An introduction to **ss3sim**
An R package for stock assessment simulation testing with Stock Synthesis 3

Sean C. Anderson* Cole C. Monnahan† Kelli F. Johnson‡ Kotaro Ono§
Juan L. Valero¶

*sean@seananderson.ca; Department of Biological Sciences, Simon Fraser University, Burnaby BC, V5A 1S6, Canada

†monnahc@uw.edu; Quantitative Ecology and Resource Management, University of Washington, Box 352182, Seattle, WA 98195-2182, USA

‡kfjohns@uw.edu; School of Aquatic and Fishery Sciences, University of Washington, Box 355020, Seattle, WA 98195-5020, USA

§kotarono@uw.edu; School of Aquatic and Fishery Sciences, University of Washington, Box 355020, Seattle, WA 98195-5020, USA

¶juan@myuw.net; Center for the Advancement of Population Assessment Methodology (CAPAM), 8901 La Jolla Shores Drive, La Jolla, CA 92037, USA

Contents

1 Installing the ss3sim R package

ss3sim requires R version 3.1.0 or greater. The package can be run on OS X, Windows, or Linux. The CRAN version of **ss3sim** can be installed in an R console with:

```
install.packages("ss3sim")
```

The development version of **ss3sim** can be installed from GitHub with the following code:

```
# Install devtools first:
install.packages("devtools")

# Install ss3sim:
devtools::install_github("ss3sim/ss3sim", dependencies = TRUE)

# If you would like to run simulations in parallel, also install:
install.packages(c("doParallel", "foreach"))

# This vignette uses ggplot2 to make plots, you can install it with:
install.packages("ggplot2")
```

You can then load the package with:

```
library(ss3sim)
```

You can read the help files and access this vignette again with:

```
?ss3sim
help(package = "ss3sim")
vignette("ss3sim-vignette")
```

See Anderson et al. (2014) for an overview of **ss3sim**.

We develop and test **ss3sim** around a specific version of SS3. Currently, **ss3sim** is tested to work with SS3 Version 3.24O. Since this is a more recent version than is provided at the normal download location http://nft.nefsc.noaa.gov/SS3.html, we provide copies of the 3.24O binaries (with the permission of R. Methot) at https://www.dropbox.com/sh/zg0sec6j20sfyyz/AACQiuk787qW882U2euVKoPna.

ss3sim requires the SS3 binary to be in your path. This means that your operating system knows where the binary file is located. See Section 14 for details. See `?run_ss3model` for instructions on how to specify the safe vs. the optimized binary.

Please direct any questions, suggestions, or bug reports to the **ss3sim** issue tracker: https://github.com/ss3sim/ss3sim/issues, or to an author as listed on the title page of this vignette.

If you use **ss3sim** in a publication, please cite the package as indicated by running `citation("ss3sim")` in the R console:

```
citation("ss3sim")


##
## To cite ss3sim in publications use:
##
##   Anderson, SC, Monnahan, CC, Johnson, KF, Ono, K, Valero, JL,
##   Cunningham, CJ, Hurtado-Ferro, F, Licandeo, R, McGilliard, CR,
##   Szuwalski, CS, Vert-pre, KA, and Whitten, AR (2014). ss3sim:
##   Fisheries stock assessment simulation testing with Stock
##   Synthesis. R package version 0.8.2.99.
##
##   Anderson, SC, Monnahan, CC, Johnson, KF, Ono, K, and Valero, JL
##   (2014). ss3sim: An R package for fisheries stock assessment
##   simulation with Stock Synthesis. PLOS ONE. 9(4): e92725. DOI:
##   10.1371/journal.pone.0092725.
```

2 An overview of the ss3sim simulation structure

ss3sim is an R package to make it relatively quick and easy to run simulations with the 3rd version of Stock Synthesis, SS3. The package consists of a series of low-level functions that facilitate manipulating SS3 configuration files, running SS3 models, and combining the output. **ss3sim** also contains some wrapper functions that tie all these low-level functions together into a complete simulation experiment. If you choose to use our wrapper function `run_ss3sim` then you will use a series of plain text control files to control the simulation. Much of this vignette focusses on how to effectively use `run_ss3sim`, but feel free to take the low-level functions (`change` functions) and use them as part of your own flexible simulation.

2.1 Setting up the file structure

The **ss3sim** package is set up assuming there is an established base-case operating model (OM) and estimation model (EM) to work with. The **ss3sim** package comes with three

generic built-in model setups that can be used as is, modified, or replaced.[1] Each OM and EM should be in its own folder. The OM folder should have the files:

```
yourOMmodel.ctl
yourOMmodel.dat
ss3.par
starter.ss
forecast.ss
```

The EM folder should have:

```
yourEMmodel.ctl
starter.ss
forecast.ss
```

In both cases, nothing more and nothing less. The names of the `.ctl` and `.dat` files are not important. The package functions will rename them after they are copied to appropriate folders. These files should be derived from `.ss_new` files but with file extensions as shown above. It is important to use `.ss_new` files so the files have consistent formatting. Many of the functions in this package depend on that formatting.

To obtain `.ss_new` files, open a command window in the OM and EM folders and type `ss3` to run the models. Once the model is finished running, it will generate a number of files, including `.ss_new` files. Remove the `.ss_new` file extension from the files needed and add the appropriate file extension.

2.2 Cases and scenarios

The high-level wrapper function `run_ss3sim` uses unique case identifiers (IDs) that combine to create unique scenarios. The types of cases are: data quality (D), estimation (E), fishing mortality (F), retrospective (R), and any other letter describing a time varying case (e.g. M for natural mortality, S for selectivity, or G for growth). These case IDs are followed by an alphanumeric stock or species identifier (e.g. `cod`). The different versions of each case are identified with numbers. For example, the base-case scenario for a cod stock might be: `D0-E0-F0-M0-R0-cod`. The order of the cases does not matter, as long as they are used consistently.

ss3sim relies on a set of plain text files to control the simulation (Figure 1). These plain text files are read by `get_caseval` and turned into argument lists that are passed

[1] See Section 8 for details on these models and how to modify them. See Section 15 and 16 for details on creating your own OM and EMs.

to `run_ss3sim`. The function `create_argfiles` creates template input files. It reads the various functions and parses the arguments and default values into plain text files. The default settings create these files:

1. `E0-spp.txt` (for estimation method cases)
2. `F0-spp.txt` (for fishing mortality cases)
3. `R0-spp.txt` (for retrospective cases)
4. `index0-spp.txt` (controlled by the `D` (data) case)
5. `agecomp0-spp.txt` (controlled by the `D` (data) case)
6. `lcomp0-spp.txt` (controlled by the `D` (data) case)
7. `X0-spp.txt` (for a time-varying parameter X, where X could be any time-varying case letter)

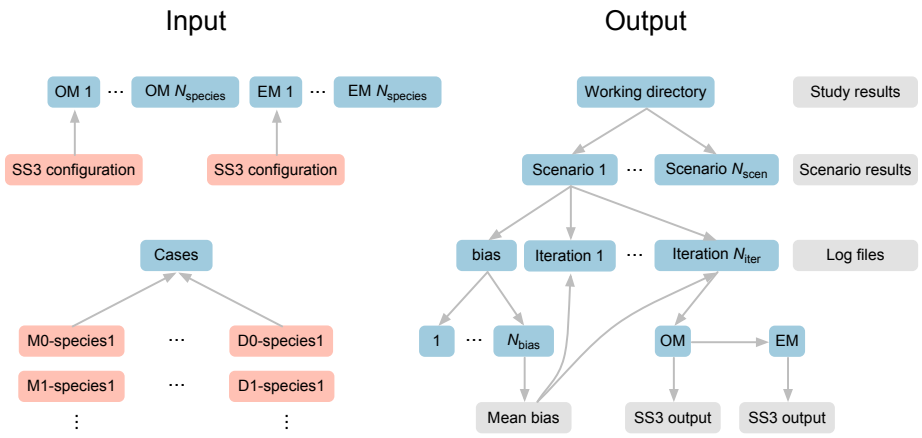

Figure 1: Illustration of input and output folder and file structure for an **ss3sim** simulation. Folders are shown in blue, input files in orange, and output files in grey. All input and output files are in plain text format. Case files (orange files at bottom left) combine cases (e.g. M0 for a given natural mortality trajectory) with species or life-history OMs and EMs (e.g. cod-like or sardine-like). Alternatively, a user can skip setting up case files and specify the simulation cases directly in R code (see section 4.

After running `create_argfiles()`, look in your working directory for the template input files.[2] Change the case ID number (defaults to `0`) and the species identifier to a three

[2] You can use `getwd()` to see your current working directory or `setwd()` to set a different working directory.

letter identifier. For example, you might use `cod`, `sar`, or `fla` for cod, sardine, or flatfish. An example filename would therefore be `M1-sar.txt` or `lcomp2-fla.txt`. The case `D1` corresponds to the files `index1-spp.txt`, `agecomp1-spp.txt`, and `lcomp0-spp.txt`. The other case types have single argument files.

The first column in the text files denotes the argument to be passed to a function. The second argument denotes the value to be passed. See the help for a `change` function to see the arguments that need to be declared. For example, see `?change_f`.

You can use any R syntax to declare argument values. For example, `c(1, 2, 4)`, or `seq(1, 100)`. Character objects do not need to be quoted as long as they are one word, but can be if you would like. However, be careful not to use the delimiter (set up as a semicolon) anywhere else in the file besides to denote columns. You can add comments after any `#` symbol.

Putting that all together, below is what an example `F1-cod.txt` file might look like:

```
years; 1913:2012
years_alter; 1913:2012
fvals; c(rep(0, 25), rep(0.114, 75))
```

2.3 Case file names

Table 1: Function names and associated case files. Case file names are given as the prefix. For example a case file for `sample_index` might be `index0-cod.txt`.

ss3sim function	Case file name	Mandatory?
`change_f()`	`F`	Yes
`change_e()`	`R`	Yes
`sample_index()`	`index`	Yes
`sample_lcomp()`	`lcomp`	Yes
`sample_agecomp()`	`agecomp`	Yes
`change_tv()`	anything, see below	No
`change_tail_compression()`	`tail_compression`	No
`change_lcomp_constant()`	`lcomp_constant`	No
`sample_calcomp()`	`calcomp`	No
`sample_wtatage()`	`wtatage`	No

`change_tv()` can take any other case file name, as long as the first line is `function_type; change_tv`.

2.4 Bias adjustment

Bias adjustment helps assure that the estimated log-normally distributed recruitment deviations are mean-unbiased leading to mean-unbiased estimates of biomass (Methot and Taylor 2011). The high-level wrapper function `run_ss3sim` allows users to specify whether or not they would like to use the bias adjustment routine built into the package by setting the argument `bias_adjust` to `TRUE` or `FALSE`. If `TRUE`, the function runs `bias_nsim` replicates for each scenario (located in the subfolder `bias` within a scenario folder) and then averages the bias-adjustment parameter estimates. The default number of bias-adjustment replicates is five. If a bias-adjustment replicate fails to converge (checked by identifying whether the Hessian was invertible), the parameter estimates from that replicate are ignored. A minimum threshold of 80% converged bias-adjustment replicates (adjustable via the `conv_crit` argument) is used to ensure reliable parameter estimates. The mean bias adjustment runs are then used in the EM for all subsequent "iterations" (i.e. replicates using same models but different process and observation errors) within that scenario. We assume bias adjustment parameters applicable to all iterations within the scenario are more likely to be found by averaging across multiple sets of parameters.

The bias adjustment process creates several files in the `bias` folder (for each scenario) which can be examined by the user. `AdjustBias.DAT` contains the calculated adjustment parameters for each bias replicate, and `AvgBias.DAT` contains the average of these estimates which are used in all subsequent simulations for that scenario. Individual plots for each successful bias adjustment replicate are also created. If `bias_adjust` is set to `FALSE`, no bias adjustment replicates are executed and no bias adjustment is performed.

2.5 Output file structure

Internally, the function `copy_ss3models` creates a folder structure and copies the operating and estimation models (Figure 1). The folder structure looks like:

```
D0-E0-F0-M0-R0-cod/1/om
D0-E0-F0-M0-R0-cod/1/em
D0-E0-F0-M0-R0-cod/2/om
D0-E0-F0-M0-R0-cod/2/em
...
```

The integer values after the scenario ID represent different iterations; the total number of iterations run are specified by the user. If you are using bias adjustment (`bias_adjust = TRUE`) then there will be some additional folders. In that case the folders will look like:

```
D0-E0-F0-M0-R0-cod/bias/1/om
```

```
D0-E0-F0-M0-R0-cod/bias/1/em
D0-E0-F0-M0-R0-cod/bias/2/om
D0-E0-F0-M0-R0-cod/bias/2/em
...
D0-E0-F0-M0-R0-cod/1/om
D0-E0-F0-M0-R0-cod/1/em
D0-E0-F0-M0-R0-cod/2/om
D0-E0-F0-M0-R0-cod/2/em
...
```

Note that the OM and EM folders will be renamed `om` and `em` within each iteration, the OM and EM are checked to make sure they contain the minimal files (as listed above in Section 2.2), the filenames will be all lowercase, the data file is renamed `ss3.dat`, the control files are renamed `om.ctl` or `em.ctl`, and the starter and control files are adjusted to reflect these new file names. The functions in this package assume you have set your working directory in R to be the base folder where you will store the scenario folders.

3 An example simulation with ss3sim

As an example, we will run a 2x2 simulation design in which we test (1) the effect of high and low precision on the index of abundance research survey and (2) the effect of fixing versus estimating natural mortality (M). All of the required files for this example are contained within the **ss3sim** package. To start, we will locate three sets of folders: the folder with the plain text case files, the folder with the OM, and the folder with the EM.

```
library(ss3sim)
d <- system.file("extdata", package = "ss3sim")
case_folder <- paste0(d, "/eg-cases")
om <- paste0(d, "/models/cod-om")
em <- paste0(d, "/models/cod-em")
```

See the folder `ss3sim/inst/extdata/eg-cases` inside the **ss3sim** source code for all the case files that are used in this example simulation. You can either download the source code from https://github.com/ss3sim/ss3sim or find this folder at the location contained in the `case_folder` object defined in the block of R code above this paragraph.

3.1 Creating the case files

We will base the simulation around the base-case files created for a cod-like species in the papers Ono et al. (2014) and Johnson et al. (2014), both of which used the **ss3sim** package.

You can refer to these papers for details on how the models were set up.

If we were starting from scratch, we would run the function `create_argfiles()`, which would create a default set of configuration files in our R working directory. Instead we will start with the configuration files from those papers.

To investigate the effect of different levels of precision on the survey index of abundance, we will manipulate the argument `sds_obs` that gets passed to `sample_index`. This argument ultimately refers to the standard error on the log(index) value, as defined in SS3. In case 0, we will specify the standard deviation at 0.1 and in case 1 we will increase the standard deviation to 0.4. We can do this by including the line: `sds_obs; 0.1` in the file `D0-cod.txt` and the line: `sds_obs; 0.4` in the file `D1-cod.txt`.

The file `index0-cod.txt` will therefore look like:

```
fleets;  2
years;   list(seq(1974, 2012, by = 2))
sds_obs; list(0.1)
```

`fleets` refers to the fleet number we want to sample from (as defined in the model), `2` in `codOM.dat` refers to the survey fleet. We then run our survey from 1974 to 2012 with a standard error on the log(survey index) of 0.1.

We will not describe the length or age composition sampling here in detail, but you can refer to the help files `?sample_lcomp` and `?sample_agecomp` along with the case files included in the package data. Also see Section **??** where we describe the theory behind the age- and length-composition sampling in **ss3sim**.

To investigate the effect of fixing versus estimating M, we will manipulate the argument `natM_val` that gets passed to `change_e`. The first entry of `natM_val` is the value which M is fixed or initialized at and the second refers the phase. In case 0, we will set the phase in which M is estimated to `-1` (any negative phase number will tell SS3 to not estimate a particular parameter) and use the argument `NA` to fix M at the true value (i.e. do not modify it from what is in the EM `.ctl` file). In case 1, we will estimate M in phase `3` and initialize the estimation of M at `0.20`. We can do this by including the line: `natM_val; c(NA, -1)` in the file `E0-cod.txt` and the line: `natM_val; c(0.20, 3)` in the file `E1-cod.txt`. This means that case `E1` will initialize M at `0.20` and estimate M in the third phase. For example, the complete case file `E0-cod.txt` is:

```
natM_type;           1Parm
natM_n_breakpoints; NULL
natM_lorenzen;       NULL
natM_val;            c(NA,-1)
par_name;            LnQ_base_3_CPUE
```

```
par_int;            NA
par_phase;          -1
forecast_num;       0
```

Since we will not be running a retrospective analysis in this example, this will be our `R0-cod.txt` file:

```
retro_yr; 0
```

We will set the fishing mortality F to a constant level at F_{MSY} (F at maximum sustainable yield) from when the fishery starts (in year 25) until the end of our simulation (year 100). Therefore, this is our `F0-cod.txt`:

```
years;       1913:2012
years_alter; 1913:2012
fvals;       c(rep(0,25), rep(0.114,75))
```

Although we do not have any time-varying processes in this example simulation, we include `M` case files in which we describe M as stationary (`M0-cod.txt`):

```
function_type; change_tv
param;         NatM_p_1_Fem_GP_1
dev;           rep(0, 100)
```

Simply by changing the deviations from a vector of zeros to any vector pattern of deviations, we could add time-varying M. In Section 6 we describe how time-varying parameters are unique in **ss3sim** and how you can make any parameter described in SS3 time varying using this approach.

3.2 Checking the case files

It is a good idea to check that the case files are manipulating the SS3 model files as intended. One way we can do this is by running the `change` functions directly on the SS3 model files using the arguments specified within the case files and inspecting the modified SS3 files. Another way is to run a single iteration of your simulation through `run_ss3sim` and carefully inspect the OM and EM model files that it creates. For example, we could run:

```
run_ss3sim(iterations = 1, scenarios =
  c("D0-E0-F0-R0-M0-cod",
    "D1-E0-F0-R0-M0-cod",
    "D0-E1-F0-R0-M0-cod",
    "D1-E1-F0-R0-M0-cod"),
  case_folder = case_folder, om_dir = om,
  em_dir = em)
```

And then check the SS3 files that are created.

3.3 Self testing: deterministic simulations to check the models for bias

Self testing is a crucial step of any simulation study. One way to accomplish this is to set up an EM that is similar to the OM and include minimal error.

We will run some simulations to check our EM for bias when process and observation error are minimized. To do this, we will start by setting up a 100 row (number of years) by 20 column (number of iterations) matrix of recruitment deviations, where all values are set to zero.

```
recdevs_det <- matrix(0, nrow = 100, ncol = 20)
```

Then we will set up case "estimation" files in which the initialized values of the recruitment deviation standard deviations, `SR_sigmaR`, are set to the nominal level of `0.001`. We will name these files `E100-cod.txt` and `E101-cod.txt`. In the case files, the key element is setting `par_name = SR_sigmaR` (the SS3 parameter name) and `par_int = 0.001` (the initial value). The arguments `par_name` and `par_int` are set up to handle vectors of parameter names and values. In this example, changing `SR_sigmaR` will be the second parameter in the vector. Again, you can look at all the case files in the package folder `/inst/extdata/eg-cases/`. As an example, here is the file `E101-cod`:

```
natM_type;           1Parm
natM_n_breakpoints; NULL
natM_lorenzen;       NULL
natM_val;            c(0.20,3)
par_name;            LnQ_base_3_CPUE,SR_sigmaR
par_int;             c(NA,0.001)
par_phase;           c(-1,-1)
forecast_num;        0
```

To minimize observation error on the survey index, we will create a data case 100 that has a standard deviation on the survey observation error of 0.001. Therefore, our case file `index100-cod.txt` will look like this:

```
fleets;  2
years;   list(seq(1974, 2012, by = 2))
sds_obs; list(0.001)
```

When we run the simulations, we will pass our deterministic recruitment deviations to the function `run_ss3sim`. Running 20 iterations should be enough to identify whether our models are performing as we expect. Note that by default 5 bias adjustment runs are performed since we specify that `bias_adjust=TRUE`.

```
run_ss3sim(iterations = 1:20,
  scenarios = c("D100-E100-F0-R0-M0-cod", "D100-E101-F0-R0-M0-cod"),
  case_folder = case_folder, om_dir = om, em_dir = em,
  bias_adjust = TRUE, user_recdevs = recdevs_det)
```

We have written out the scenario names in full for clarity, but **ss3sim** also contains a convenience function `expand_scenarios`. With this function we could instead write:

```
x <- expand_scenarios(list(D = 100, E = 100:101, F = 0, M = 0,
    R = 0), species = "cod")
run_ss3sim(iterations = 1:20, scenarios = x,
  case_folder = case_folder, om_dir = om, em_dir = em,
  bias_adjust = TRUE, user_recdevs = recdevs_det)
```

Note that due to the way that SS3 is being used as an operating model, you may see an ADMB error in the console:

```
Error -- base = 0 in function prevariable& pow(const prevariable& v1,
  CGNU_DOUBLE u)
```

However, this is not a problem since ADMB is not used to optimize the OM — the error can safely be ignored.

3.4 Running the stochastic simulations

The package contains a set of normally-distributed recruitment deviations, with a mean of `-0.5` and a standard deviation of `1` (bias-corrected standard-normal deviations). To

use the pre-specified deviations remove the argument `user_recdevs` from `run_ss3sim`. Process error will now be unique for each iteration but consistent across scenarios for a specific iteration, to facilitate comparison between scenarios.

We will only run 100 iterations here to save time and keep the package size down, but in a real simulation testing study you may want to run many more iterations, perhaps testing how many iterations are required before the key results stabilize.

```
run_ss3sim(iterations = 1:100, scenarios =
  c("D0-E0-F0-R0-M0-cod",
    "D1-E0-F0-R0-M0-cod",
    "D0-E1-F0-R0-M0-cod",
    "D1-E1-F0-R0-M0-cod"),
  case_folder = case_folder, om_dir = om,
  em_dir = em, bias_adjust = TRUE)
```

3.5 Reading in the output and plotting the results

The function `get_results_all` reads in a set of scenarios and combines the output into two `.csv` files: `ss3sim_scalar.csv` and `ss3sim_ts.csv`. The `scalar` file contains values for which there is a single estimated value (e.g. MSY) and the `ts` file refers to values for which there are time series of estimates available (e.g. biomass for each year). The column names refer to the output from SS3, so it can be useful to consult the SS3 manual for details.

```
get_results_all(user_scenarios =
  c("D100-E100-F0-R0-M0-cod",
    "D100-E101-F0-R0-M0-cod",
    "D0-E0-F0-R0-M0-cod",
    "D1-E0-F0-R0-M0-cod",
    "D0-E1-F0-R0-M0-cod",
    "D1-E1-F0-R0-M0-cod"))
```

We will read in the `.csv` files:

```
scalar_dat <- read.csv("ss3sim_scalar.csv")
ts_dat <- read.csv("ss3sim_ts.csv")
```

Or if you would like to follow along with the rest of the vignette without running the simulations above, you can load a saved version of the output:

```
data("ts_dat", package = "ss3sim")
data("scalar_dat", package = "ss3sim")
```

First, we will calculate some useful values in new columns and separate the deterministic from the stochastic simulation runs:

```
scalar_dat <- transform(scalar_dat,
  steep = (SR_BH_steep_om - SR_BH_steep_em)/SR_BH_steep_om,
  logR0 = (SR_LN_R0_om - SR_LN_R0_em)/SR_LN_R0_om,
  depletion = (depletion_om - depletion_em)/depletion_om,
  SSB_MSY = (SSB_MSY_em - SSB_MSY_om)/SSB_MSY_om,
  SR_sigmaR = (SR_sigmaR_em - SR_sigmaR_om)/SR_sigmaR_om,
  NatM =
    (NatM_p_1_Fem_GP_1_em - NatM_p_1_Fem_GP_1_om)/
     NatM_p_1_Fem_GP_1_om)

ts_dat <- transform(ts_dat,
  SpawnBio = (SpawnBio_em - SpawnBio_om)/SpawnBio_om,
  Recruit_0 = (Recruit_0_em - Recruit_0_om)/Recruit_0_om)
ts_dat <- merge(ts_dat, scalar_dat[,c("scenario", "replicate",
    "max_grad")])

scalar_dat_det <- subset(scalar_dat, E %in% c("E100", "E101"))
scalar_dat_sto <- subset(scalar_dat, E %in% c("E0", "E1"))
ts_dat_det <- subset(ts_dat, E %in% c("E100", "E101"))
ts_dat_sto <- subset(ts_dat, E %in% c("E0", "E1"))
```

Now we will turn the scalar data into long-data format so we can make a multipanel plot with the R package **ggplot2**.

```
scalar_dat_long <- reshape2::melt(scalar_dat[,c("scenario", "D", "E",
  "replicate", "max_grad", "steep", "logR0", "depletion", "SSB_MSY",
  "SR_sigmaR", "NatM")], id.vars = c("scenario", "D", "E",
  "replicate", "max_grad"))
scalar_dat_long <- plyr::rename(scalar_dat_long,
  c("value" = "relative_error"))
```

Now we can create boxplots of the deterministic model runs. The following code produces Figure 2.

```
library(ggplot2)
p <- ggplot(subset(scalar_dat_long, E %in% c("E100", "E101") &
       variable != "SR_sigmaR"), aes(D, relative_error)) +
     geom_boxplot() +
     geom_hline(aes(yintercept = 0), lty = 2) +
     facet_grid(variable~E) +
     theme_bw() + ylim(-0.4, 0.4)
print(p)
```

Let's look at the relative error in estimates of spawning biomass. We will colour the time series according to the maximum gradient. Small values of the maximum gradient (approximately 0.001 or less) indicate that model convergence is likely. Larger values (greater than 1) indicate that model convergence is unlikely. Results of individual iterations are jittered around the vertical axis to aid in visualization. The following three blocks of code produce Figures 3, 4, and 5.

```
p <- ggplot(ts_dat_sto, aes(x = year)) + xlab("Year") +
    theme_bw() + geom_line(aes(y = SpawnBio, group = replicate,
    colour = max_grad), alpha = 0.3, width = 0.15) + facet_grid(D~E) +
    scale_color_gradient(low = "gray", high = "red")
print(p)
```

```
p <- ggplot(ts_dat_sto, aes(year, SpawnBio_em, group = replicate)) +
  geom_line(alpha = 0.3, aes(colour = max_grad)) + facet_grid(D~E) +
  scale_color_gradient(low = "darkgrey", high = "red") + theme_bw()
print(p)
```

```
p <- ggplot(subset(scalar_dat_long, E %in% c("E0", "E1")),
       aes(D, relative_error)) +
     geom_boxplot() + geom_hline(aes(yintercept = 0), lty = 2) +
     facet_grid(variable~E) +
     geom_jitter(aes(colour = max_grad),
       position = position_jitter(height = 0, width = 0.1),
       alpha = 0.4, size = 1.5) +
     scale_color_gradient(low = "darkgrey", high = "red") +
     theme_bw()
print(p)
```

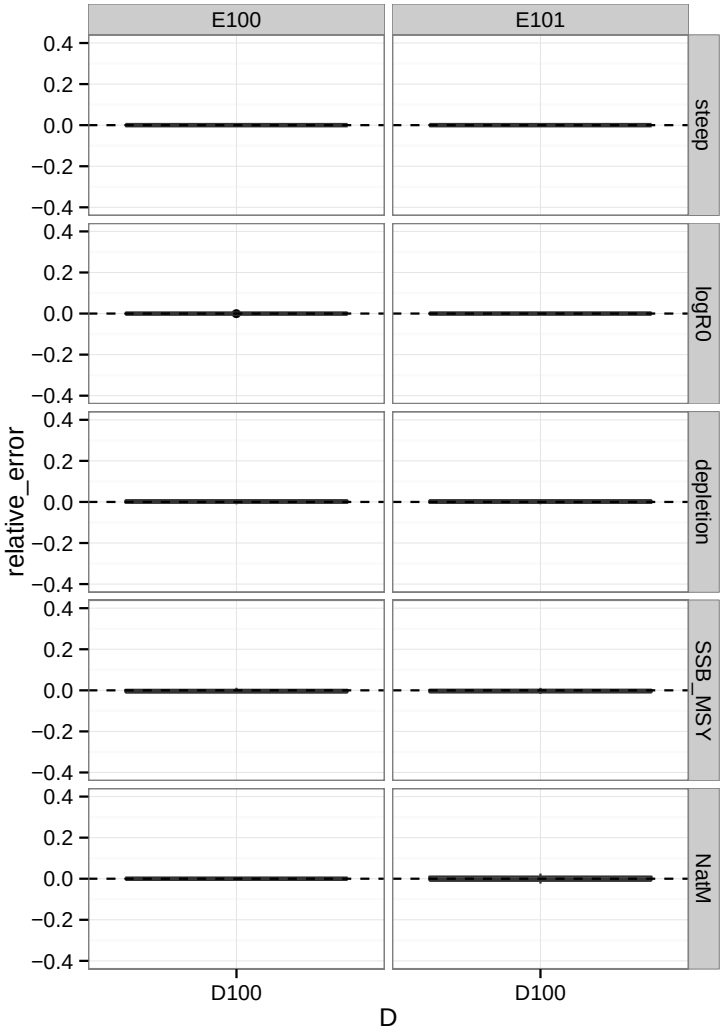

Figure 2: Relative error box plots for deterministic runs. In case E100, M is fixed at the true value; in E101 we estimate M. In case D100, the standard deviation on the survey index observation error is 0.001.

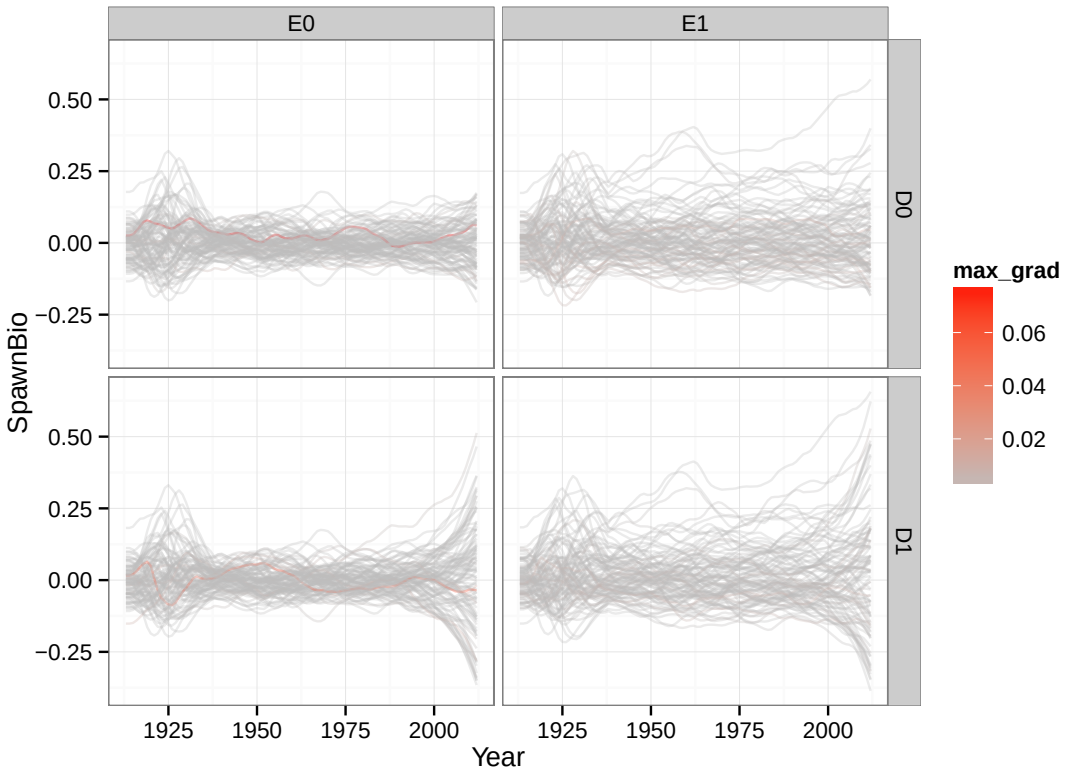

Figure 3: Time series of relative error in spawning stock biomass.

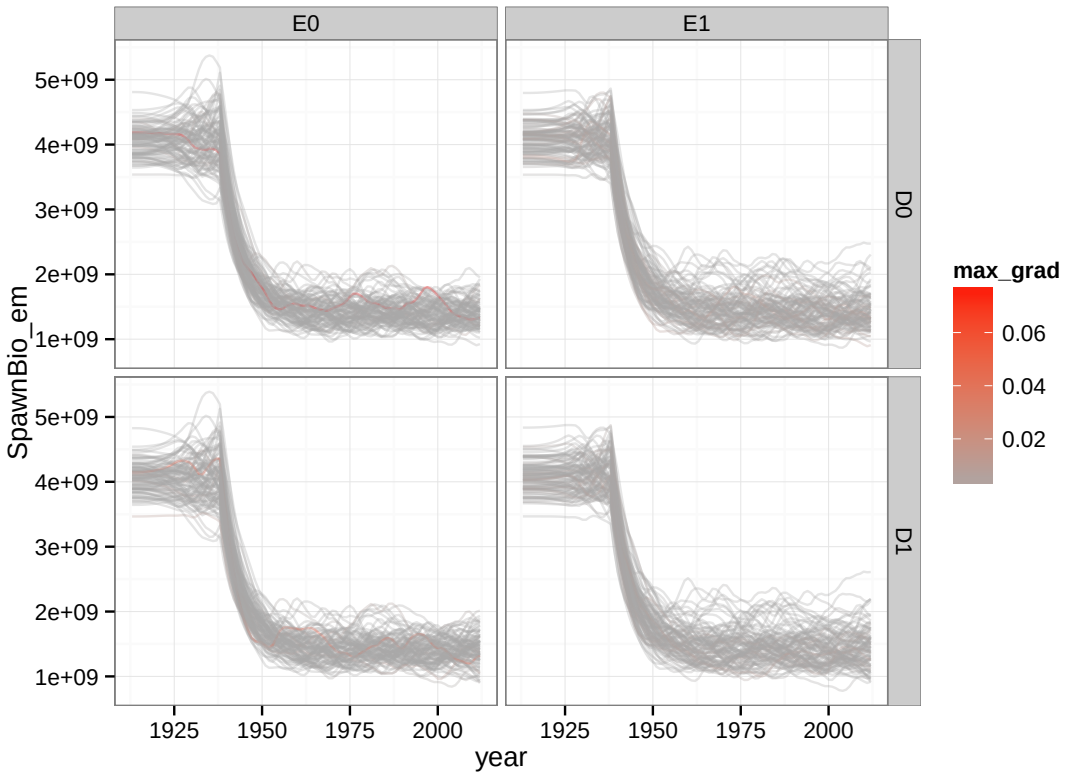

Figure 4: Spawning stock biomass time series.

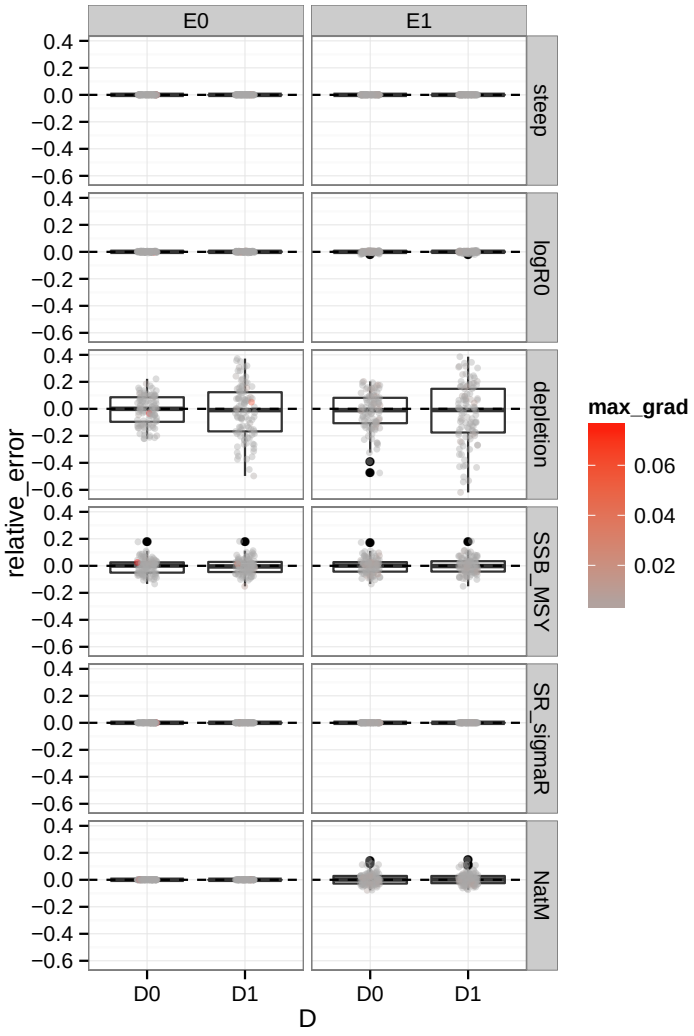

Figure 5: Relative error box plots for stochastic runs. In case E0, M is fixed at the true value; in E1 we estimate M. In case D1, the standard deviation on the survey index observation error is 0.4. In case D0, the standard deviation is quartered representing an increase in survey sampling effort.

4 Using ss3sim_base directly

An alternative approach to using **ss3sim** is to skip setting up the case files and use `ss3sim_base` directly by passing lists of arguments. The lists correspond to the arguments to each of the `change` or `sample` functions without any references to the files that need to be modified. (Although if you passed the file names they would just be ignored.) The documentation for each function includes an asterisk beside the arguments that you can specify. Note that if you do not include an argument then `ss3sim_base` will assume the value of that argument is `NULL`.

When we call `ss3sim_base` directly, the scenario ID only serves the function of identifying the output folder name and could technically be any character string. For example, we could have run the scenario `D1-E0-F0-R0-M0-cod` that we ran before:

```
d <- system.file("extdata", package = "ss3sim")
om <- paste0(d, "/models/cod-om")
em <- paste0(d, "/models/cod-em")

F0 <- list(years = 1913:2012, years_alter = 1913:2012, fvals =
  c(rep(0, 25), rep(0.114, 75)))

index1 <- list(fleets = 2, years = list(seq(1974, 2012, by = 2)),
  sds_obs = list(0.1))

lcomp1 <- list(fleets = c(1, 2), Nsamp = list(100, 100), years =
  list(1938:2012, seq(1974, 2012, by = 2)), lengthbin_vector = NULL,
  cpar = c(1, 1))

agecomp1 <- list(fleets = c(1, 2), Nsamp = list(100, 100), years =
  list(1938:2012, seq(1974, 2012, by = 2)), agebin_vector = NULL,
  cpar = c(1, 1))

E0 <- list(natM_type = "1Parm", natM_n_breakpoints = NULL,
  natM_lorenzen = NULL, natM_val = c(NA,-1), par_name =
  "LnQ_base_3_CPUE", par_int = NA, par_phase = -1, forecast_num = 0)

M0 <- list(NatM_p_1_Fem_GP_1 = rep(0, 100))

R0 <- list(retro_yr = 0)

ss3sim_base(iterations = 1:20, scenarios = "D1-E0-F0-R0-M0-cod",
  f_params = F0, index_params = index1, lcomp_params = lcomp1,
  agecomp_params = agecomp1, estim_params = E0, tv_params = M0,
```

```
  retro_params = R0, om_dir = om, em_dir = em)
```

5 An example of customizing the case specifications

You do not need to stick with the scenario ID scheme laid out by default in **ss3sim**. You are free to choose your own scheme by adjusting the list of values passed to the `case_files` argument in `run_ss3sim()`.

Below is an example where we'll specify the arguments to the sampling functions via their own case files. We'll arbitrarily call these cases X, Y, and Z.

We will start by copying over some case arguments built into the package. We will copy what was called `index0-cod.txt`, `lcomp0-cod.txt`, and `agecomp0-cod.txt` into `X-cod.txt`, `Y-cod.txt`, and `Z-cod.txt`. Then we'll set up our custom `case_files` list and call `run_ss3sim()`:

```
case_folder <- system.file("extdata", "eg-cases", package = "ss3sim")
om <- system.file("extdata", "models/cod-om", package = "ss3sim")
em <- system.file("extdata", "models/cod-em", package = "ss3sim")
files <- list.files(case_folder, pattern = "0-cod")

temp_case_folder <- "ss3sim-example-cases"
dir.create(temp_case_folder)
file.copy(paste0(case_folder, "/", files), temp_case_folder)

# now make X, Y, and Z case files:
setwd(temp_case_folder)
file.copy("index0-cod.txt", "X0-cod.txt")
file.copy("lcomp0-cod.txt", "Y0-cod.txt")
file.copy("agecomp0-cod.txt", "Z0-cod.txt")
setwd("..")

# our custom specification:
case_files <- list(M = "M", F = "F", R = "R", E = "E",
  X = "index", Y = "lcomp", Z = "agecomp")

# and use run_ss3sim() with our new case_file list:
run_ss3sim(iterations = 1,
  scenarios = "X0-Y0-Z0-E0-F0-R0-M0-cod",
  case_folder = temp_case_folder,
  om_dir = om, em_dir = em,
```

```
  case_files = case_files)
```

6 Time-varying parameters in the OM

The **ss3sim** package includes the capability for inducing time-varying changes in parameters in the OM. The package currently does not have built-in functions to turn on/off the estimation of time-varying parameters, so this cannot be done with case arguments. However, it is possible to create versions of an EM with and without time-varying estimation of a parameter. This approach would allow for testing of differences between estimating a single, constant parameter, vs. a time-varying estimate.

You can specify any parameter defined in your OM model as time varying through the `change_tv` function. The function works by adding an environmental deviate (env) to the base parameter (par), creating a time varying parameter (par') for each year (y),

$$par'_y = par + link * env_y. \tag{1}$$

The $link$ is pre-specified to a value of one and par is the base value for the given parameter, as defined in the `.par` file. For all catchability parameters (q), the deviate will be added to the log transform of the base parameter using the following equation:

$$\log(q'_y) = \log(q) + link * env_y. \tag{2}$$

The vector of deviates must contain one value for every year of the simulation and can be specified as zero for years in which the parameter does not deviate from the base parameter value.

Currently, `change_tv` function only works to add time-varying properties to a time-invariant parameter. It cannot alter the properties of parameters that already vary with time. Also, it will not work with custom environmental linkages. Environmental linkages for all parameters in the OM must be declared by a single line (i.e. `0 #_custom_mg-env_setup (0/1)` prior to using `change_tv`. Additionally, SS3 does not allow more than one stock recruit parameter to vary with time. Therefore, if the `.ctl` file already has a stock recruit parameter that varies with time and you try to implement another, the function will fail.

Since the `change_tv` function can be used for a number of purposes, it interacts differently with the case files than the other `change` functions. To pass arguments to `change_tv` through `run_ss3sim` you need to: (1) create a case file with an arbitrary letter not used elsewhere (i.e. anything but D, E, F, or R) and (2) include the line:

```
function_type; change_tv
```

in your case file. For example, you might want to use M for natural mortality, S for selectivity, or G for growth.

7 Incorporating a new case ID

ss3sim is designed with a set of mandatory case IDs that must be specified for each run: data (`D`), estimation (`E`), fishing effort (`F`),and retrospective (`R`). Also, by default, `run_ss3sim` is set up to require a natural mortality (`M`) case. In many instances a user may wish to include a new case ID. This section is designed to explain how to accomplish this within the **ss3sim** framework. Note that as currently developed, **ss3sim** incorporates all new cases by using the `change_tv` function, even if the changes are not time-varying. However, you can simply pass a constant vector of values to induce a constant change in a parameter.

As an example, you might want to incorporate changes in selectivity (time-varying or not) in the OM to more accurately mimic the behaviour of a real fishery. Using the help file for `change_tv`, we can create a set of "S" (for selectivity) case argument files which contain the changes required. You may choose to only have a "base case" which applies to all scenarios in the simulation test, or you may have multiple cases to investigate how different patterns of selectivity in the OM affect the EM. Now that the case argument files have been created, you need to tell **ss3sim** to use them, since they are outside of the mandatory set. This is done through the `case_files` argument in the `get_caseargs` function, which is passed through by the `run_ss3sim`. In this case, we need to tell it to look for case argument files beginning with "S" (see the help file for `get_caseargs` for more information):

```
case_files = list(E = "E", D = c("index", "lcomp", "agecomp"),
  F = "F", M = "M", R = "R", S = "S")
```

Note that `S` is simply added to the end of the default list of case IDs.

Note that the data case ID is a special case because its case argument files do not begin with `D`. This is because there are three functions associated with the data case, independently modifying three different parts of the `.dat` file.

This provides a template for how you could incorporate multiple changes to selectivity (for example) through different functions (which you would need to write yourself). For example, you may want to add time-varying deviations to multiple parameters in the selectivity setup of the OM. You could then create functions that modify selectivity in different ways, and pass them the same way the `D` case is handled above.

We recommend verifying the functionality of any new cases by: (1) thoroughly verifying that your R functions work outside of **ss3sim**, and then (2) running a single iteration and manually checking the produced files. If you write external functions for manipulating new aspects of SS3 models, please contact the **ss3sim** package developers for potential inclusion in future versions.

8 Modifying the OM and EM models included with ss3sim

ss3sim comes with three built-in SS3 operating and estimation model setups: a cod-like (slow-growing and long-lived), flatfish-like (fast-growing and long-lived), and sardine-like (fast-growing and short-lived). These model setups are based on North Sea cod (*Gadus morhua*, R. Methot, pers. comm.), yellowtail flounder (*Limanda ferruginea*, R. Methot, pers. comm.), and Pacific sardine (*Sardinops sagax caeruleus;* Hill et al. 2012). Further details on these models are available in Johnson et al. (2014) and Ono et al. (2014). These models were stripped down and simplified to make them more generic for simulation testing. In doing this, we removed many of the subtle features of the model setups. While these model setups are generic and cover a wide range of life history types, they may not be suitable for all users. Therefore, in this section, we outline strategies for modifying the existing SS3 models.

Before proceeding it is worth considering the scope and place of **ss3sim** as a simulation package. The package was designed as a tool for examining structural differences in alternative model setups. These differences could be between an OM and EM (e.g. Johnson et al. 2014) or between multiple EMs (e.g. Ono et al. 2014). Therefore, the specific details (e.g. many fleets, tagging data, seasons, etc.) of the original model setups were not important and removed to produce a set of generic life-history-type models. **ss3sim** is not designed for testing arbitrary SS3 models, but rather properties of assessment models in general. Thus **ss3sim** is not ideal for quickly exploring properties of a particular assessment model and other software packages should be explored if that is your goal (see accompanying for alternatives).

Here is a list of SS3 features that are not currently implemented in **ss3sim**:

- Seasons, sexes, hermaphrodism, multiple areas, movement, growth morphs, and platoons
- Data other than scientific surveys, commercial indices of abundance and age/length compositions: age-at-length, discards, mean weights, etc.

Some of the features may work within the **ss3sim** framework, but are untested.

It is possible to create new models that will work within the **ss3sim** framework, but this task will be complicated and likely require extensive knowledge of SS3 and R, as well as modification of the **ss3sim** functions. This process is described in more detail in Sections 15 and 16.

Instead of creating entirely new models, we recommend adapting the current built-in models to match the desired model setups for a new simulation study. Since these models have been thoroughly tested and used with **ss3sim** already (see Johnson et al. 2014; Ono et al. 2014), they make an ideal starting place. Before proceeding it would be wise to examine

the built-in models to determine how closely they match your desired model setups and whether simple changes can get you reasonably close for simulation purposes.

Say for example you want to modify the cod model to have different maturity, and then explore different sampling schemes using the `sample_index`, `sample_lcomp`, and `sample_agecomp` functions. The following steps provide a basic guideline for how to accomplish this:

- Using the original cod model create the case argument files you desire for your simulation and verify they run with the original cod model using the function `run_ss3sim`. It is probably best to do a shorter deterministic (Section 3.3) run. After running, read in the data and do visual checks for proper functionality.
- Find the original cod model. This can be found in the `inst/extdata/models` folder inside the package, located by

  ```
  library(ss3sim)
  d <- system.file("extdata", package = "ss3sim")
  paste0(d, "/models")
  ```

  Make a copy of the cod models (OM and EM) and rename them as desired.
- Make a single change to either the `.dat` or `.ctl` files for the new model and run them manually with SS3 if there is any question if they might break the model.
- Rerun the model through `run_ss3sim` and verify it is still working. If errors occur in the R function you will need to examine the function to determine why the error is occurring and fix by changing the R function and reloading it.
- Repeat previous steps with all small changes until the models are satisfactory.
- Turn off deterministic runs and run the simulation stochastically.
- If your new model works well with the package and is significantly different than what is built-in, please contact the **ss3sim** package managers for inclusion in future versions.

9 Adjusting the available OM data dynamically

ss3sim can dynamically adjust the type of data available from the OM for sampling. For example, you might wish to sample conditional length-at-age in a certain bin structure for certain years and fleets. To do that, the OM will need to generate length and age data for appropriate years and fleets at a sufficiently fine bin size. However, the idea behind **ss3sim** is to avoid having to hand code many different versions of an SS3 OM that have only minor differences. One way around this would be to write an OM that

included all possible data types for all possible years at an extremely fine bin structure. But, this can substantially slow down how quickly the OM runs in SS3. Therefore, we made **ss3sim** capable of dynamically modifying the OM to include just the data types and quantities that are needed based on arguments that are passed to the sampling functions (e.g. `sample_agecomp`). Internally, **ss3sim** uses the function `calculate_data_units` to generate a superset of all necessary fleet and year combinations. See section 10 below for more information on these types of data.

The dynamic creation of available data is the default behavior of the package. In practice this means the user does not need to manage which data types, fleets, or years are produced by the OM. However, in some cases the user may wish to disable this behavior and there is a top-level argument available in `ss3sim_base` called `call_change_data` (default of `TRUE`). For example, in some cases it may be more beneficial to call `change_data` before the simulation to manually prepare the OM data file, which will then be available for all subsequent EMs. Dynamic OM data creation would then be unnecessary.

In the default case of dynamic data creation, we apply the following rules if the arguments to any sampling function are not `NULL`:

- If `sample_agecomp`, generate age-composition data
- If `sample_lencomp`, generate length-composition data
- If `sample_mlacomp`, generate age-composition data and mean length-at-age data
- If `sample_calcomp`, generate length- and age-composition data and conditional length-at-age data
- If `sample_wtatage`, generate empirical weight-at-age, age-composition, and mean length-at-age data

For instance, if empirical weight-at-age data are to be sampled (`sample_wtatage` is called in a scenario), `change_data` adds age composition and mean length-at-age data to the model.

Another feature of `change_data` is that it can also modify the binning structure of the data (not the population-level bins). The ages and lengths in the data are specified in the arguments `len_bin` and `age_bin` in `ss3sim_base`. These bins need to be consistent across the entire data file for SS3 to operate.[3] Also, when either of these are set to `NULL` (the default value) then the respective bins are taken from the original OM (i.e. be left unchanged).

[3]Empirical weight-at-age data are a special case because they are generated in a separate file and use the population bins

10 Generating observation error

Simulated data arising from sampling the "truth" (i.e. observations), can be added within the **ss3sim** framework automatically and dynamically. Currently **ss3sim** supports five types of data: indices of abundance, length and age compositions, mean length (size) at age, empirical weight at age, and conditional age at length. Functions generate these data using the truth from the OM `.dat` files, and create the input `.dat` files for the EM. Since this sampling process is done dynamically with functions there is a lot of flexibility that can be added by the user.

This section briefly details the background and functionality of these "sampling" functions. Note that the years/fleets to be sampled must be present in the OM output data file. **ss3sim** does this data creation dynamically but in some cases it may need to be done manually (see section 9)

In simulation work, the true underlying dynamics of the population are known and therefore a variety of sampling techniques are possible. For instance ideal sampling (in the statistical sense) can be done easily. However, there is some question about the realism behind providing the model with this kind of data, since it is unlikely to happen in practice (Pennington et al. 2002). Thus, there is a need to be able to generate more realistic data. In **ss3sim** we can accomplish this in several ways. One way is to use flexible distributions which allow the user to control the statistical properties (e.g. overdispersion) of the data.

For example, consider simulating age composition data. Under perfect mixing of fish and truly random sampling of the population, the samples would be multinomial. However, in practice neither of these are true because the fish tend to aggregate by size and age, and it is difficult to take random samples (e.g. Pennington et al. 2002). This causes the data to have more variance than expected, i.e. be overdispersed, and the effective sample size is smaller (Hulson et al. 2012; Maunder 2011). For example, if a multinomial likelihood is assumed for overdispersed data, the model puts too much weight on those data, at the cost of fitting other data less well (Maunder 2011). Analysts thus often "tune" their model to find a more appropriate sample size (called "effective sample size") that (hopefully) more accurately reflects the information in the composition data. In our case, we exactly control how the data are generated and specified in the EM – providing a large amount of flexibility to the user. What is optimal will depend on the questions addressed by a simulation and how the results are meant to be interpreted. We caution users to carefully consider how data are generated and fit in an **ss3sim** simulation.

Below we summarize the types of data currently incorporated into the package, how sampling is done, and how effective sample sizes are dealt with.

10.1 Indices of abundance

The function [`sample_index`] facilitates generating relative indices of abundance (CPUE for fishery fleets). It samples from the biomass trends for the different fleets to simulate the collection of CPUE and survey index data. The user can specify which fleets are sampled in which years and with what amount of noise. Different fleets will "see" different biomass trends due to differences in selectivity, so each index is associated with the same fleet between the OM and EM. Since $q = 1$ for the OM, the indices are actually absolute indices of (spawning) biomass.

In practice, sampling from the abundance indices is relatively straightforward. The OM `.dat` file contains the annual biomass values for each fleet, and the `sample_index` function uses these true values as the mean of a distribution. The function uses a bias-corrected log-normal distribution with expected values given by the OM biomass and a user-provided standard deviation term that controls the level of variance.

More specifically, let

$$\begin{aligned} B_y &= \text{the true (OM) biomass in year } y \\ \sigma_y &= \text{the standard deviation provided by the user} \\ X \sim N(0, \sigma_y^2) &= \text{a normal random variable} \end{aligned}$$

Then the sampled value, B_y^{obs} is

$$B_y^{\text{obs}} = B_y e^{X - \sigma_y^2/2}$$

which has expected value $\mathrm{E}\left[B_y^{\text{obs}}\right] = B_y$ due to the bias adjustment term $\sigma_y^2/2$ (`SR_sigmaR` in SS3). This process generates log-normal values centered at the true value. It is possible for the user to specify the amount of uncertainty (e.g. to mimic the amount of survey effort), but it is not possible to induce bias in this process (as currently implemented).

10.1.1 Effective sample size

For index data, which are assumed log-normal by SS3, the weight of the data is determined by the CV of each point. As the CV increases the data have less weight in the joint likelihood. This is equivalent to decreasing the effective sample size in other distributions. **ss3sim** sets these values automatically at the truth internally, although future versions may allow for more complex options. The user-supplied σ_y term is written to the `.dat` file along with the sampled values, so that the EM has the correct level of uncertainty. Thus, the EM has unbiased estimates of both B_y and the true σ_y for all fleets.

10.2 Age and length compositions

The functions `sample_agecomp` and [`sample_lcomp`] sample from the true age and length compositions using a multinomial or Dirichlet distribution. The user can further specify which fleets are sampled in which years and with what amount of noise (via sample size).

The following calculations demonstrate how **ss3sim** handles compositional data. They are shown for age compositions but apply equally to length compositions. The multinomial distribution $\mathbf{m} \sim \text{MN}(\mathbf{p}, n, A)$ is defined as

$$\begin{aligned}
\mathbf{m} &= m_1, \ldots, m_A \\
&= \text{the number of fish observed in bin } a \\
\mathbf{p} &= p_1, \ldots, p_A \\
&= \text{the true proportion of fish in bin } a \\
n &= \text{sample size} \\
A &= \text{number of age bins}
\end{aligned}$$

In the case of SS3, actual proportions are input as data so $\mathbf{m}/n$ is the distribution used instead of $\mathbf{m}$. Thus, the variance of the estimated proportion for age bin a is $\text{Var}[M_a/n] = p_a q_a/n$. Note that m_a/n can only take on values in the set $\{0, 1/n, 2/n, \ldots, 1\}$. With sufficiently large sample size this set approximates the real interval $[0, 1]$, but the key point being that only a finite set of rational values of m_a/n are possible.

The multinomial, as described above, is based on assumptions of ideal sampling and is likely unrealistic. One strategy to add realism is to allow for overdispersion.

10.2.1 Sampling with overdispersion

The composition sampling functions provided in the package allow the user to specify the level of overdispersion through the argument `cpar`. However, the package does not currently allow for specifying the effective sample size. See the function documentation for `sample_agecomp` and `sample_lcomp` for more detail.

Here we describe the implementation of the Dirichlet distribution utilized in the package. This distribution has the same range and mean, but a different variance controlled by a parameter (the variance is determined exclusively by the cell probabilities in the multinomial) and as such is more flexible. Let $\mathbf{d} \sim \text{Dirichlet}(\boldsymbol{\alpha}, A)$ be a Dirichlet random vector.

It is characterized by:

$$\begin{aligned}
\mathbf{d} &= d_1, \ldots, d_A \\
&= \text{the proportion of fish observed in bin } a \\
\boldsymbol{\alpha} &= \alpha_1, \ldots, \alpha_A \\
&= \text{concentration parameters for the proportion of fish in bin } a \\
A &= \text{number of age bins}
\end{aligned}$$

Since we are using the Dirichlet distribution to generate random samples, it is convenient to parameterize the vector of concentration parameters $\boldsymbol{\alpha}$ as $\lambda \mathbf{p}$ so that $\alpha_a = \lambda p_a$. The mean of d_a is then $\mathrm{E}[d_a] = p_a$ and the variance is $\mathrm{Var}[d_a] = \frac{p_a q_a}{\lambda+1}$. The marginal distributions for the Dirichlet are beta distributed. In contrast to the multinomial above, the Dirichlet generates points on the real interval $[0, 1]$.

The following steps are used to generate overdispersed samples:

1. Get the true proportions at age (from the operating model "truth") for the number of age bins A.
2. Determine a realistic sample size, say $n = 100$. Calculate the variance of the samples from a multinomial distribution, call it V_{m_a}.
3. Specify a level, c, that scales the standard deviation of the multinomial. Then $\sqrt{V_{d_a}} = c\sqrt{V_{m_a}}$ from which $\lambda = n/c^2 - 1$ can be solved. Samples from the Dirichlet with this value of λ will then give the appropriate level of variance. For instance, we can generate samples with twice the standard deviation of the multinomial by setting $c = 2$.

10.2.2 Effective sample sizes

For both multinomial and Dirichlet generated composition data the effective sample size is set automatically by the `sample_lcomp` and `sample_agecomp` functions. In the case of the multinomial the effective sample size is just the total sample size (i.e. how many fish were sampled, the number of sampled tows, etc.). However, with the Dirichlet distribution the effective sample size depends on the parameters passed to these functions and is automatically calculated internally and passed on to the `.dat` file. The effective sample size is calculated as $N_{eff} = n/c^2$. Note that these values will not necessarily be integer-valued, and SS3 handles this without issue.

Given that the effective sample size is known and passed to the EM, there is much similarity between the multinomial and Dirichlet methods for generating data. The main difference arises when sample sizes (n) are small; in this case the multinomial will be restricted to few

potential values (i.e. $0/n, 1/n, \ldots, n/n$), whereas the Dirichlet has values in $[0, 1]$. Thus without the Dirichlet it would be impossible to generate realistic values that would come about from highly overdispersed data with a large n.

10.3 Mean length at age

TODO

10.4 Conditional age at length

TODO

10.5 Empricial weight at age

TODO

11 Generating process error

Process error is incorporated into the OM in the form of deviates in recruitment ('recdevs') from the stock-recruit relationship. Unlike the observation error, the process error affects the population dynamics and thus must be done before running the OM.

These built-in recruitment deviations are standard normal deviates and are multiplied by σ_r (recruitment standard deviation) as specified in the OM, and bias adjusted within the package. That is,

$$\text{recdev}_i = \sigma_r z_i - \sigma_r^2/2$$

where z_i is a standard normal deviate and the bias adjustment term ($\sigma_r^2/2$) makes the deviates have an expected value of 1 after exponentiation.

If the recruitment deviations are not specified, then the package will use these built-in recruitment deviations. Alternatively you can specify your own recruitment deviations, via the argument `user_recdevs` to the top-level function `ss3sim_base`. Ensure that you pass a matrix with at least enough columns (iterations) and rows (years). The user-supplied recruitment deviations are used exactly as specified (i.e. not multiplied by σ_r as specified in the SS3 model), and **it is up to you to bias correct them manually** by subtracting $\sigma_r^2/2$ as is done above. This functionality allows for flexibility in how the recruitment deviations are specified, for example running deterministic runs (Section 3.3) or adding serial correlation.

Note that for both built-in and user-specified recdevs **ss3sim** will reuse the same set of recruitment deviations for all iterations across scenarios. For example if you have two scenarios and run 100 iterations of each, the same set of recruitment deviations are used between those two scenarios.

12 Stochastic reproducibility

In many cases, you may want to make the observation and process error reproducible. For instance, you may want to reuse process error so that differences between scenarios are not confounded with process error. More broadly, you may want to make a simulation reproducible on another machine by another user (such as a reviewer).

By default **ss3sim** sets a seed based on the iteration number. This will create the same recruitment deviations for a given iteration number. You can therefore avoid having the same recruitment deviations for a given iteration number by either specifying your own recruitment deviation matrix through the `user_recdevs` argument or by changing the iteration numbers (e.g. using iterations 101 to 200 instead of 1 to 100).

If you want the different scenarios to have different process error you will need to make separate calls to `run_ss3sim` for each scenario and pass a different matrix of `user_recdevs` (see Section 11) or pass different iteration numbers. For most applications this should not be necessary.

The observation error seed (affecting the index, length composition, and age composition sampling) is set during the OM generation. Therefore, a given iteration-case-file-argument combination will generate repeatable results. Given that different case arguments can generate different sampling routines (e.g. stochastically sampling or not sampling from the age compositions or sampling a different number of years) the observation error is not necessarily comparable across different case arguments.

13 Parallel computing with ss3sim

ss3sim can easily run multiple scenarios in parallel to speed up simulations. To run a simulation in parallel, you need to register multiple cores or clusters and set `parallel = TRUE` in `run_ss3sim`. For example, we could have run the previous example in parallel with the following code. First, we register four cores:

```
require(doParallel)
registerDoParallel(cores = 4)
```

We can check to make sure we are set up to run in parallel:

```
require(foreach)
getDoParWorkers()

## [1] 4
```

And then run our simulation on four cores simultaneously by setting `parallel = TRUE`:

```
run_ss3sim(iterations = 1, scenarios =
  c("D1-E0-F0-R0-M0-cod",
    "D2-E0-F0-R0-M0-cod",
    "D1-E1-F0-R0-M0-cod",
    "D2-E1-F0-R0-M0-cod"),
  case_folder = case_folder, om_dir = om, em_dir = em,
  bias_adjust = TRUE, parallel = TRUE)
```

In addition to the check with `getDoParWorkers()` above, if the simulations are running in parallel, you will also see simultaneous output messages from **ss3sim** as the simulations run. On a 2.27 GHz Intel Xeon quad-core server running Ubuntu 10.04, this example ran 1.9 times faster on two cores than on a single core, and 3.2 times faster on four cores.

Alternatively, you can run iterations in parallel. This is useful, for example, if you want to run a single scenario as quickly as possible. To do this, just set `parallel_iterations = TRUE` as well as setting `parallel = TRUE`. For example:

```
run_ss3sim(iterations = 1:2, scenarios = "D0-E0-F0-R0-M0-cod",
  case_folder = case_folder, om_dir = om, em_dir = em,
  parallel = TRUE, parallel_iterations = TRUE)
```

If you are running bias adjustment iterations, these will be run first in serial. The rest of the iterations will be run in parallel:

```
run_ss3sim(iterations = 1:2, scenarios = "D0-E0-F0-R0-M0-cod",
  case_folder = case_folder, om_dir = om, em_dir = em,
  parallel = TRUE, parallel_iterations = TRUE, bias_nsim = 2,
  bias_adjust = TRUE)
```

14 Putting SS3 in your path

Instead of copying the SS3 binary file (`ss3.exe` or `ss3`) to each folder within a simulation and running it, **ss3sim** relies on a single binary file being available to the operating system

regardless of the folder. To accomplish this, SS3 must be in your system path, which is a list of folders that your operating system looks in whenever you type the name of a program on the command line. This approach saves on storage space since the SS3 binary is about 2.2 MB and having it located in each folder can be prohibitive in a large-scale simulation testing study.

14.1 For Unix (OS X and Linux)

To check if SS3 is in your path: open a Terminal window and type `which SS3` and hit enter. If you get nothing returned, then SS is not in your path. The easiest way to fix this is to move the SS3 binary to a folder that's already in your path. To find existing path folders type `echo $PATH` in the terminal and hit enter. Now move the SS3 binary to one of these folders. For example, in a Terminal window type:

```
sudo cp ~/Downloads/SS3 /usr/bin/
```

You will need to use `sudo` and enter your password after to have permission to move a file to a folder like `/usr/bin/`.

Also note that you may need to add executable permissions to the SS3 binary after downloading it. You can do that by switching to the folder where you placed the binary (`cd /usr/bin/` if you followed the instructions above), and running the command:

```
sudo chmod +x SS3
```

Check that SS3 is now executable and in your path:

```
which SS3
```

If you followed the instructions above, you will see the following line returned:

```
/usr/bin/SS3
```

If you have previously modified your path to add a non-standard location for the SS3 binary, you may need to also tell R about the new path. The path that R sees may not include additional paths that you have added through a configuration file like `.bash_profile`. If needed, you can add to the path that R sees by including a line like this in your `.Rprofile` file. (This is an invisible file in your home directory.)

```
Sys.setenv(PATH=paste(Sys.getenv("PATH"),"/my/folder",sep=":"))
```

14.2 For Windows

To check if SS is in your path for Windows 7 and 8: open a DOS prompt and type `ss3 -?` and hit enter. If you get a line like "`ss3 is not recognized...`" then SS3 is not in your path. To add the SS3 binary file to your path, follow these steps:

1. Find the correct version of the `ss3.exe` binary on your computer
2. Record the folder location. E.g. `C:/SS3.24o/`
3. Click on the start menu and type `environment`
4. Choose `Edit environment variables for your account` under Control Panel
5. Click on `PATH` if it exists, create it if does not exist
6. Choose `PATH` and click edit
7. In the `Edit User Variable` window add to the **end** of the `Variable value` section a semicolon and the SS3 folder location you recorded earlier. E.g. `;C:/SS3.24o/`. **Do not overwrite what was previously in the PATH variable**.
8. Restart your computer
9. Go back to the DOS prompt and try typing `ss3 -?` and hitting return again.

15 Setting up a new operating model

In some cases the user may wish to adapt their own SS3 model to work with the **ss3sim** package. This is possible but may be difficult because the functions in **ss3sim** were developed to work with these three model setups and a model with a different structure may cause errors in these functions. This stems from the high flexibility of SS3, allowing for more complex model setups than those used while developing **ss3sim**. For instance, the `sample_index` function does not have the capability to handle more than one season (it could, but currently it is not developed to). Given the many options available in SS3 it is extremely difficult to write auxiliary functions that will interact reliably with all combinations of these options. For this reason, we recommend that users strongly consider trying to modify an existing model rather than creating a new one (Section 8). It is likely that the **ss3sim** functions will need to be modified in some way in the process, so the user should be familiar with both SS3 and R.

The main purpose of the OM is to generate data files that can be read into the EM. Thus the user needs to setup the `.dat` files in the OM such that they conform to the structure needed by the EM. Two key examples are with survey and age/length composition data. For the indices of abundance (CPUE and scientific survey) the OM `.dat` file will determine which years are available to the sampling function `sample_index`, so if the year y is desired in the EM it needs to be in the OM. In practice it may be easiest to just include all years in the OM if surveys will be dynamic (i.e. changing years between scenarios), or if it will be fixed for all scenarios to set it to match the EM exactly.

Similarly with age/length compositions the OM `.dat` file will determine which years and bins are available to the sampling functions `sample_agecomp` and `sample_lcomp`. If dynamic binning is to be used, the user should setup the `.dat` file so that all desired combinations of bins are possible (see Section **??** for more details). Specifically, the user must specify small enough OM `.ctl` bins (no smaller than the population bin specified in the appropriate section in the OM `.dat` file) so that they can easily be re-binned. Alternatively, if composition data is not to be explored in the simulation then the user can just set the OM `.dat` file to match the desired input for the EM `.dat`.

For those users who choose to create a new **ss3sim** model setup, we outline the steps to take an existing SS3 model and modify it to work with the **ss3sim** package. First, we cover setting up an operating model and then in Section 16 we cover setting up an estimation model.

The first step is to run the assessment model to make sure the model runs and estimates parameters as desired. The .par file generated will be used in the OM. We recommend opening a command window inside your folder to help test whether the model still runs at many points along the process. After turning parameter estimation off in the starter file (see below), the model can be checked by running `ss3_24o_safe -nohess` to make sure the input files are read in properly and the model writes the files. Use the `.ss_new` files

produced as a starting point so that the comments match the instructions below.

You will see an error by ADMB that it cannot find the dat file because we have specified a different file in the starter file. You can safely ignore this error.

15.1 Starter file modifications

1. Delete the starter.ss file and rename `starter.ss_new` to `starter.ss`. Modify this new file.

2. Turn off parameter estimation by changing
   `# Turn off estimation for parameters entering after this phase`
   to `0`.

3. Use the `.par` file to initialize model parameters. To do so change
   `# 0=use init values in Starter file` to `1`. Parameter values specified in the `.ctl` file will now be ignored.

4. Generate detailed report files (containing age-structure information) by setting
   `# detailed age-structured reports in REPORT.SSO` to `1`.

5. Generate data by setting
   `# Number of datafiles to produce` to `3`.
   If `X=1` it only generates the original data If `X=2` it generates the original data and the expected value data (based on model specification) If `X>=3` it generates all the above and `X-2` bootstrapped data sets

6. Turn off parameter jittering by setting
   `# jitter initial parm value by this fraction` to `0`. Jitter is used, among other things, to test for convergence to the same solution when starting from alternative initial values; however, the OM is used here as the truth, so jittering is not needed.

7. Turn off retrospective analyses by setting
   `# retrospective year relative to end year`
   to `0`. To analyze the data for retrospective patterns, use the R case file.

8. Specify how catch is reported by setting `# F_report_units` to 1 if catch is reported in biomass or `2` if catch is reported in numbers. Additionally, comment out the next line, `#_min and max age over which average F will be calculated`, by removing all characters prior to the hash symbol.

9. Implement catches using instantaneous fishing mortality by changing
   `# F_report_basis` to `0`.

15.2 Control file modifications

1. Specify all environmental deviates to be unconstrained by bounds by setting `#_env/block/dev_adjust_method` to `1`. If the method is set to `2`, parameters adjusted using environmental covariate inputs will be adjusted using a logistic transformation to ensure that the adjusted parameter will stay within the bounds of the base parameter.

2. Turn on recruitment deviations by specifying `#do_recdev` to `1`. Using the next two lines, specify the use of recruitment deviations to begin and end with the start and end years of the model.

3. Turn on additional advanced options for the recruitment deviations by specifying `# (0/1) to read 13 advanced options` to `1`.

4. Set `#_recdev_early_start` to `0` so that the model will use the `# first year of main recr_devs`.

5. Set `#_lambda for Fcast_rec_like occurring before endyr+1` to `1`. This lambda is for the log likelihood of the forecast recruitment deviations that occur before the first year of forecasting. Values larger than one accommodate noisy data at the end of the time series.

6. Recruitment is log-normally distributed in SS. If inputting a normally distributed recruitment deviations specify `#_max_bias_adj_in_MPD` to `-1` so that SS performs the bias correction for you. If inputting bias corrected normal recruitment deviation, specify it at `0`. Either method will lead to the same end result.

7. Use any negative value in line `# F ballpark year`, to disable the use of a ballpark year to determine fishing mortality levels.

8. Specify `# F_Method` to `2`, which facilitates the use of a vector of instantaneous fishing mortality levels. The max harvest rate in the subsequent line will depend upon the fishing mortality levels in your simulation. Following the max harvest rate, specify a line with three value separated by spaces. The first value is the overall start F value, followed by the phase. The last value is the number of inputs. Set the number of inputs to `1`, because the actual fishing mortality trajectory will be specified in the `.dat` file. Next, specify a single line with six values, separated by spaces, where the values correspond to fleet number, start year, season, fishing mortality level, the standard error of the fishing mortality level, and a negative phase value. E.g `1 2000 1 0 0.01 -1`

9. If needed, change the specification of the `.ctl` using the functions available in the **ss3sim** package. E.g `change_growth`, `change_sel`.

15.3 Data file modifications

1. Specify the start and end year for the simulation by modifying `#_styr` and `#_endyr`. Years can be specified as a number line (i.e. `1`, `2`, `3`, ...) or as actual years (i.e. `1999`, `2000`, `2001`, ...).
2. Specify the names for each fleet in an uncommented line after the line `#_N_areas`. Names must be separated by a `%` with no spaces. It is these names which you will use in the plain text case files to specify and change characteristics of each fleet throughout the simulation. E.g. `Fishery%Survey1%Survey2`
3. Specify the number of mean body weight observations across all selected sizes and ages to be specific to measured fish by setting `#N observations` to `0`. Subsequently, specify `1` degree of freedom for the Student's T distribution used to evaluated the mean body weight deviations in the following line. The degrees of freedom must be specified even if there are zero mean body weight observations.
4. Set the length bin method to 1 or 2 in the line labelled `# length bin method`. Using a value of `1`, the bins refer to the data bins (specified later). Using a value of `2` instructs SS to generate the binwidths from a user specified minimum and maximum value. In the following three lines, specify the binwidth for population size composition data; the minimum size, or the lower edge of the first bin and size at age zero; and the maximum size, or lower edge of the last bin. The length data bins MUST be wider than the population bin, but the boundaries do not have to align.
5. Specify `#_comp_tail_compression` to any negative value to turn off tail compression.
6. Specify `#_add_to_comp` to a very small number E.g `1e-005`. This specifies the value that will be added to each composition (age and length) data bins.
7. Set the length bin range method for the age composition data (used when the conditional age at length data exists) to 1, 2 or 3 in the line `#_Lbin_method` depending on the data you have or the purpose of the study.

15.4 Testing the new operating model

After completing the above steps, check that the SS3 model setup is functional by running a single iteration of the model and verifying that the data are read in correctly and expected values of the population dynamics are written to the `.dat` files (and sensical). We also advise manually testing the **ss3sim** R functions that manipulate the OM (e.g., `change_tv`) and check that the model setup still runs correctly after this manipulation. The help files for the functions demonstrate how to use the functions to test models. Note that the OM will not be a valid SS3 model in the sense that ADMB cannot run and produce maximum likelihood estimates of parameters; it is intended to only be run for one iteration to generate the population dynamics using values specified in the input files.

16 Setting up a new estimation model

Unlike the OM, the EM needs to be a valid SS3 model setup and run to achieve MLE estimates (and possibly standard errors). Thus the OM needs to be adapted to create a new EM.

16.1 Starter file modifications

1. Change the names of the `.dat` and `.ctl` files to your chosen naming scheme.
2. Specify the model to use parameter values found in the `.ctl` file, by changing `# 0=use init values in control file; 1=use ss3.par` to `0`.
3. Turn on parameter estimation by changing `# Turn off estimation for parameters entering after this phase` to a value larger than the max phase specified in the `.ctl` file.

16.2 Control file modifications

1. Set the phases of the parameters to positive or negative value to inform SS to estimate or fix the parameters, respectively.
2. Set the `#_recdev phase` to a positive value to estimate yearly recruitment deviations.
3. If using bias adjustment set `#_recdev_early_phase` to a positive value. Estimates for the years and maximum bias adjustment can initially be inputted with approximations or use the bias adjustment function within **ss3sim** to find appropriate values for the base case EM and input them in the appropriate lines.
4. Specify `# F_Method` to `3`, which allows the model to use catches to estimate appropriate fishing mortality levels. The max harvest rate in the subsequent line will depend upon the fishing mortality levels in your simulation. An additional line must be inserted after the maximum harvest rate to specify the number of iterations used in the hybrid method from `3` to `7`.
5. Use the functions in the **ss3sim** package to change the estimation specification in the EM. E.g. `change_e`

16.3 Data file modifications

The `data.ss_new` files produced when executing the OM contain the expected values of the OM population dynamics. The data the EM model is fit to needs to be sampled with observation error from these expected values in order to mimic the random sampling

process done with real fisheries data. The **ss3sim** package provides three functions which carry out the random sampling process and generate `.dat` files to be used in the EM. See Section 10 for more details.

16.4 Testing the new estimation model

After completing the above steps run the model manually and verify that it loads the data properly and the objective function value (negative log-likelihood) is sensible. If it works correctly, try running deterministic cases on the model (Section 3.3) and further verify that **ss3sim** functions that modify the EM (e.g., `change_e`) act correctly on the model setup. It is possible that some of the functions will not work perfectly with the new model setups. In this case, it may be necessary to modify the **ss3sim** functions to be compatible with the new OM and EM.

17 Adding a new SS3 manipulation function

You can create your own new function to manipulate SS3 configuration files during an **ss3sim** simulation. All this function needs to do is read in the appropriate SS3 files, manipulate them in some way, and write them back out. You then need to insert your new function into the main **ss3sim** simulation routine so it gets called at the appropriate time. The following instructions briefly outline all the steps to this process. Feel free to contact any of the **ss3sim** developers for help. If you create a new manipulation function it would be great to get it incorporated into the official package.

The new manipulation function can be used by one or more cases. Cases can be mandatory (must be included in all scenarios, like `F` and `D`) or optional like `S` where if not specified then nothing occurs.

1. Write your change function in the `/R` folder.
    - Aim for singular names (e.g. `change_bin`).
    - Input and output file arguments should be written as `file_in` and `file_out` or if multiple types are needed then add a prefix. E.g. `dat_file_in`.
    - In general try to match argument names as best as possible. A good approach is to start with a similar function and modify as needed.
    - Unless otherwise necessary, arguments should take SS file names (as character values) and do the reading internally using r4ss functions (as opposed to taking lists that have already been read in by r4ss). Currently, there are a few places in ss3sim (e.g. the sampling functions) that take an already-read version as an argument value. This will be standardized eventually.

- Use **roxygen2** `@param`, `@author`, etc. to document your function. Then run `devtools::document()` and `devtools::check(cran = TRUE)` on your new code. Test that the documentation is right with `devtools::dev_help("your_change_function")`.
- Also make sure you have an example and can run `devtools::dev_example("your_change_function")`. Note that your example *should not* run SS3 itself. See other functions for how to create a working example from packaged files.

2. Now go to `/R/case-parsing.r`.

   - You need to pick a capital letter to represent your case. This should be different than any existing case letter. E.g., at the time of writing this, the letters D, E, F, G, M, R, and S are being used.
   - Add your new letter to the `get_caseargs` default argument values if you want it to be a mandatory case. Otherwise your new case letter must be passed explicitly in the `run_ss3sim` call.

3. Go to `/R/ss3sim_base.r`.

   - Add a new argument to the function `ss3sim_base` that corresponds to your new case. For example, `f_params`. Make sure to also document the new argument with roxygen2 `@param` syntax.
   - Now that your case arguments are passed to this function, you can use them in the function call. So add a call to your function within the code of `ss3sim_base`. Where in the code depends on when your function needs to be run in the flow of a simulation. The `change_f` needs to occur before the OM runs, and the sampling of data occurs after the OM runs.
   - Run `devtools::document()` to update the argument list for `ss3sim_base` and check it.

4. Go to `/R/run_ss3sim.r`.

   - `run_ss3sim` calls `ss3sim_base` so `run_ss3sim` needs to be updated to include your new case argument from step 3.
   - The variable `a` is a list of list of arguments, one element of which is your new set of arguments. For instance `a$F` will contain the vector of `F` values.
   - Update the call to `ss3sim_base` to include your new argument. There are two places to do this. (1) The parallel section and (2) the sequential section. For instance `f_params=a$F`.
   - If the case is optional and not passed, the element of the list will be NULL. That is, `a$X` will be NULL if the case `X` isn't used. You can test for this in `ss3sim_base` and skip your function, effectively turning it off if there is no case argument available. This will also prevent previous code from breaking.

5. Test your new function and make sure that all existing examples continue to work.

    - First test that `get_caseargs` reads in and parses your case files properly. If your case isn't mandatory you need to explicitly pass the cases to the `case_files` argument. E.g. if your case is `X`then `case_files=list(M="M", F="F", D=c("index", "lcomp", "agecomp"), R="R", E="E", X="X")`. This function will return a list of lists of the case arguments read from file.
    - You will have to do more work if your case is mandatory, since it will break any previous scenarios. You will have to: add new case files for the example models, the example code in the functions, and the vignette code.
    - Remember to run `devtools::document()` after editing the examples.
    - In general, the user should be able to easily bypass your function and have it do nothing.
    - You can check all the examples with `devtools::run_examples()`.
    - Of course, do another `devtools::check(cran = TRUE)`. This will also test the vignette building.

References

Anderson, S. C., Monnahan, C. C., Johnson, K. F., Ono, K., and Valero, J. L. (2014). ss3sim: An R package for fisheries stock assessment simulation with Stock Synthesis. *PLOS ONE*, 9(4):e92725.

Hill, K. T., Crone, P. R., Lo, N. C. H., Demer, D. A., Zwolinski, J. P., and Macewicz, B. J. (2012). Assessment of the Pacific sardine resource in 2012 for U.S. management in 2013. Technical report, Pacific Fishery Management Council, 7700 NE Ambassador Place, Portland, OR 97220, USA.

Hulson, P.-J. F., Hanselman, D. H., and Quinn, T. J. (2012). Determining effective sample size in integrated age-structured assessment models. *ICES Journal of Marine Science: Journal du Conseil*, 69(2):281–292.

Johnson, K. F., Monnahan, C. C., McGilliard, C. R., Vert-pre, K. A., Anderson, S. C., Cunningham, C. J., Hurtado-Ferro, F., Licandeo, R., Muradian, M. L., Ono, K., Szuwalski, C. S., Valero, J. L., Whitten, A. R., and Punt, A. E. (2014). Time-varying natural mortality in fisheries stock assessment models: identifying a default approach. *ICES J Mar Sci*, In press. DOI: 10.1093/icesjms/fsu055.

Maunder, M. N. (2011). Review and evaluation of likelihood functions for composition data in stock-assessment models: Estimating the effective sample size. *Fisheries Research*, 109(2):311–319.

Methot, R. D. and Taylor, I. G. (2011). Adjusting for bias due to variability of estimated recruitments in fishery assessment models. *Can. J. Fish. Aquat. Sci.*, 68(10):1744–1760.

Ono, K., Licandeo, R., Muradian, M. L., Cunningham, C. J., Anderson, S. C., Hurtado-Ferro, F., Johnson, K. F., McGilliard, C. R., Monnahan, C. C., Szuwalski, C. S., Valero, J. L., Vert-pre, K. A., Whitten, A. R., and Punt, A. E. (2014). The importance of length and age composition data in statistical catch-at-age models for marine species. *ICES J Mar Sci*, In press. DOI: 10.1093/icesjms/fsu007.

Pennington, M., Burmeister, L.-M., Hjellvik, V., et al. (2002). Assessing the precision of frequency distributions estimated from trawl-survey samples. *Fishery Bulletin*, 100(1).