F, print = TRUE,
    plotdir = dirname.plots, filenameprefix = "all_jitters_",
    ylimAdj = 1)
```

Repeat for all converged runs

```
converged_sum$likelihoods %>%
    filter(str_detect(Label, "TOTAL")) %>%
    select(-Label) %>%
    pivot_longer(cols = everything(), names_to = "jitter",
        values_to = "likelihood") %>%
    separate(jitter, into = c("replist", "jitter"),
        sep = "(?<=[A-Za-z])(?=[0-9])") %>%
    mutate(jitter = as.numeric(jitter)) %>%
    ggplot(aes(x = jitter, y = likelihood)) + geom_point(size = 3) +
    geom_hline(aes(yintercept = likelihood[1]), color = "red") +
    theme_classic() + labs(y = "Total Likelihood",
    x = "Jitter runs at a converged solution")
ggsave(paste0(dirname.plots, "/converged_likelihoods.png"))


SSplotComparisons(converged_sum, subplots = 2, pch = "",
    legend = FALSE, lwd = 1, new = F, print = TRUE,
    plotdir = dirname.plots, filenameprefix = "converged_",
    ylimAdj = 1)
```

Repeat for converged runs at the minimum solution

```
y <- which(jit_summary$likelihoods[jit_summary$likelihoods$Label ==
    "TOTAL", 1:Njitter] == min(na.omit(jit.likes)))

jit_min <- jit_mods[y]
min_sum <- SSsummarize(jit_min)

SSplotComparisons(min_sum, subplots = 2, pch = "",
    legend = FALSE, lwd = 1, new = F, print = TRUE,
    plotdir = dirname.plots, filenameprefix = "converged_min_",
    ylimAdj = 1)
```