SEBDAM Data Example

Raphaël McDonald

Contents

1 SEBDAM Data Example

This document aims to demonstrate how to fit SEBDAM 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 sf 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_data.RData")
form_data$geometry<-form_data$geometry*1000
sf::st_crs(form_data)<-32619
str(form_data)
```

```
## Classes 'sf' and 'data.frame':   3035 obs. of  6 variables:
##  $ I       : num  25.3 543.1 23.7 448.6 105.2 ...
##  $ IR      : num  0 123.15 0 6.76 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 ...
```

```
##  $ geometry:sfc_POINT of length 3035; first list element:  'XY' num  705019 4843182
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA NA
##   ..- attr(*, "names")= chr [1:5] "I" "IR" "L" "N" ...
```

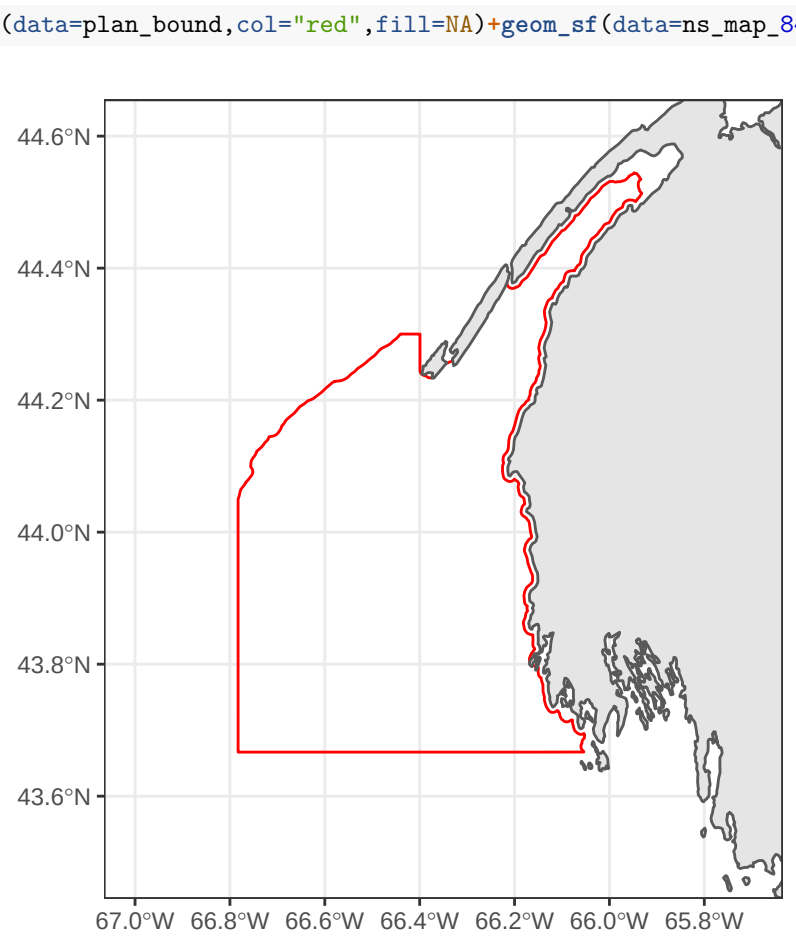
As seen above, each row contains 1 survey tow with its observation of commercial size biomass (column I), the observation of recruit size biomass (column IR), the observed number of clappers (column L), the observed number of shells (live scallops + clappers, column N), the year in which this tow was taken (column Year) and the locations making this data.frame an sf object (column geometry). If the user was not using observations of natural mortality, the data would look very similar except removing the columns L and N.

The only other requirement to get started is to have the polygon/polygons that represent the edges of the modelling area, as shown in the following example:

```
load("./Data/spa3_model_area.RData")
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.0.5
```

```
ggplot()+geom_sf(data=plan_bound,col="red",fill=NA)+geom_sf(data=ns_map_84)+coord_sf(xlim=c(-67,-65.7),
```



The area of this example is on the western shore of Nova Scotia, Canada, with the red polygon showing the area that is included in the modelling approach and with land in grey.

Once we have the modelling polygon and the formatted data, we can take the first step in utilizing SEBDAM.

1.1 Step 1: Setting up predictive knots and mesh

The first step utilizes the function setup_mesh() to obtain the locations of predictive knots based on the locations of each observations, and from those knots create a predictive mesh on which the random fields can be computed. This function requires the following parameters:

- data: Formatted data into an sf data frame, as demonstrated above.
- seed: to set the seed for reproducibility
- nknot: the number of predictive knots to be created (default of 25)
- model_bound: sf polygon/polygons representing the borders of the modelling area (and islands or any land barriers for barrier models)

There are other parameters that have not been fully tested or are unecessary for the usual purposes and are therefore not discussed here. Therefore, we would obtain the knots and mesh the following way:

```
knots_mesh<-setup_mesh(data=form_data,seed=20,nknot=25,model_bound=plan_bound)
```

This gives us the following objects:

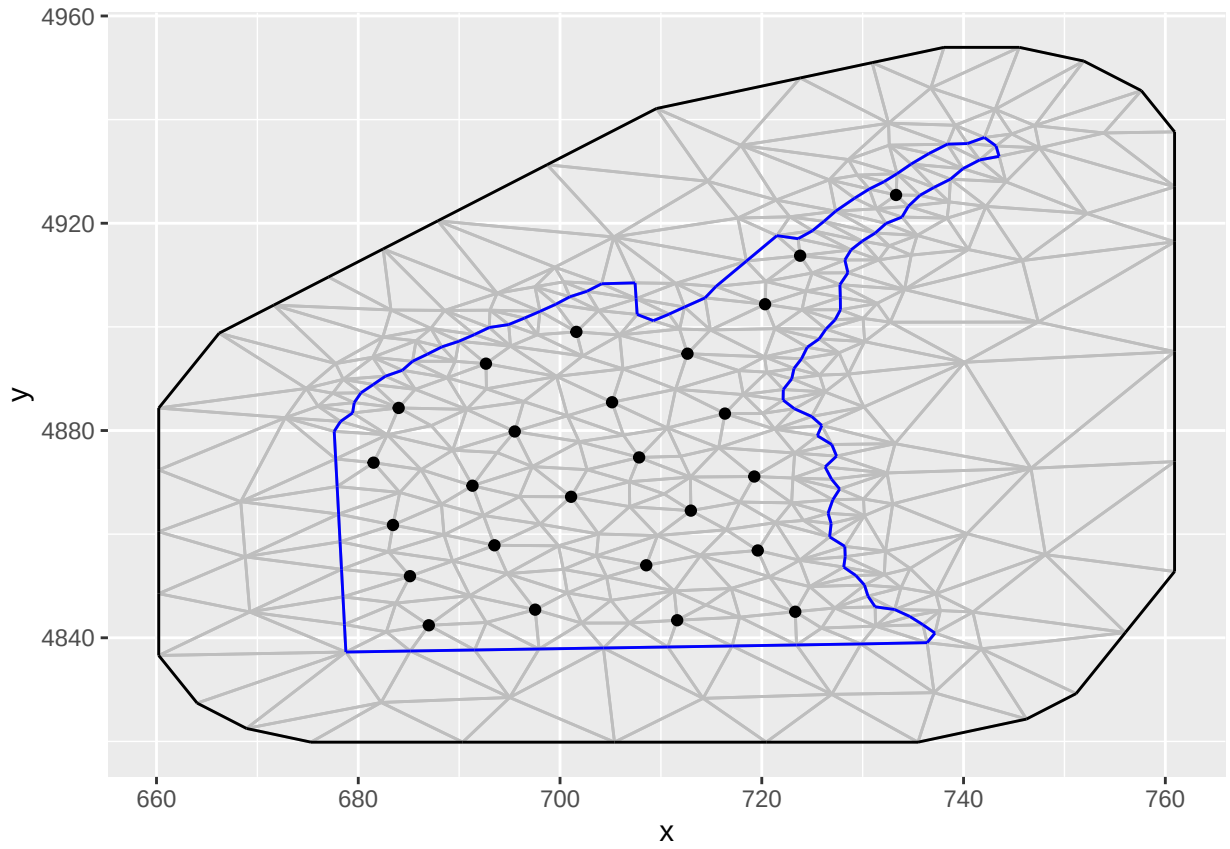
- knots: a kmeans object that includes everything obtained from running kmeans to obtain the knot location, mostly in case the user wants to examine it more closely
- mesh: the predictive mesh used for the random fields created with INLA::inla.mesh.2d.

```
library(inlabru)
```

```
## Warning: package 'inlabru' was built under R version 4.0.5
```

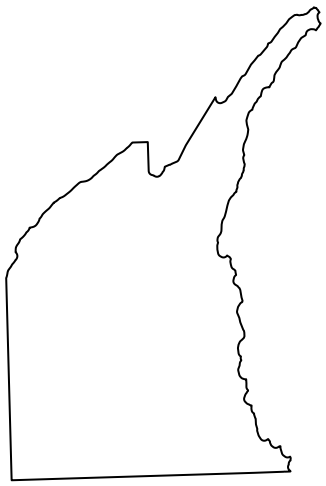
```
## Loading required package: sp
```

```
ggplot()+gg(knots_mesh$mesh)+geom_point(data=as.data.frame(knots_mesh$knots$centers),aes(x=X,y=Y))
```



- crs: utm coordinate reference system that was automatically chosen based on the WGS84 coordinates of data
- utm_bound: sf polygon representing the the modelling projected into the UTM system

```
plot(knots_mesh$utm_bound)
```



1.2 Step 2: Create the predictive grid

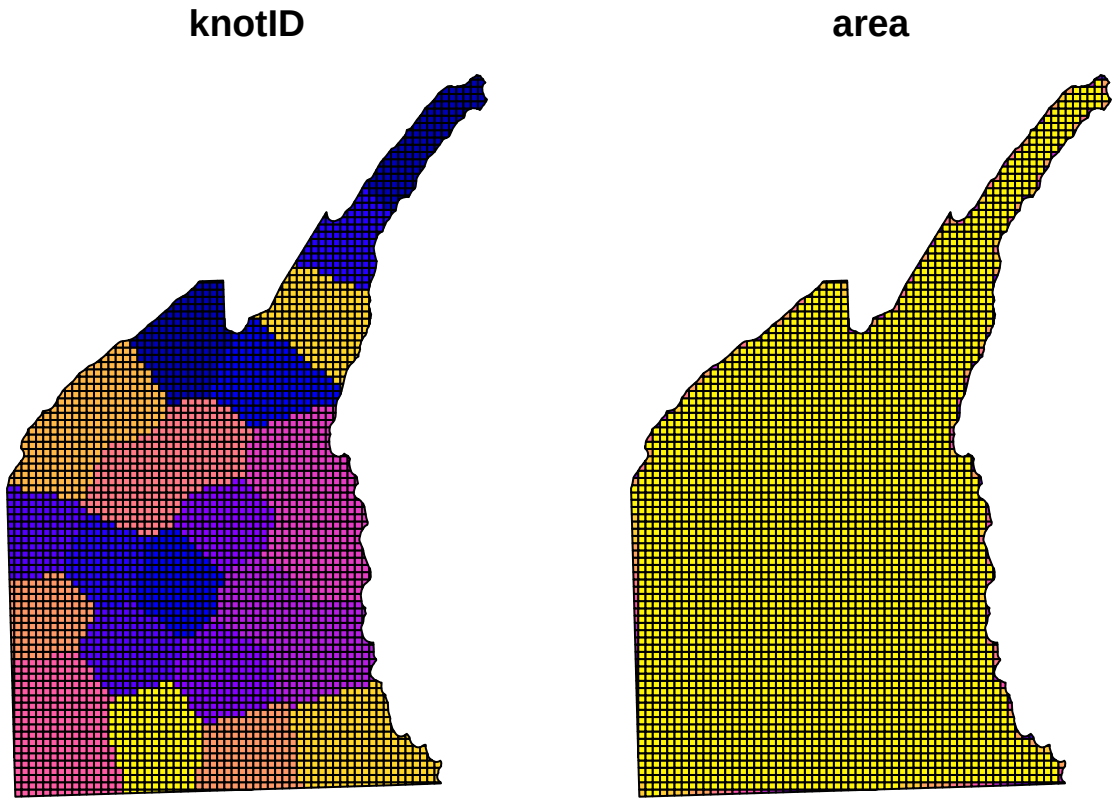
The second step utilizes the function setup_pred_grid() to create the predictive grid used to extrapolate from the densities predicted by the model to the whole area of interest. It simply requires the knots object from Step 1, and the boundaries of the modelling area which can also be obtained from the results of step 1.

```
pred_grid<-setup_pred_grid(knots=knots_mesh$knots,model_bound=knots_mesh$utm_bound)
```

The output has 2 components:

- grid: Contains the sf object used for extrapolation and plotting.

```
plot(pred_grid$grid)
```



- area: contains the area in square kilometers (given that everything follows the default UTM coordinate reference system) associated with each knot.

1.3 Step 3: Setup the data object

The third step, arguably the most important one, is where the data is given to the function data_setup() to format it properly for TMB. It is also the step in which many of the modelling decisions must be made through its required parameters.

- data: formatted dataframe containing the user data, as described earlier in this document.
- growths: a dataframe containing the growth rates in 2 columns: g for commercial size growth rates, and gR for recruit growth rates.

```
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: dataframe of catches attributed to each knots and scaled down to kilometer square. The example given here has already done the work, but there is a function currently being worked on to allow spreading the catch and scaling it appropriately as part of this package.

```
load("./Data/spa3_sim_catch.RData")
str(as.data.frame(sim_catches))
```

```
## 'data.frame':    25 obs. of  23 variables:
##  $ V1 : num  4.21 4.1 4.36 4.06 5.62 ...
##  $ V2 : num  9.61 14.87 7.92 9.41 13.03 ...
##  $ V3 : num  8.87 12.87 8.53 10.16 9.73 ...
##  $ V4 : num  12.6 13.7 10.5 11.2 10.7 ...
##  $ V5 : num  15.2 14.6 14.5 12.5 12.3 ...
##  $ V6 : num  20.8 14.2 14.7 18.1 15.2 ...
##  $ V7 : num  7.31 5.35 9.21 8.6 7.96 ...
##  $ V8 : num  24.6 20.1 18 19.7 18.4 ...
##  $ V9 : num  7.48 9.5 7.79 7.6 6.17 ...
##  $ V10: num  10.31 6.63 5.76 9.58 7.73 ...
##  $ V11: num  13.2 13.8 11 15.1 8.9 ...
##  $ V12: num  7.1 10.63 9.77 12.7 13.98 ...
##  $ V13: num  6.02 6.18 8.39 6.55 6.73 ...
##  $ V14: num  5.55 7.47 3.87 4.1 6.01 ...
##  $ V15: num  8.46 6.26 5 7.13 7.8 ...
##  $ V16: num  18.3 25.4 25.8 14.7 14.3 ...
##  $ V17: num  8.24 14.98 9.81 10.38 6.67 ...
##  $ V18: num  17.2 19.7 17.4 14.9 18.2 ...
##  $ V19: num  12.7 14 12.9 11.6 10.5 ...
##  $ V20: num  16.7 16.1 22.4 16.7 20 ...
##  $ V21: num  5.03 6.06 6.21 6.68 6.76 ...
##  $ V22: num  7.25 8.21 8.72 11.4 7.19 ...
##  $ V23: num  11.32 8.57 9.55 12.72 8.39 ...
```

- model: character string to choose which model, only options are SEBDAM or TLM. The choice here has to be SEBDAM, TLM is covered by a different example file.
- mesh: mesh obtained in step 1.
- bound: only required if using a barrier model, will be ignored in this example.
- obs_mort: whether to fix the instantaneous natural mortality or use observations to predict it. If fixing, obs_mort=FALSE, and the data only needs observations of commercial biomass and recruitment. If predicting, obs_mort=TRUE, and the data needs the number of dead animals L (clappers in this scallop example) and the total number of animals, dead or alive, in the same observation N.
- prior: whether to use a prior beta distribution to inform the estimation of the commercial catchability/ies. prior=TRUE if wants to inform, prior=FALSE to freely estimate.
- prior_pars: only if informing with a beta distribution, specify the two shape parameters desired (default are 10 and 12).
- fix_m: only if obs_mort=FALSE, value at which the instantaneous natural mortality is to be fixed (default at 0.1).
- mult_qI: whether to estimate a single commercial biomass catchability or one per knot. If a single, mult_qI=FALSE, else for multiple mult_qI=TRUE.
- spat_approach: what type of spatial approach to use in the estimation of the Gaussian Markov Random Fields. Options are: "spde" for the normal isotropic Stochastic Partial Differential Equation, "spde_aniso" for the SPDE approach that accounts for geometric anisotropy, and "barrier" for a barrier model using a finite element method.

- knot_obj: kmeans object obtained in step 1.
- knot_area: area attributed to each knot in step 2.
- separate_R_aniso: only if using "spde_aniso" as a spatial choice, whether to estimate separate anisotropy parameters for the recruitment or to have them be the same as for the commercial biomass.
- all_se: whether to obtain the standard error for all the random effects (not getting them speeds up computational speed dramatically). all_se=TRUE to obtain them, all_se=FALSE otherwise.

```
set_data<-data_setup(data=form_data,growths=spa3_growths,catch=as.data.frame(sim_catches),model="SEBDAM
```

This function returns 4 lists necessary for fitting the SEBDAM model through TMB. These are the following:

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

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

```
## List of 22
##  $ model      : chr "SEBDAM"
##  $ options_vec: num [1:7] 0 1 0 1 0 1 0
##  $ prior_pars : num [1:2] 10 12
##  $ logI       : num [1:3035] 3.23 6.3 3.17 6.11 4.66 ...
##  $ logIR      : num [1:3035] NA 4.81 NA 1.91 NA ...
##  $ area       : num [1:25] 102.2 148.2 157.6 124 97.6 ...
##  $ C          : num [1:25, 1:23] 0.0411 0.0276 0.0277 0.0327 0.0575 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : NULL
##   .. ..$ : chr [1:23] "V1" "V2" "V3" "V4" ...
##  $ 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 ...
##  $ n_i        : int 3035
##  $ n_t        : int 22
##  $ n_s        : int 25
##  $ n_m        : int 340
##  $ s_i        : int [1:3035] 18 14 14 15 7 9 11 10 11 12 ...
##  $ t_i        : num [1:3035] 0 0 0 0 0 0 0 0 0 0 ...
##  $ v_i        : num [1:25] 112 113 114 115 116 117 118 119 120 121 ...
##  $ gI         : 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 ...
##  $ mesh_obj   :List of 9
##   ..$ n_s     : int 340
##   ..$ n_tri   : int 644
##   ..$ Tri_Area: num [1:644] 44.37 3.89 5.12 4.17 4.65 ...
##   ..$ E0      : num [1:644, 1:2] 4.471 -2.931 5.434 -1.651 0.719 ...
##   ..$ E1      : num [1:644, 1:2] 4.922 3.276 -0.851 -1.967 1.938 ...
##   ..$ E2      : num [1:644, 1:2] -9.393 -0.345 -4.584 3.617 -2.657 ...
##   ..$ TV      : num [1:644, 1:3] 226 87 85 86 112 35 37 60 221 5 ...
##   ..$ G0      :Formal class 'dgTMatrix' [package "Matrix"] with 6 slots
##   .. .. ..@ i       : int [1:340] 0 1 2 3 4 5 6 7 8 9 ...
##   .. .. ..@ j       : int [1:340] 0 1 2 3 4 5 6 7 8 9 ...
##   .. .. ..@ Dim     : int [1:2] 340 340
```

```
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ x       : num [1:340] 12.45 8.94 9.4 6.35 6.19 ...
##   .. .. ..@ factors : list()
##   ..$ G0_inv  :Formal class 'dgTMatrix' [package "Matrix"] with 6 slots
##   .. .. ..@ i       : int [1:340] 0 1 2 3 4 5 6 7 8 9 ...
##   .. .. ..@ j       : int [1:340] 0 1 2 3 4 5 6 7 8 9 ...
##   .. .. ..@ Dim     : int [1:2] 340 340
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ x       : num [1:340] 0.0803 0.1118 0.1064 0.1574 0.1616 ...
##   .. .. ..@ factors : list()
##  $ 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 SEBDAM's parameters.

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

```
## List of 22
##  $ log_sigma_epsilon: num -1
##  $ log_sigma_upsilon: num -1
##  $ log_R0           : num 4.55
##  $ log_B0           : num 7.64
##  $ log_m0           : num -2.3
##  $ logit_p_I        : num 0
##  $ logit_p_IR       : num 0
##  $ log_qR           : num -1
##  $ omega_B          : num [1:340, 1:23] 0 0 0 0 0 0 0 0 0 0 ...
##  $ omega_R          : num [1:340, 1:22] 0 0 0 0 0 0 0 0 0 0 ...
##  $ log_kappa_B      : num -1
##  $ log_tau_B        : num 1
##  $ log_kappa_R      : num -1
##  $ log_tau_R        : num 1
##  $ log_H_input_B    : num [1:2] 0 0
##  $ log_H_input_R    : num [1:2] 0 0
##  $ log_qI           : num -1
##  $ omega_m          : num [1:340, 1:23] 0 0 0 0 0 0 0 0 0 0 ...
##  $ log_kappa_m      : num 0
##  $ log_tau_m        : num 0
##  $ log_H_input_m    : num [1:2] 0 0
##  $ log_S            : num -1
```

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

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

```
##  chr [1:3] "omega_B" "omega_R" "omega_m"
```

- map: which parameters to fix or ignore, depending on modelling choices. In this example, as we have decided to make both the commercial biomass and the recruitment have the same anisotropy parameters, the recruitment anisotropy parameters are ignored:

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

```
## List of 1
##  $ log_H_input_R: Factor w/ 0 levels: NA NA
```

1.4 Step 4: 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 3.
- 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.

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

```
## Warning in fit_model(set_data): Standard Errors for random fields and densities
## (B, R, and m) will not be calculated, if desired rerun data_setup() with
## all_se=T
```

```
## [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.

1.5 Step 5: 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 4.

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

```
## 'data.frame':    24 obs. of  2 variables:
##  $ Estimate: num  0.249 -0.464 -0.464 4.88 0.053 ...
##  $ SE      : num  0.0856 0.3802 0.3802 1.7351 0.0122 ...
```

```
rownames(params)
```

```
##  [1] "H_B"           "H_B.1"         "H_B.2"         "H_B.3"
##  [5] "kappa_B"       "tau_B"         "kappa_R"       "tau_R"
##  [9] "H_m"           "H_m.1"         "H_m.2"         "H_m.3"
## [13] "kappa_m"       "tau_m"         "qI"            "S"
## [17] "sigma_epsilon" "sigma_upsilon" "R0"            "B0"
## [21] "m0"            "qR"            "p_I"           "p_IR"
```

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

1.6 Step 6: Get predicted random effects/processes.

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

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

```
## Warning in get_processes(mod_fit): Standard errors for processes were not
## calculated as part of fitting process, if desired please rerun data_setup() with
## all_se=T
```

```
str(pred_proc)
```

```
## List of 2
##  $ densities:List of 6
##   ..$ B   : num [1:25, 1:23] 391 345 381 341 381 ...
##   ..$ se_B: logi [1:25, 1:23] NA NA NA NA NA NA ...
##   ..$ R   : num [1:575] 22.7 18.1 18.8 18.5 22 ...
##   ..$ se_R: logi [1:25, 1:23] NA NA NA NA NA NA ...
##   ..$ m   : num [1:25, 1:23] 0.0381 0.0391 0.0146 0.0975 0.026 ...
##   ..$ se_m: logi [1:25, 1:23] NA NA NA NA NA NA ...
##  $ totals   :'data.frame':   23 obs. of  6 variables:
##   ..$ totB     : num [1:23] 1422 1777 2667 2505 3082 ...
##   ..$ se_totB  : num [1:23] 377 469 697 653 805 ...
##   ..$ totR     : num [1:23] 71.2 60.9 96.3 117.1 121.6 ...
##   ..$ se_totR  : num [1:23] 60.9 52.6 82 99.2 101.9 ...
##   ..$ mean_m   : num [1:23] 0.0501 0.0448 0.0328 0.0202 0.0204 ...
##   ..$ se_mean_m: num [1:23] 0.0118 0.01076 0.00776 0.00492 0.00477 ...
```

The output contains 2 main data frames:

- densities: Contains the predicted densities and standard errors for biomass, recruitment and natural mortality.
- totals: contains the predicted overall process for the whole area including total biomass, total recruitment, and mean natural mortality.