SEBDAM Simulation Example

Raphaël McDonald

Contents

1 Simulation Example

This document aims to show a step by step example of how to run a simulation experiment using SEBDAM, including all the decisions that are required to do this.

1.1 Step 1: Simulate Area

The first step is to choose the area that is desired for the modelling exercise, which is done using the function *simulate_area()*. This function takes 2 mandatory arguments:

- n_obs: The number of observations desired
- n_knots: The number of knots desired for the modelling process.

The other optional arguments are the following:

- seed: Setting the seed to be able to reproduce the knot locations
- x_coords and y_coords: Coordinates used to create the simulation area (only allows single continuous polygon, so cannot be used for simulating barrier models)
- area: sf object, present if users want to create more complex objects than allowed by x_coords and y_coords (e.g. complex islands and land boundaries for barrier models).

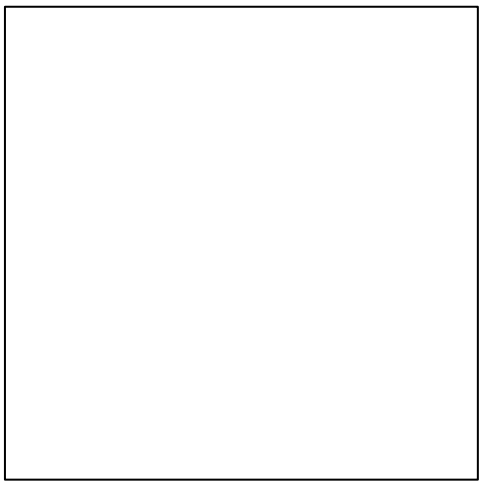
If x_coords and y_coords are not specified and area is not specified, the function defaults to creating a 50X50km square as the modelling area, as was done for the simulation study in McDonald et al. (Submitted). If specifying the coordinates in order to create a different polygon, it must be a single continuous polygon. For more complex areas, they must be created outside of SEBDAM and input into simulate_area() with the area parameter. No matter what area is simulated, its units must be in kilometer for the model to make sense. Furthermore, if not using the default, the user is required to provide extra parameters to create the mesh (e.g. max.edge, cutoff, max.n, etc.) The following R code shows the default, alongside other commented out approaches if using either different coordinates or a user-provided modelling area:

```
areasim<-simulate_area(n_obs=1000,n_knots=20,seed=20)
#areasim<-simulate_area(n_obs=1000,n_knots=20,seed=20,x_coord=c(175,175,225,250,225),
#y_coord=c(175,225,225,225,175),max.edge=c(8,30),cutoff=2)
#areasim<-simulate_area(n_obs=1000,n_knots=20,seed=20,area=your_sf_area,max.edge=c(10,25),cutoff=1)
```

We can now examine everything that comes out of this function, all of which are required for the next functions in approaching simulations:

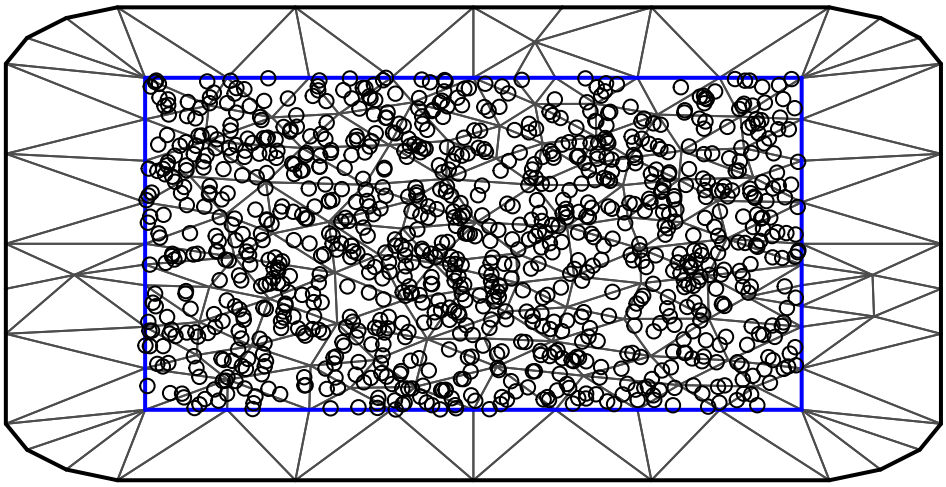
- area: sf polygon representing the modelling area

```
plot(areasim$area)
```



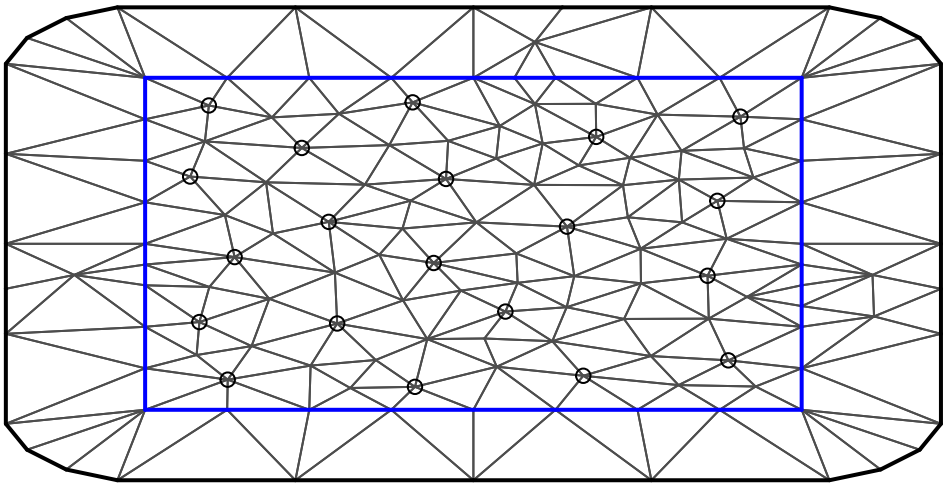
- knots: kmeans object from which the knot locations are obtained for closer examination if the user desires it
- mesh: INLA mesh created based on the knots created previously.
- sim_obs: sf points for the locations of all the simulated data points

```
plot(areasim$mesh)
plot(areasim$sim_obs,add=T)
```



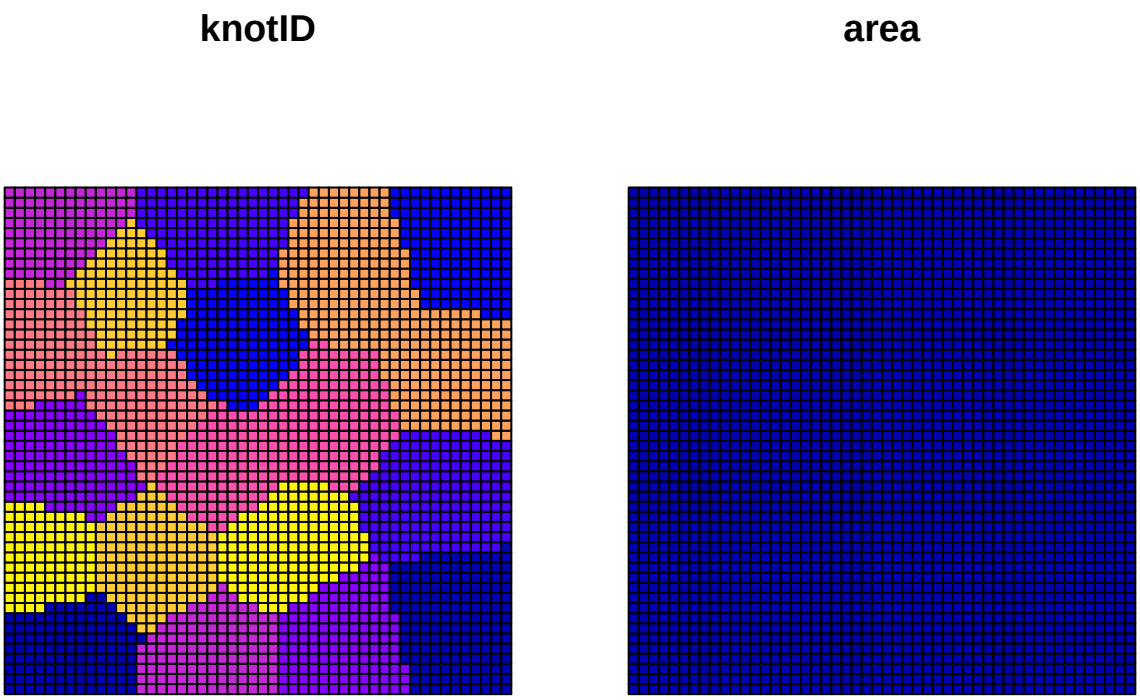
- stratarea: area covered by each knot
- knots_sf: sf points for the locations of the knots

```
plot(areasim$mesh)
plot(areasim$knots_sf,add=T)
```



- grid: grid used for predictions and for plotting purposes

```
plot(areasim$grid)
```



1.2 Step 2: Simulation Object

The second step is to create the object in which TMB will simulate data and random effects. Many of the modelling decisions are made at this stage through the *simulate_obj()* function. These parameters are not all required for SEBDAM (as some are specific to TLM), and those that are not will be ignored here.

- model: Choose between SEBDAM or TLM model
- n_years: The number of years in which to split the total number of observations simulated in step 1.
- obs_mort: TRUE if we want to simulate observations of natural mortality as is possible in scallop fisheries with clappers, FALSE otherwise (and by default).
- even_spread: TRUE to split the total number of observations evenly in each year, FALSE to introduce some variability in the number of observations per year
- fix_m: only required if obs_mort is FALSE, and is the value at which we want to fix the natural mortality
- gI: commercial size growth rates
- gR: recruit growth rates
- prior_q: TRUE if user desires to use a beta distribution to inform the commercial size catchability qI
- catch_spread: choice of simulation approach for landings. If left NULL, no catch is simulated, other choices are proportional (prop), aggregated into knots with higher than average biomass (aggregated), or aggregated the same way with low levels of landings in all knots below average (aggregated_extra).

- simul_area_obj: object simulated in step 1 with simulate_area()
- mult_qI: choice of using separate commercial catchabilities at each knots (TRUE) or a single overall one (FALSE)
- spat_approach: choice between 3 possible spatial approaches: the stochastic partial differential equations approach ("spde"), the spde approach that allows for geometric anisotropy ("spde_aniso"), or a barrier model ("barrier").
- separate_R_aniso: only used if the spatial approach is spde_aniso, allows the user to specify if they want the anisotropy of the recruitment and biomass to be the same (FALSE) or recruitment to have its own separate anisotropy parameters (TRUE).
- bound: only used if the spatial approach is a barrier model, sf object that represents the land boundaries (including islands) of the modelling area

```
simobj<-simulate_obj(model="SEBDAM",n_years=10,obs_mort=FALSE,even_spread=FALSE,fix_m=0.25,
                     gI=rep(1.2,10),gR=rep(1.3,10),prior_q=TRUE,catch_spread="prop",
                     simul_area_obj = areasim,mult_qI=FALSE,spat_approach="spde")
#Examples if using different approaches
#simobj<-simulate_obj(model="SEBDAM",n_years=10,obs_mort=TRUE,even_spread=FALSE,prior_q=TRUE,
#catch_spread="aggregated",simul_area_obj = areasim,mult_qI=TRUE,
#spat_approach="spde_aniso",separate_R_aniso=TRUE)
#simobj<-simulate_obj(model="SEBDAM",n_years=10,obs_mort=FALSE,even_spread=FALSE,prior_q=TRUE,
#catch_spread="aggregated_extra",simul_area_obj = areasim,mult_qI=FALSE,
#spat_approach="barrier",bound=my_sf_obj)
```

This outputs the object that can be used to simulate new data, i.e. the skeleton of the upcoming simulations.

```
str(simobj)
```

```
## List of 20
##  $ model      : chr "SEBDAM"
##  $ options_vec: num [1:7] 1 1 0 0 0 0 0
##  $ prior_pars : num [1:2] 10 12
##  $ logI       : num [1:1000] 1 1 1 1 1 1 1 1 1 1 ...
##  $ logIR      : num [1:1000] 1 1 1 1 1 1 1 1 1 1 ...
##  $ area       : num [1:20] 113 155 130 127 168 125 121 133 98 117 ...
##  $ C          : num [1:20, 1:11] 1 1 1 1 1 1 1 1 1 1 ...
##  $ n_tows     : num [1:10] 81 128 120 92 110 62 108 71 109 119
##  $ pos_tows_I : num [1:10] 1 1 1 1 1 1 1 1 1 1
##  $ pos_tows_IR: num [1:10] 1 1 1 1 1 1 1 1 1 1
##  $ n_i        : int 1000
##  $ n_t        : num 10
##  $ n_s        : int 20
##  $ n_m        : int 147
##  $ s_i        : int [1:1000] 4 9 3 4 3 8 3 19 15 6 ...
##  $ t_i        : int [1:1000] 0 0 0 0 0 0 0 0 0 0 ...
##  $ v_i        : num [1:20] 36 37 38 39 40 41 42 43 44 45 ...
##  $ gI         : num [1:10] 1.2 1.2 1.2 1.2 1.2 1.2 1.2 1.2 1.2 1.2
##  $ gR         : num [1:10] 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3
##  $ mesh_obj   :List of 3
##   ..$ M0:Formal class 'dgTMatrix' [package "Matrix"] with 6 slots
##   .. .. ..@ i       : int [1:147] 0 1 2 3 4 5 6 7 8 9 ...
```

```
##   .. .. ..@ j       : int [1:147] 0 1 2 3 4 5 6 7 8 9 ...
##   .. .. ..@ Dim     : int [1:2] 147 147
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ x       : num [1:147] 55.8 53.9 55.4 51.8 38.3 ...
##   .. .. ..@ factors : list()
##   ..$ M1:Formal class 'dgTMatrix' [package "Matrix"] with 6 slots
##   .. .. ..@ i       : int [1:961] 0 19 29 36 112 125 126 127 1 18 ...
##   .. .. ..@ j       : int [1:961] 0 0 0 0 0 0 0 0 1 1 ...
##   .. .. ..@ Dim     : int [1:2] 147 147
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ x       : num [1:961] 4.252 -1.349 -1.386 -0.132 -0.494 ...
##   .. .. ..@ factors : list()
##   ..$ M2:Formal class 'dgTMatrix' [package "Matrix"] with 6 slots
##   .. .. ..@ i       : int [1:2525] 0 9 14 19 29 36 64 83 101 112 ...
##   .. .. ..@ j       : int [1:2525] 0 0 0 0 0 0 0 0 0 0 ...
##   .. .. ..@ Dim     : int [1:2] 147 147
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : NULL
##   .. .. .. ..$ : NULL
##   .. .. ..@ x       : num [1:2525] 0.4063 0.0219 0.029 -0.2048 -0.2029 ...
##   .. .. ..@ factors : list()
```

1.3 Step 3: Choosing Model Parameters

The third step involves choosing the values of the parameters used by SEBDAM using *simulate_SEBDAM_params()*. These are the last preliminary decisions that are required before actually simulating data and random effects. Parameters are the following:

- simul_data_obj: simulation object obtained by step 2 using *simulate_obj()* function
- sigma_epsilon: commercial biomass observation variance (default: 0.1).
- sigma_upsilon: recruit biomass observation variance (default: 0.1).
- R0: mean recruitment density in year 1 of simulations (default: 100 kg/km^2).
- B0: mean commercial biomass density in year 1 of simulations (default: 500 kg/km^2).
- m0: if using mortality observations (obs_mort=TRUE in Step 2), represents mean instantaneous natural mortality in year 1 of simulations (default: 0.1). If not, sets the true value of the natural mortality for the simulations.
- p_I: probability of positive commercial size biomass observations (default: 0.9).
- p_IR: probability of positive recruit biomass observations (default: 0.4).
- qR: recruit catchability (default: 0.2).
- range_B: distance at which two points with commercial size biomass stop being correlated (default: 40km).
- range_R: distance at which two points with recruit size biomass stop being correlated (default: 30km).

- sigma_B: the commercial biomass random field marginal variance (default: 0.1).
- sigma_R: the recruit biomass random field marginal variance (default: 0.1).
- H_input_B: only required if using spde_aniso, anisotropy parameters for commercial biomass (default: -1.2 and -0.5).
- H_input_R: only required if using spde_aniso AND specifying that recruitment has separate anisotropy parameters (separate_R_aniso=TRUE), anisotropy parameters for recruitment (default: -1 and -0.8).
- qI: commercial size catchability (default: 0.45), if using multiple catchabilities (mult_qI=TRUE) they MUST be specified.
- range_m: only required if using observations of natural mortality (obs_mort=TRUE), distance at which two points with natural mortality stop being correlated (default: 20km).
- sigma_m: only required if using observations of natural mortality (obs_mort=TRUE), the natural mortality random fields marginal variance (default: 0.1).
- H_input_m: only required if using observations of natural mortality (obs_mort=TRUE) and using spde_aniso, anisotropy parameters for natural mortality (default: -0.5 and 0.4).
- S: only required if using observations of natural mortality (obs_mort=TRUE), clapper catchability (default: 0.4).

Since all parameters have defaults, if the user is happy with those defaults then it is not necessary to specify anything.

```
#If defaults are good and using a single qI
# simpar<-simulate_SEBDAM_params(simobj)
#If using multiple qIs
#simpar<-simulate_SEBDAM_params(simobj,qI=rbeta(20,10,12))
#Other examples:
simpar<-simulate_SEBDAM_params(simobj,sigma_epsilon=0.5,sigma_upsilon=0.5,sigma_B=0.2)
```

1.4 Step 4: Simulate Data

Now that the modelling decisions have been made, the actual data can be simulated using the *simulate_data()* function which requires the following:

- simul_obj: object obtained from *simulate_SEBDAM_params()* function in step 3.
- seed: to set the simulation seed for reproducibility
- format: choose between default formatted data (data formatted as is necessary for model fitting functions, "formatted") or raw data ("raw") where everything is returned inside lists as understood by TMB.
- sim_obs_loc: locations of observations, part of the list returned by *simulate_area()* in step 1.

```
simdata<-simulate_data(simpar,seed=20,sim_obs_loc=areasim$sim_obs)
```

```
## Constructing atomic log_dbinom_robust
## Constructing atomic D_lgamma
## Constructing atomic log_dbinom_robust
## Constructing atomic D_lgamma
```

The raw format (not shown here) simply fills out the lists in the object obtained from *simulate_SEBDAM_params()* in step 3 with simulated data. The formatted data contains 4 lists:

- data: This contains the simulated observations formatted properly to fit SEBDAM to:

```
str(simdata$data)
```

```
## Classes 'sf' and 'data.frame':   1000 obs. of  5 variables:
##  $ I       : num  241 0 323 88 223 ...
##  $ IR      : num  0 0 0 0 0 ...
##  $ Year    : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Knot    : num  4 9 3 4 3 8 3 19 15 6 ...
##  $ geometry:sfc_POINT of length 1000; first list element:  'XY' Named num  219 196
##   ..- attr(*, "names")= chr [1:2] "lon" "lat"
##  - attr(*, "sf_column")= chr "geometry"
##  - attr(*, "agr")= Factor w/ 3 levels "constant","aggregate",..: NA NA NA NA
##   ..- attr(*, "names")= chr [1:4] "I" "IR" "Year" "Knot"
```

- growths: contains the commercial size and recruit growth rates as specified by the user in *simulate_obj()* in step 2.

```
str(simdata$growths)
```

```
## 'data.frame':    10 obs. of  2 variables:
##  $ g : num  1.2 1.2 1.2 1.2 1.2 1.2 1.2 1.2 1.2 1.2
##  $ gR: num  1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3 1.3
```

- processes: contains the simulated underlying processes. Not used in model fitting, but necessary to compare the model output at the end.

```
str(simdata$processes)
```

```
## List of 5
##  $ density_B: num [1:20, 1:11] 457 682 625 525 430 ...
##  $ density_R: num [1:20, 1:10] 137.2 79.9 94.2 83.4 95.3 ...
##  $ totals   :'data.frame':   11 obs. of  2 variables:
##   ..$ B: num [1:11] 1345 1927 2363 2232 2830 ...
##   ..$ R: num [1:11] 240 243 252 256 238 ...
##  $ omega_B  : num [1:147, 1:11] -0.0764 -0.3025 0.2626 0.4385 0.1763 ...
##  $ omega_R  : num [1:147, 1:10] 0.2624 0.0222 0.013 -0.2249 0.1333 ...
```

- catch: contains appropriately formatted simulated landings

```
str(simdata$catch)
```

```
## 'data.frame':    20 obs. of  11 variables:
##  $ V1 : num  51.7 65.3 82.1 62.4 28.3 ...
##  $ V2 : num  75.5 52.9 73.2 150.9 73.2 ...
##  $ V3 : num  218.3 40.2 112.8 80.4 91.5 ...
##  $ V4 : num  90.2 154.2 48.6 34 100.1 ...
```

```
##  $ V5 : num  423.5 64 157.4 86.7 84.6 ...
##  $ V6 : num  270.9 163.3 71 52.4 130.3 ...
##  $ V7 : num  92.6 51.6 65.6 87.9 157.2 ...
##  $ V8 : num  173 66.5 49.7 69.4 104.5 ...
##  $ V9 : num  183.2 76.7 55.8 132.8 72.3 ...
##  $ V10: num  189.4 72.6 54.6 156.9 67.4 ...
##  $ V11: num  0 0 0 0 0 0 0 0 0 0 ...
```

We now have data simulated from the SEBDAM equations that can be fitted to it again or for any other purposes. For details of how to use the model fitting functions, see *Fitting SEBDAM to Data.pdf* example.

2 Simulation Study Example

This is an example of how one could test out SEBDAM to see the impact of using the "wrong" spatial approach (wrong in that the simulations used a different one) on the outcome, from start to finish:

```
#Setting up to simulate data
areasim<-simulate_area(n_obs=1000,n_knots=20,seed=20)
simobj<-simulate_obj(model="SEBDAM",n_years=10,obs_mort=FALSE,even_spread=FALSE,fix_m=0.25,
                     gI=rep(1.2,10),gR=rep(1.3,10),prior_q=TRUE,catch_spread="prop",
                     simul_area_obj = areasim,mult_qI=FALSE,spat_approach="spde_aniso")
simpar<-simulate_SEBDAM_params(simobj,sigma_epsilon=0.5,sigma_upsilon=0.5,sigma_B=0.2)
#Setup for running the simulations multiple time
n_sim<-10
sim_data_list<-list()
mod_fit_list<-list()
parameters_list<-list()
pred_pro_list<-list()
#Running the simulations (can take up to 40-50 min if
#getting all standard errors, closer to 10-20 without)
for (i in 1:n_sim){
  simdata<-simulate_data(simpar,sim_obs_loc=areasim$sim_obs)
  sim_data_list[[i]]<-simdata
  dat_set<-data_setup(data=simdata$data,growths= simdata$growths,
                      catch=simdata$catch,model="SEBDAM",
                      mesh=areasim$mesh,bound=areasim$area,
                      obs_mort=FALSE,prior=TRUE,prior_pars=c(10,12),
                      mult_qI=FALSE,
                      spat_approach = "spde",
                      knot_obj = areasim$knots,
                      knot_area = areasim$stratarea,
                      all_se=F)
  mod_fit<-fit_model(dat_set,optim="optimr",
                       optim_method="nlminb",
                      control=list(eval.max=10000,iter.max=10000),
                      silent=T)
  mod_fit_list[[i]]<-mod_fit
  parameters<-get_parameters(mod_fit)
  parameters_list[[i]]<-parameters
  pred_processes<-get_processes(mod_fit)
  pred_pro_list[[i]]<-pred_processes
}
```

```
## Warning in fit_model(dat_set, optim = "optimr", optim_method = "nlminb", :
## 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)"

## 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

## Warning in get_processes(mod_fit): 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)"

## 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

## Warning in get_processes(mod_fit): 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)"

## 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

## Warning in get_processes(mod_fit): 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)"

## 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

## Warning in get_processes(mod_fit): 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)"
```

```
## 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

## Warning in get_processes(mod_fit): 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)"

## 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

## Warning in get_processes(mod_fit): 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)"

## 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

## Warning in get_processes(mod_fit): 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)"

## 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

## Warning in get_processes(mod_fit): 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)"

## 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

## Warning in get_processes(mod_fit): 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)"
```

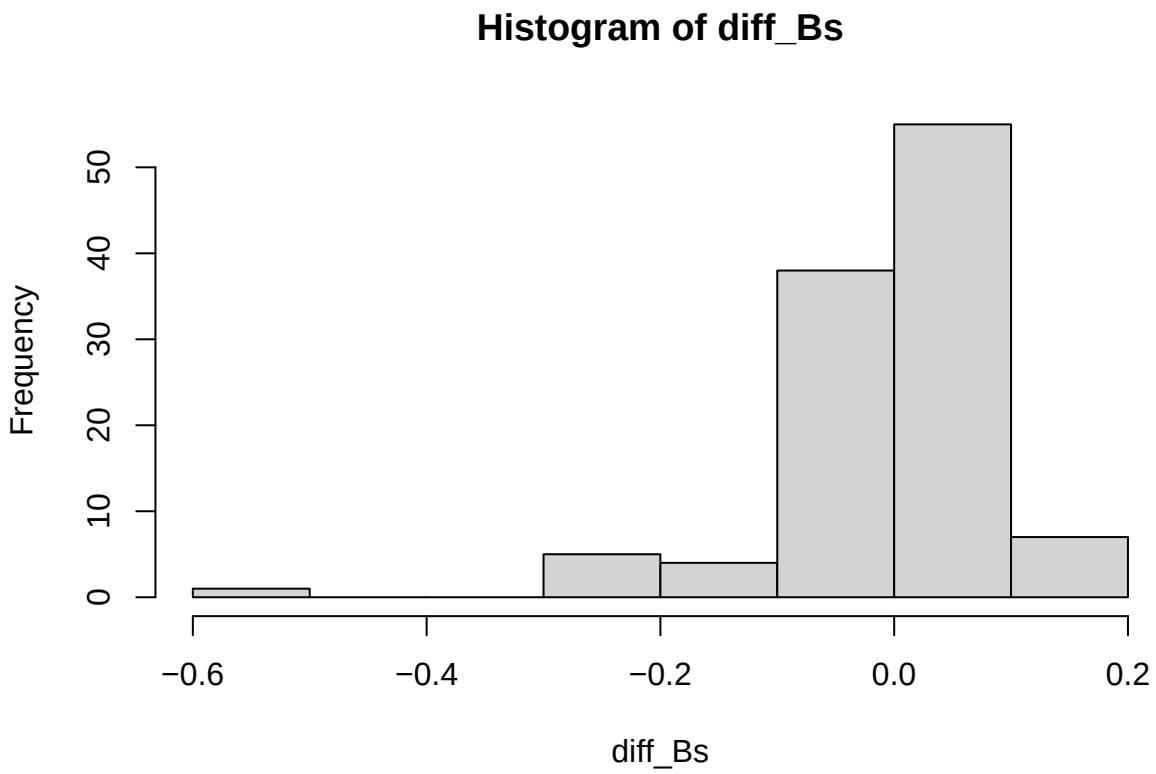
```
## Warning in get_processes(mod_fit): Standard errors for processes were not
## calculated as part of fitting process, if desired please rerun data_setup() with
## all_se=T
```

To see how the model is impacted by fitting an spde approach to data simulated with spde_aniso, we can see how well it captures the simulated biomass. However, it is important to only look at those fits that were successful, and remove those that did not converge or gave false convergence.

```
#ADD X AND relative convergence together
good_fits<-c()
for (i in 1:n_sim){
  if (mod_fit_list[[i]]$opt$message=="relative convergence (4)") {
    good_fits<-c(good_fits,i)
  }
}

diff_Bs<-matrix(rep(NA,n_sim*11),ncol=n_sim)
for (i in good_fits){
  sim_val<-sim_data_list[[i]]$processes$totals$B
  pred_val<-pred_pro_list[[i]]$totals$totB
  diff_Bs[,i]<-(sim_val-pred_val)/sim_val
}

hist(diff_Bs)
```




```
median(diff_Bs)
```

```
## [1] 0.004978413
```

```
mean(diff_Bs)
```

```
## [1] -0.002481233
```