SEBDAM Data Example

Raphaël McDonald

Contents

1 TLM Data Example

This document aims to demonstrate how to fit TLM to any data, using survey data from the Scallop Production Area 3, in the Canadian Maritimes Inshore Scallop Fishery, as an example alongside simulated landings as the real one are protected by privacy legislation.

Preliminary work (not shown here) done by the user should be relatively straightforward, simply consisting in making sure their data is organised in a single data frame with specific column names. The only modelling decision that is required at this stage will depend on the species of interest in that most species do not have observations that can be linked to the natural mortality of this species. Scallops, as per the example, have clappers, which are dead animals whose shells are still hinged together and which are used as observations for natural mortality. Therefore, if there are, the data will need to include these, while if they do not exist the model will not require them and will instead fix the instantaneous natural mortality to a user-defined choice.

If the user's data contains observations of natural mortality, it should look similar to the following example:

```
#Loading raw data
load("./Data/form_spa3_tlm.RData")
str(form_TLM_data)
```

```
## 'data.frame':    3035 obs. of  5 variables:
##  $ I   : num  84.9 1819.7 79.4 1503.2 352.6 ...
##  $ IR  : num  0 412.6 0 22.6 0 ...
##  $ L   : num  4 0 0 0 0 4 0 0 0 0 ...
##  $ N   : num  8 397 12 250 29 8 18 42 32 74 ...
##  $ Year: num  1 1 1 1 1 1 1 1 1 1 ...
```

As seen above, each row contains 1 survey tow with its observation of commercial size biomass (column I), the observation of recruit size biomass (column IR), the observed number of clappers (column L), the observed number of shells (live scallops + clappers, column N), the year in which this tow was taken (column

Year). If the user was not using observations of natural mortality, the data would look very similar except removing the columns L and N.

One can proceed with TLM simply with this formatted data frame.

1.1 Step 1: Setup data for TMB.

The first step consists in reformatting the data in a way that TMB can understand using the data_setup() function. This is also the step where the few required modelling decisions have to be made. As this function is also used for SEBDAM, it has a lot of possible parameters that are ignored when using TLM, so we will only look at the ones required by TLM here.

- data: formatted dataframe as shown earlier in this document.
- growths: dataframe containing the growth rates g and gR, which should look like the following:

```
load("./Data/spa3_growths.RData")
colnames(spa3_growths)<-c("g","gR")
str(spa3_growths)
```

```
## 'data.frame':    22 obs. of  2 variables:
##  $ g : num  1.2 1.41 1.09 1.32 1.08 ...
##  $ gR: num  1.64 1.58 1.36 1.72 1.49 ...
```

- catch: vector of landings in each year like the following:

```
load("./Data/sim_catches_TLM.RData")
str(sim_catches_TLM)
```

```
##  num [1:23] 15.4 35.1 36.9 37.8 41.5 ...
```

- model: choice of model, has to be "TLM" for our purpose here.
- obs_mort: whether to use observations of mortality to predict natural mortality, if there are obs_mort=TRUE, if not obs_mort=FALSE.
- prior: whether to use a prior beta distribution to estimate the commercial catchability, if yes prior=TRUE, if not prior=FALSE.
- prior_pars: choice of shape parameters used for the prior beta distribution (defaults are 10 and 12).

```
set_data<-data_setup(data=form_TLM_data,growths = spa3_growths,catch = sim_catches_TLM,model="TLM",obs_m
```

This returns the raw lists required by TMB to fit the model.

- data: all necessary data formatted in the appropriate way for TLM.

```
str(set_data$data)
```

```
## List of 14
##  $ model      : chr "TLM"
##  $ options_vec: num [1:2] 1 1
##  $ prior_pars : num [1:2] 10 12
##  $ logI       : num [1:3035] 4.44 7.51 4.37 7.32 5.87 ...
##  $ logIR      : num [1:3035] NA 6.02 NA 3.12 NA ...
##  $ C          : num [1:23, 1] 15.4 35.1 36.9 37.8 41.5 ...
##  $ g          : num [1:22] 1.2 1.41 1.09 1.32 1.08 ...
##  $ gR         : num [1:22] 1.64 1.58 1.36 1.72 1.49 ...
##  $ n_tows     : int [1:22] 119 100 152 160 148 111 123 109 162 147 ...
##  $ pos_tows_I : int [1:22] 117 95 135 141 133 109 112 106 142 132 ...
##  $ pos_tows_IR: int [1:22] 35 15 55 65 54 42 24 14 45 44 ...
##  $ t_i        : num [1:3035] 0 0 0 0 0 0 0 0 0 0 ...
##  $ L          : num [1:3035] 4 0 0 0 0 4 0 0 0 0 ...
##  $ n_bin      : num [1:3035] 12 397 12 250 29 12 18 42 32 74 ...
```

- par: starting values for all of TLM's parameters.

```
str(set_data$par)
```

```
## List of 13
##  $ log_sigma_tau    : num -1
##  $ log_sigma_phi    : num -1
##  $ log_sigma_epsilon: num -1
##  $ log_sigma_upsilon: num -1
##  $ log_q_R          : num -1
##  $ log_q_I          : num -1
##  $ logit_p_I        : num 0
##  $ logit_p_IR       : num 0
##  $ log_B            : num [1:23] 12.4 12.4 12.4 12.4 12.4 ...
##  $ log_R            : num [1:23] 11.2 11.2 11.2 11.2 11.2 ...
##  $ log_sigma_m      : num -1
##  $ log_S            : num -1
##  $ log_input_m      : num [1:23] -1.2 -1.2 -1.2 -1.2 -1.2 ...
```

- random: vector of character strings indicating which parameters are random effects.

```
str(set_data$random)
```

```
##  chr [1:3] "log_B" "log_R" "log_input_m"
```

- map: which parameters to fix or ignore, depending on modelling choices. In this example, as we are using data to get to the natural mortality, it should be empty:

```
str(set_data$map)
```

```
##  list()
```

1.2 Step 2: Fit the model!

Now that everything is formatted appropriately, the model can be directly fit using the fit_model() function. This function requires few decisions and parameters, atleast compared to the previous ones where the modelling decisions were made. It requires the following:

- tmb_obj: list obtained from data_setup() in step 1.
- optim: which optimizer to use, with 3 options: "nlminb" for backward compatibility if required (as it it not recommended to use anymore), default "optimr" from the package optimx which offers various optimizers, and "parallel" for non-Windows users if desired.
- control: for specifications using "nlminb", ignored otherwise
- optim_method: for choice of optimization method when using "optimr", default is "nlminb" (see package optimx for more options).
- cores: for use with "parallel"
- bias.correct: for bias correction with function sdreport after fitting the model, default is FALSE.
- silent: to silence the fitting process, default is TRUE.

```
#This can take quite some time, ~30 min depending on the computer
mod_fit<-fit_model(set_data)
```

```
## [1] 0
## [1] "relative convergence (4)"
```

This returns all of the raw output from the model fitting process. As there are two functions to get the predicted random effects and the parameter estimates alongside standard errors, we will only look at the most important output for this point to know if the model successfully fit.

- message: This indicates if the model has successfully converged. The two successful messages would be "relative convergence (4)" or "X and relative convergence [5]". Other possible output include "false convergence [8]", which indicate that the model would not have successfully converged. If the output doesn't exist (NA), then the model did not successfully converge.

1.3 Step 3: Obtaining parameter estimates with standard errors

While the parameter estimates are technically available in the output from step 4, the get_parameters() has been created to obtain them in a more user-friendly fashion. The only thing it needs is:

- return_obj: object returned from step 2.

```
params<-get_parameters(mod_fit)
str(params)
```

```
## 'data.frame':    10 obs. of  2 variables:
##  $ Estimate: num  0.533 0.285 0.153 0.282 1.165 ...
##  $ SE      : num  0.0782 0.1394 0.0404 0.0707 0.0157 ...
```

```
rownames(params)
```

```
##  [1] "sigma_m"       "S"             "sigma_tau"     "sigma_phi"
##  [5] "sigma_epsilon" "sigma_upsilon" "q_I"           "q_R"
##  [9] "p_I"           "p_IR"
```

The row names are the parameters, with estimates in the first column and standard errors in the second column.

1.4 Step 4: Get predicted random effects/processes.

The function get_processes() works in the exact same way as get_parameters().

```
pred_proc<-get_processes(mod_fit)
str(pred_proc)
```

```
## List of 1
##  $ processes:'data.frame':   23 obs. of  6 variables:
##   ..$ B   : num [1:23] 1789 2096 2792 2688 3518 ...
##   ..$ se_B: num [1:23] 476 548 727 698 921 ...
##   ..$ R   : num [1:23] 162 133 211 254 290 ...
##   ..$ se_R: num [1:23] 183 152 234 282 323 ...
##   ..$ m   : num [1:23] 0.2307 0.1846 0.1365 0.0538 0.081 ...
##   ..$ se_m: num [1:23] 0.113 0.0905 0.067 0.0266 0.0399 ...
```

This returns a single list, processes, which contains the predicted biomass, recruitment and natural mortality along with their respective standard errors.