Example #1: Calibration Protocol - Time-Averaged

Julia L. Blanchard

July 2020

Calibrating a multi-species model to time-averaged species' catches

In this example we will explore how we can learn about models by fitting size spectrum ecological models to data using the "mizer" R package.

Recall, there are three different kinds of size spectrum models in mizer, of increasing complexity:

1) community model: purely size-based and representative of a single but "average" species across the whole community.

2) trait-based model, which disaggregates the size spectrum into different groups with different life-histories, through differences in each "species" asymptotic which determines other life-history parameters such as the size at maturity (Hartvig et al. 2011, Andersen & Pedersen, 2010).

3) multispecies model - which has the same equations and parameters as the trait-based model but is parameterised to represent multiple species in a real system, where each species can have many differing species-specific traits (Blanchard et al. 2014).

Here we focus on multispecies size spectrum models. In practice, these models have been parametrised in a few different ways depending on data availability for a system or research questions.

Some studies have focused on many species-specific values, for example where each species have different values of life-history, size-selective feeding trait parameters (e.g. β and σ), and details of species interactions (Blanchard et al. 2014, Reum et al. 208) to better capture the dynamics of marine food webs.

Others, such as Jacobsen et al. (2014, 2016), have represented variation in only a couple of the most important life history parameters for each species - asymptotic size (which links to other parameters such as maturation size) and recruitment parameters (R_{max}, $eRepro$) to broadly capture fished communities or carry out across ecosystem comparisons.

Once you have parametrised the multispecies model for your system (section 1), you may find that species do not coexist or the biomass or catches are very different from your observations. After the model is parameterised and assessed for basic principles and coexistence (section 3), further calibration to observational data is used to ensure the relative abundance of each species reflects the system (section 4), at least during a stable period, which is time-averaged.

The background resource parameters and the recruitment parameters, $Rmax$ (maximum recruitment) and $erepro$ (reproductive efficiency) greatly affect the biomasses of species in your system.

The recruitment parameters are highly uncertain and capture density dependent processes in the model that limit the number of offspring that successfully recruit to the smallest size class for each species. In the default mizer package these parameters are used to implement an emergent Beverton-Holt type stock recruitment relationship.

As a starting point, we will estimate these parameters as a means of fitting the modelled species catches to the observed catches. This could similarly be carried out with biomasses. Other model detailed approaches also exist, see the main paper, but this approach has been used to get models in the right "ball-park", which can them be further evaluated using diagnostics (example X) and fitted to time series data (example XX).

A Simple Protocol for Multispecies Model Calibration

We will adapt the "recipe" for calibration in Jacobsen et al 2014 (see supp. mat.) and Blanchard et al (2014), into the following steps:

0. Run the model with the chosen species-specific parameters. This will relate some of the missing parameters to w_{inf} (h and γ - explain through simple example of how the model works?). R_{max} (see example that explains R_{max}?) could also be automatically calculated based on equilbrium assumptions (Andersen et al. 2016) but by default it is "Inf", which means there is no density dependence associated with spawner-recruit dynamics (RF: default is the 2016 method at the moment but setting up at Inf to start with something that doesn't coexist below).

1. Obtain the time-averaged data (e.g. catches or biomasses for each species) and the time-averaged fishing mortalty inputs (e.g. from stock assessments). Typically this should be over a stable part of the time series for your system.

2. Start with the chosen parameters for κ and λ of the resource spectrum that are obtained from the literature regarding the community size spectrum. These can be very uncertain and sometimes are not available. Calibrate the carrying capacity of the background resource spectrum, κ, by examining the feeding level, biomass through time, and overall size spectrum.

3. Calibrate the maximum recruitment, R_{max}, which will affect the relative biomass of each species (and, combined with the fishing parameters, the catches) by minimising the error between observed and estimated catches (again or biomasses).

4. Check that the physiological recruitment, RDI, is much higher than the realised recruitment, RDD. This can be done using the `getRDD()` and `getRDI()` functions and calculating the ratio which should be around 100 for a species with $w_{inf} = 1500g$ (e.g. Whiting/Plaice), but varies with asymptotic size and fishing mortality (Andersen 2019). High RDI/RDD ratio indicates the carrying capacity is controlling the population rather than predation or competition. Larger species often require more of this density dependent control than smaller ones. If RDI/RDD is too high, the efficiency of reproduction (*erepro*) can be lowered to ensure species do not outcompete others or or over-resilient to fishing. Lowering *erepro* biologically means higher egg mortality rate or wasteful energy invested into gonads. If $RDI/RDD = 1$ the species is in the linear part of the stock recruitment relationship (no spawner-recruit density dependence).

5. Verify the model after the above step by comparing the model with: species biomass or abundance distributions, feeding level, natural mortality, growth, vulnerablity to fishing (fmsy) and catch, diet composition. Many handy functions for plotting these are available here: https://sizespectrum.org/mizer/reference/index.html

6. The final verification step is to force the model with time-varying fishing mortality to assess whether changes in time series in biomassess and catches capture observed trends. The model will not cpature all of the fluctuations from environmental processes (unless some of these are included), but should match the magnitude and general trend in the data. (RF: this is going to be Example2)

Step 0. Run the model with the chosen species-specific parameters.

In this section you will:

- obtain or create a dataframe of species-specific parameters
- run the dataframe through the `mizer` package and examine the model output

A species-specific dataframe is already stored in `mizer`, which contains the North Sea Model Parameters (RF however is probably not updated so using the .csv in the repository)

```
# if user has not installed the requird packages
# install.packages("tidyverse")
# install.packages("plotly")
#devtools::install_github("sizespectrum/mizer")
#devtools::install_github("sizespectrum/mizerExperimental")

library(mizerExperimental) # for projectToSteady()
library(mizer)
library(tidyverse)
library(plotly)

# loading North Sea data
nsParams <- read.csv("data/nsparams.csv")[,-1]

# This data frame already has Rmax values, let's remove them to calibrate them again later
nsParams[,"r_max"] <- Inf

# If you want to make it less multi-species and more trait-based model
# nsParams[,"beta"] <-100
# nsParams[,"sigma"] <-1.5
```

```
params_uncalibrated <- newMultispeciesParams(nsParams, inter, kappa = 1e11, max_w=1e6) # inter comes wi
```

```
## Note: No h provided for some species, so using f0 and k_vb to calculate it.

## Note: Because you have n != p, the default value is not very good.

## Note: No ks column so calculating from critical feeding level.

## Note: Using z0 = z0pre * w_inf ^ z0exp for missing z0 values.

## Note: Using f0, h, lambda, kappa and the predation kernel to calculate gamma.
```

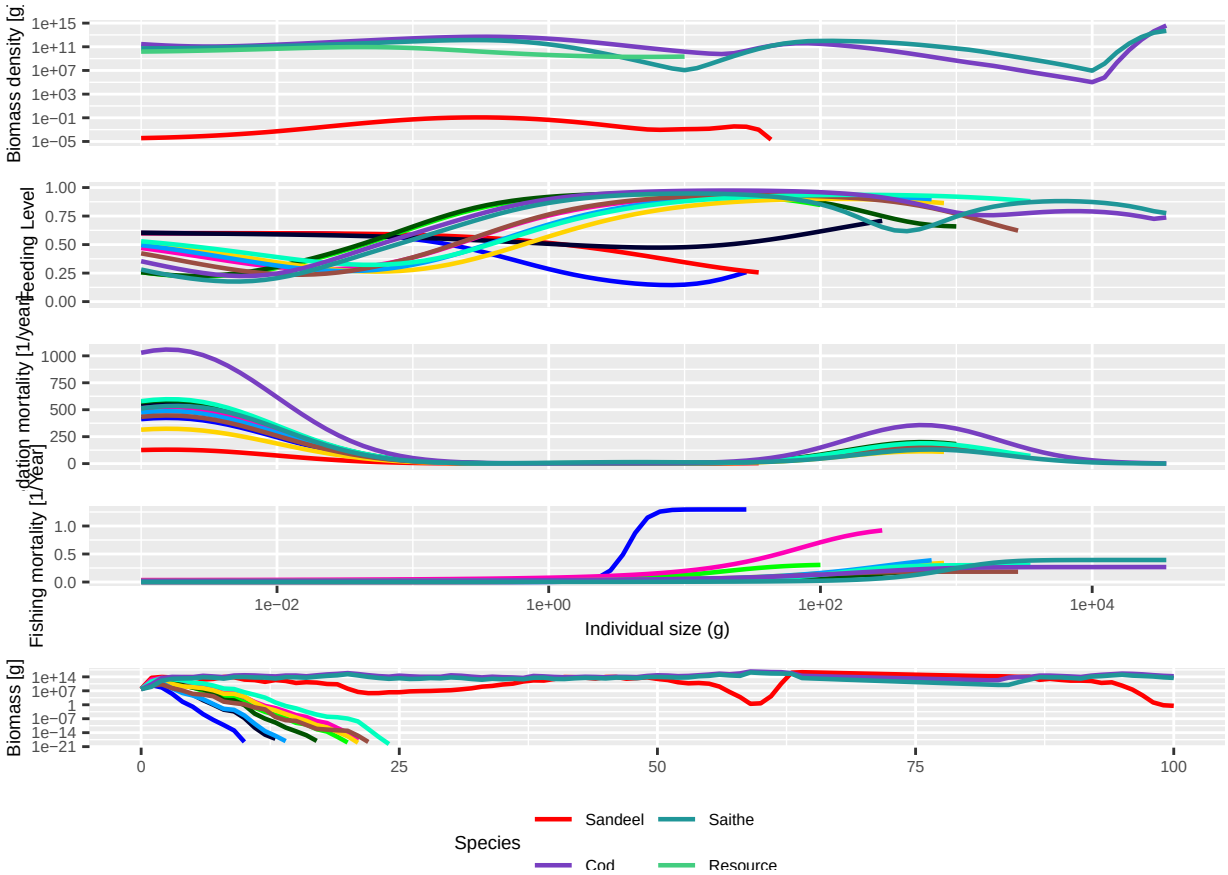
```
# note the volume of this model is set to the reflect the entire volume of the North Sea - hence the ve

#  Add other params for info
#  param$Volumecubicmetres=5.5e13    #unit of volume. Here total volume of North sea is used (Andersen

# have a look at species parameters that have been calculated
# params_uncalibrated@species_params

# alternative params without redundant parameters to reduce the size of the dataframe on the screeen
params_uncalibrated@species_params[,-which(colnames(params_uncalibrated@species_params) %in% c("sel_fun
```

```
##          X1 species   w_inf w_mat   beta sigma R_max  k_vb  l25  l50     a     b
## Sprat     1   Sprat    33.0    13  51076   0.8   Inf 0.681  7.6  8.1 0.007 3.014
## Sandeel   2 Sandeel    36.0     4 398849   1.9   Inf 1.000  9.8 11.8 0.001 3.320
## N.pout    3  N.pout   100.0    23     22   1.5   Inf 0.849  8.7 12.2 0.009 2.941
## Herring   4 Herring   334.0    99 280540   3.2   Inf 0.606 10.1 20.8 0.002 3.429
## Dab       5     Dab   324.0    21    191   1.9   Inf 0.536 11.5 17.0 0.010 2.986
## Whiting   6 Whiting  1192.0    75     22   1.5   Inf 0.323 19.8 29.0 0.006 3.080
## Sole      7    Sole   866.0    78    381   1.9   Inf 0.284 16.4 25.8 0.008 3.019
## Gurnard   8 Gurnard   668.0    39    283   1.8   Inf 0.266 19.8 29.0 0.004 3.198
## Plaice    9  Plaice  2976.0   105    113   1.6   Inf 0.122 11.5 17.0 0.007 3.101
## Haddock  10 Haddock  4316.5   165    558   2.1   Inf 0.271 19.1 24.3 0.005 3.160
## Cod      11     Cod 39851.3  1606     66   1.3   Inf 0.216 13.2 22.9 0.005 3.173
## Saithe   12  Saithe 39658.6  1076     40   1.1   Inf 0.175 35.3 43.6 0.007 3.075
##          catchability w_min alpha         n   p         q        h k       ks
## Sprat     1.29533333 0.001   0.6 0.6666667 0.7 0.7166667 14.46675 0 1.593753
## Sandeel   0.06510547 0.001   0.6 0.6666667 0.7 0.7166667 25.62741 0 2.936414
## N.pout    0.31380000 0.001   0.6 0.6666667 0.7 0.7166667 31.20422 0 3.372902
## Herring   0.18150000 0.001   0.6 0.6666667 0.7 0.7166667 28.36363 0 2.920263
## Dab       0.97800000 0.001   0.6 0.6666667 0.7 0.7166667 34.87720 0 3.781368
## Whiting   0.24266667 0.001   0.6 0.6666667 0.7 0.7166667 31.77220 0 3.301616
## Sole      0.37383333 0.001   0.6 0.6666667 0.7 0.7166667 24.73805 0 2.567302
## Gurnard   0.46250569 0.001   0.6 0.6666667 0.7 0.7166667 20.64990 0 2.193126
## Plaice    0.18483333 0.001   0.6 0.6666667 0.7 0.7166667 16.94072 0 1.740765
## Haddock   0.30150000 0.001   0.6 0.6666667 0.7 0.7166667 41.46028 0 4.196598
## Cod       0.26666749 0.001   0.6 0.6666667 0.7 0.7166667 69.40226 0 6.511735
## Saithe    0.39300000 0.001   0.6 0.6666667 0.7 0.7166667 59.95343 0 5.700788
##                 z0        gamma     w_mat25 erepro
## Sprat   0.18705957 5.652974e-11   11.647460      1
## Sandeel 0.18171206 3.790575e-11    3.583834      1
## N.pout  0.12926608 9.750228e-11   20.607045      1
## Herring 0.08647736 2.514308e-11   88.699888      1
## Dab     0.08735805 7.579184e-11   18.815128      1
## Whiting 0.05658819 9.927702e-11   67.196884      1
## Sole    0.06294752 5.184323e-11   69.884760      1
## Gurnard 0.06863713 4.638552e-11   34.942380      1
## Plaice  0.04171321 4.489177e-11   94.075638      1
## Haddock 0.03685027 7.707429e-11  147.833146      1
## Cod     0.01756590 2.325827e-10 1438.909287      1
## Saithe  0.01759431 2.435635e-10  964.051303      1
```

```r
# lets' change the plotting colours
# looks good but hard to distinguish some species
# library(viridis)
# params_uncalibrated@linecolour[1:12] <-plasma(dim(params_uncalibrated@species_params)[1])

# easier to read plots
library(pals)
params_uncalibrated@linecolour[1:12] <-glasbey(dim(params_uncalibrated@species_params)[1])

params_uncalibrated@linecolour["Resource"] <-"seagreen3"

# run with fishing
sim_uncalibrated <- project(params_uncalibrated, t_max = 100, effort = 1)
```

```
plotSummary(sim_uncalibrated, short = T)
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.

## Warning: Removed 24 row(s) containing missing values (geom_path).

## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.

## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.

## Warning: Removed 24 row(s) containing missing values (geom_path).
```



Oh dear, all of the species but three have collapsed! This is because there was no density dependence (R_{max} is set at Inf) and the largest species (cod and saithe) have outcompeted all of the rest.

Step 1. Obtain the time-averaged data (e.g. catches or biomasses for each species) and the time-averaged fishing mortalty inputs (e.g. from stock assessments).

In this section you will:

- Download fisheries data and process them in a format comparable to the model output

The following .csv are extracted from the ICES database using "data/getICESFishdata_param.R". Fishing data is averaged over 2014-2019 as it's a relatively stable period in catches.

```
# fisheries mortality F
fMat <- read.csv("data/fmat.csv")
fMatWeighted <- read.csv("data/fmatWeighted.csv") # Sandeel and Cod have multiple data base so average

# read in time-averaged  catches
catchAvg <-read.csv("data/time-averaged-catches.csv") # only that one is used at the moment | catches a

# ssb
ssbAvg <- read.csv("data/time-averaged-SSB.csv")
```

Step 2. Calibrate the carrying capacity of the background resource spectrum, κ, at steady state

In this section you will:

- guess reasonable *erepro* and R_{max} values which will stop out-competition from a few species
- vary κ values until you reach coexistence for all species
- check species' growth curve, which is influenced by κ to see if it diverged from the Von Bertalanffy curves (data poor case) or growth data (data rich case)

```
# the fishing mortality rates are already stored in the param object as
params_uncalibrated@species_params$catchability
```

```
##  [1] 1.29533333 0.06510547 0.31380000 0.18150000 0.97800000 0.24266667
##  [7] 0.37383333 0.46250569 0.18483333 0.30150000 0.26666749 0.39300000
```

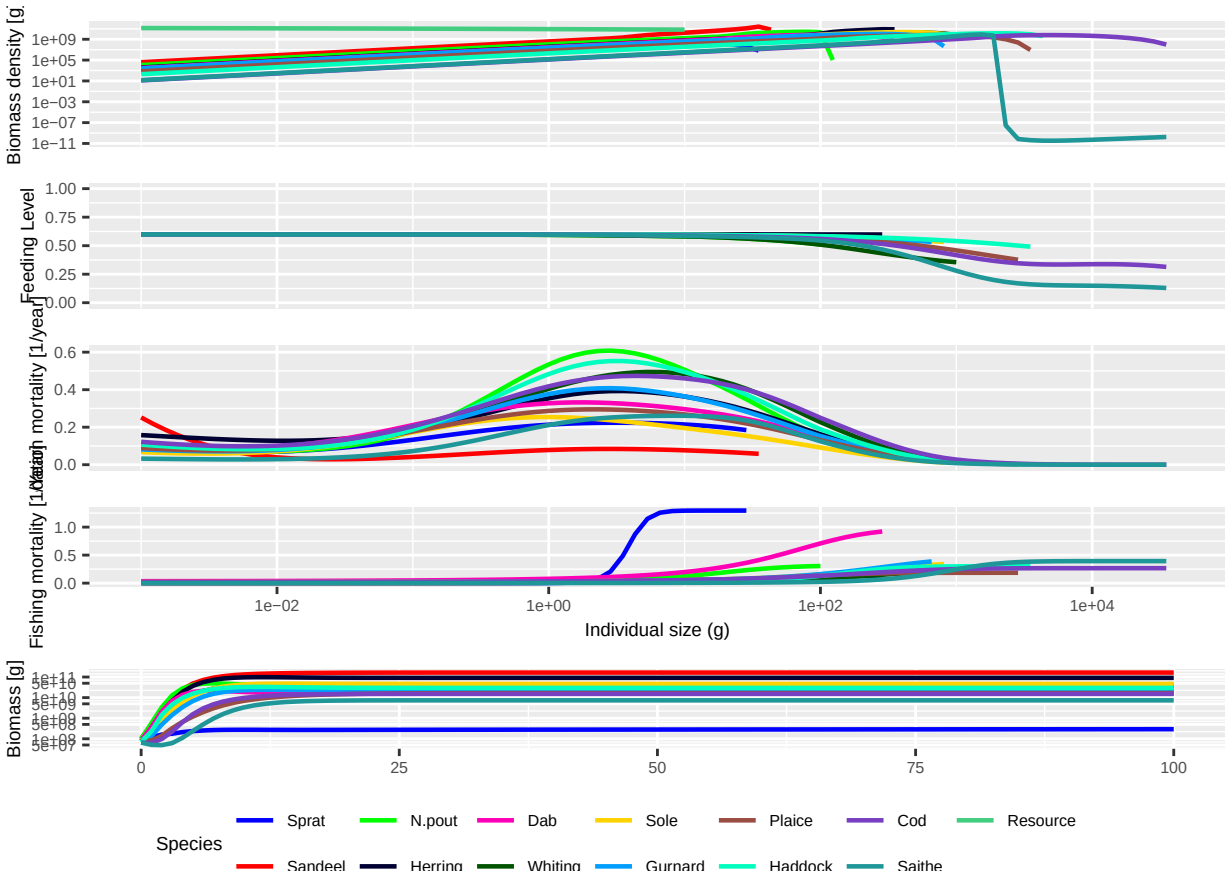
```
# let's start again and replace with the initial pre-calibration "guessed" Rmax
params_guessed <- params_uncalibrated
# penalise the large species with higher density dependence
params_guessed@species_params$R_max <- params_guessed@resource_params$kappa*params_guessed@species_para
# and reduce erepro
params_guessed@species_params$erepro <- 1e-3

params_guessed <- setParams(params_guessed)
# run with fishing
sim_guessed <- project(params_guessed, t_max = 100, effort =1)
plotSummary(sim_guessed, short = T)
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```

```
## Warning: Removed 23 row(s) containing missing values (geom_path).
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```

```
## Warning: Removed 23 row(s) containing missing values (geom_path).
```



Here, Sprat's biomass is orders of magnitude lower than the other species and so are Saithe's largest individuals, but at least species coexist.

In `mizerExperiental`, the `projectToSteady()` function looks for a biomass equilibrium state for a set of intial parameters (RF: is that right?). Let's see how it behaves with our model:

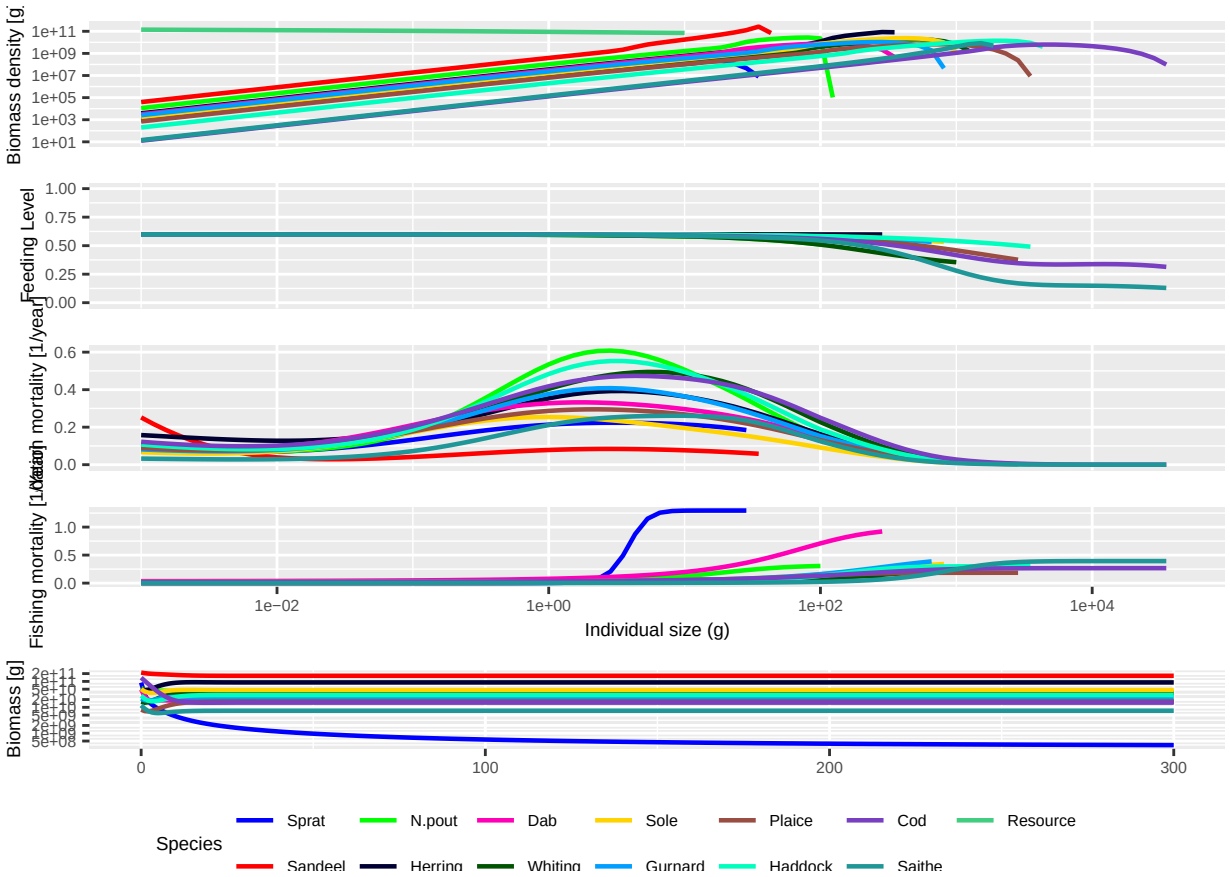
```
## compare with Gustav's projectToSteady
params_steady <- projectToSteady(params_guessed, t_max = 100, return_sim = F)
```

```
## Convergence was achieved in 61.5 years.
```

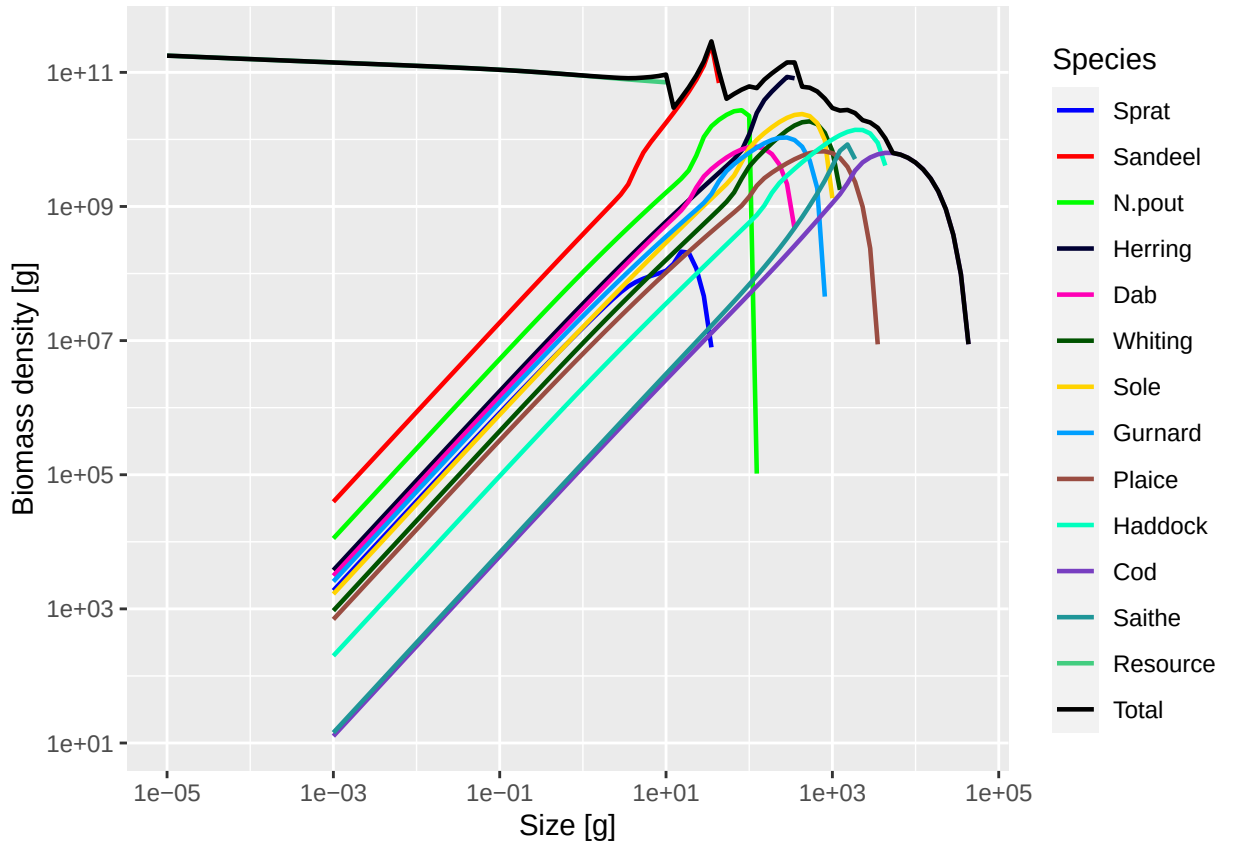
```
sim_steady <- project(params_steady, t_max = 300, effort =1)
plotSummary(sim_steady, short = T)
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
```

```
## Warning: Removed 23 row(s) containing missing values (geom_path).

## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.

## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.

## Warning: Removed 23 row(s) containing missing values (geom_path).
```




```
# zoom on the size spectrum
plotSpectra(sim_steady,power=2,total = T)
```



Species are coexisting (but Sprat's still low). This is in part because we applied a stronger R_{max} effect for larger species. You can play with the above parameters but but it would take a lot of trial an error to achieve the right combination to get the biomass or catches similar to the observations.

We could explore the effects further using Rshiny app, where we also have a plot of the biomass or catch data. First let's look at the basic diagnostics and tune κ and *erepro* to make sure the feeding levels are high enough for each species and that biomasses coexist.

RF: Rshiny section disabled for pdf

```
# Optional
# adjust Rmax and/or reproductive efficiency to examine whether improved steady state is achievable tha
library(shiny) # no need if runtime = shiny is in the YAML
# runApp("shiny-equilibrium")
# is there a way to save the final chosen values?
params_shiny <- params_guessed

shinyApp(

  ui=fluidPage(

  # Application title
  titlePanel("North Sea Model Example"),

  fluidRow(
    column(4, wellPanel(
       sliderInput("kappa", "log10 Resource Carrying Capacity:", min = 8, max = 12, value = 10.7,
                   step = 0.1),
```

```
    #   sliderInput("Rmax", "log10 Maximum Recruitment:", min = 1, max = 12, value = 12,
    #               step = 0.1),
       sliderInput("erepro", "log10 Reproductive Efficiency:", min = -8, max = 1, value = -3,
                   step = 0.1)
          )),
    column(6,
           plotOutput("distPlot", width = 600, height = 600)
    ))



  ),

  server = function(input, output) {

  output$distPlot <- renderPlot({
    # set up params using values given, need check and change parameter values so units work in days un
    params_shiny@species_params$erepro <- rep(10^input$erepro,12)
   # params@species_params$Rmax <- rep(10^input$Rmax,12)
    params_shiny <- setParams(params_shiny,kappa=10^input$kappa)
    # run without fishing
    sim_shiny <- project(params_shiny, effort = 1, t_max = 100)
    plot(sim_shiny)
     })

},

  options = list(height = 500)
)
```

Rshiny makes it easier to fine tune one parameter at a time, but we need to make some species-specific adjustments.

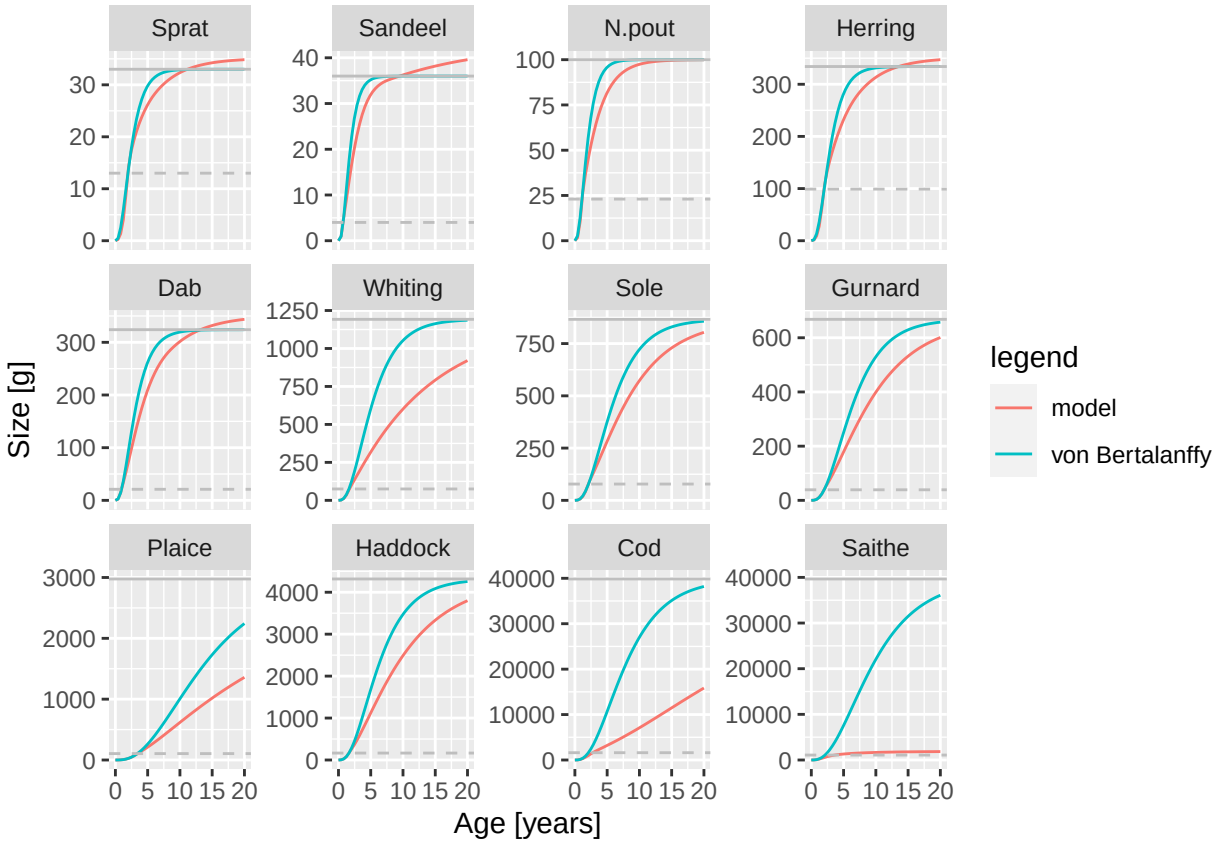
The shiny app helps with understanding the model but it is tricky to arrive at the best fit especially if we want to change several species parameter combinations at a time.

However, varying κ will affect the species' growth curve, we need to check the modelled growth curves agaisnt the "observed" von Bertlanffy growth curves in case varying κ divert too much the emergent growth curves.

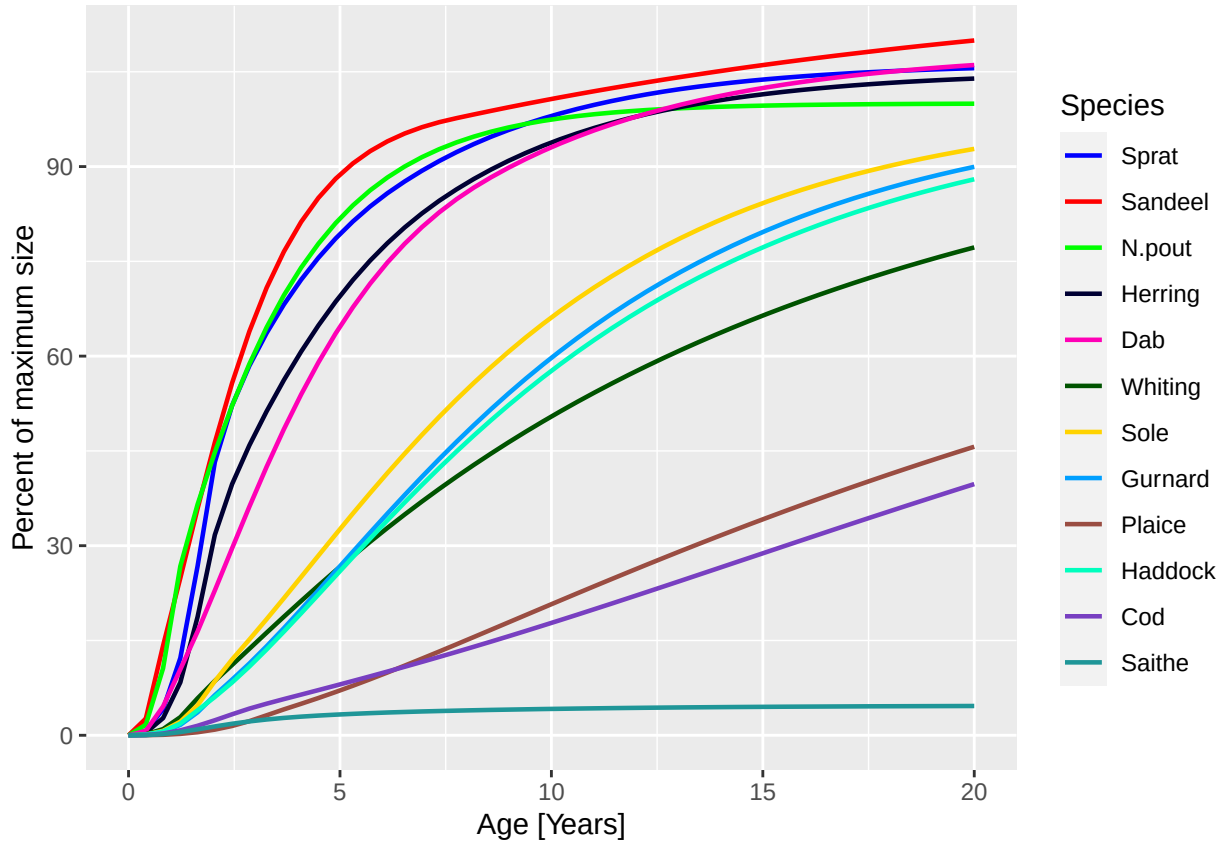
```
# This function is going to be integrated in sizespectrum/mizer
plotGrowthCurves <- function (object,
                              species,
                              max_age = 20,
                              percentage = FALSE,
                              species_panel = FALSE,
                              highlight = NULL)
{
    if (is(object, "MizerSim")) {
        params <- object@params
        t <- dim(object@n)[1]
        params@initial_n[] <- object@n[t, , ] # Designed to work also with single species
        params@initial_n_pp <- object@n_pp[t, ]
    } else if (is(object, "MizerParams")) {
        params <- validParams(object)
```

```
}
if (missing(species)) {
    species <- params@species_params$species
}
ws <- getGrowthCurves(params, species, max_age, percentage)
plot_dat <- reshape2::melt(ws)
plot_dat$Species <- factor(plot_dat$Species, params@species_params$species)
plot_dat$legend <- "model"

# creating some VB
if (all(c("a", "b", "k_vb") %in% names(params@species_params)))
{
    VBdf <- data.frame("species" = params@species_params$species, "w_inf" = params@species_params$w
    VBdf$L_inf <- (VBdf$w_inf / VBdf$a)^(1/VBdf$b)
    plot_dat2 <- plot_dat
    plot_dat2$value <- apply(plot_dat,1,function(x, VBdf){VBdf[which(VBdf$species == x[1]),]$a *
            (VBdf[which(VBdf$species == x[1]),]$L_inf * (1 - exp(-VBdf[which(VBdf$species == x[1]),
                                                                 (as.numeric(x[2]) - VBdf[which
    plot_dat2$legend <- "von Bertalanffy"
    plot_dat <- rbind(plot_dat,plot_dat2)
}

p <- ggplot(filter(plot_dat, legend == "model")) + geom_line(aes(x = Age, y = value,
                                                                 colour = Species, linetype = Speci
y_label <- if (percentage)
    "Percent of maximum size"
else "Size [g]"
linesize <- rep(0.8, length(params@linetype))
names(linesize) <- names(params@linetype)
linesize[highlight] <- 1.6
p <- p + scale_x_continuous(name = "Age [Years]") +
    scale_y_continuous(name = y_label) +
    scale_colour_manual(values = params@linecolour) +
    scale_linetype_manual(values = params@linetype) +
    scale_size_manual(values = linesize)

# starting cases now
if(!percentage)
{

    if (length(species) == 1) {
        idx <- which(params@species_params$species == species)
        w_inf <- params@species_params$w_inf[idx]
        p <- p + geom_hline(yintercept = w_inf, colour = "grey") +
            annotate("text", 0, w_inf, vjust = -1, label = "Maximum")
        w_mat <- params@species_params$w_mat[idx]
        p <- p + geom_hline(yintercept = w_mat, linetype = "dashed", colour = "grey") +
            annotate("text", 0, w_mat, vjust = -1, label = "Maturity")
        if ("von Bertalanffy" %in% plot_dat$legend)
            p <- p + geom_line(data = filter(plot_dat, legend == "von Bertalanffy"), aes(x = Age, y

    } else if(species_panel) # need to add either no panel if no param for VB or create a panel wit
    {
```

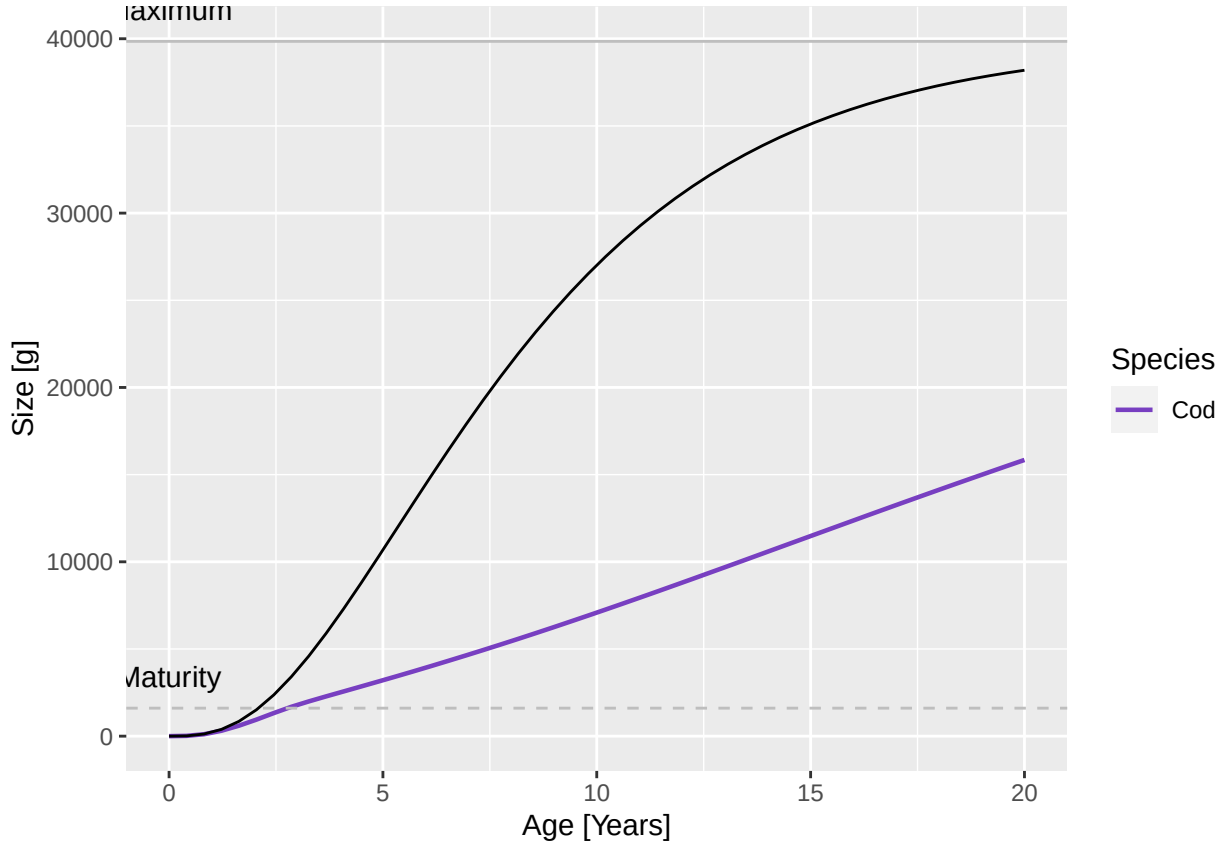
```
            p <- ggplot(plot_dat) +
                geom_line(aes(x = Age, y = value , colour = legend)) +
                scale_x_continuous(name = "Age [years]") +
                scale_y_continuous(name = "Size [g]") +
                geom_hline(aes(yintercept = w_mat),
                           data = tibble(Species = object@params@species_params$species[],
                                         w_mat = object@params@species_params$w_mat[]),
                           linetype = "dashed",
                           colour = "grey") +
                geom_hline(aes(yintercept = w_inf),
                           data = tibble(Species = object@params@species_params$species[],
                                         w_inf = object@params@species_params$w_inf[]),
                           linetype = "solid",
                           colour = "grey") +
                facet_wrap(~Species, scales = "free_y")

        }
    }
    return(p)
}

# All emergent and observed growth curves as pannels
plotGrowthCurves(sim_guessed, species_panel = T)
```



```
# All emergent growth curves together
plotGrowthCurves(sim_guessed, percentage = T)
```




```
# One by one observed vs emergent
plotGrowthCurves(sim_guessed, species = "Cod")
```



To conclude this section, we are going to choose some values that enable the most species to coexist as a starting point for optimisation. Note we won't vary *erepro* at the same time as R_{max} (they depend on each other). However we will use the value of *erepro* selected from the shiny app.

Step 3. Calibrate the maximum recruitment

In this section you will:

- use a package that will calibrate R_{max} per species

Rmax will affect the relative biomass of each species (and, combined with the fishing parameters, the catches) by minimising the error between observed and estimated catches or biomasses. We could also include κ in our estimation here (as in Blanchard et al 2104 & Spence et al 2016) but instead we will use the value that seemed OK in terms of feeding levels in the Rshiny app, roughly $log10(11.5)$. Same goes for *erepro*, a value of $1e-3$ seemed ok.

First let's set up a function running the model and outputing the difference between predicted catches (`getYield()`) and actual catches (`catchAvg`). `err` is the sum of squared errors between the two.

```
# we need 12 Rmaxs, log10 scale
vary <- log10(params_steady@species_params$R_max)
#vary<-runif(10,3,12) # or use completley made up values, same for each species test for effects of ini

## set up the enviornment to keep the current state of the simulations
state <- new.env(parent = emptyenv())
```

```
state$params <-  params_steady

catchAvg <-read.csv("data/time-averaged-catches.csv") # only that one is used at the moment | catches a


## the following getError function combines the steps of the optimisastion above - this time with the m

## update below with project_steady and saving the state from each iteration
#RF the function takes a bunch of RMax and compare the theoretical catches versus data
getError <- function(vary,params,dat,env=state,data_type="catch", tol = 0.1,timetorun=10) {

  #env$params@species_params$R_max[]<-10^vary[1:12]
  params@species_params$R_max[]<-10^vary[1:12]

  params <- setParams(params)
  # run to steady state and update params
  # env$params<- projectToSteady(env$params, distance_func = distanceSSLogN,
  #                  tol = tol, t_max = 200,return_sim = F)
  params<- projectToSteady(params, distance_func = distanceSSLogN,
                   tol = tol, t_max = 200,return_sim = F)

  # create sim object

  sim <- project(params, effort = 1, t_max = timetorun) #Change t_max to determine how many years the m

  #
  # sim <- project(env$params, effort = 1, t_max = timetorun) #Change t_max to determine how many years
  #
  # env$params <-sim@params
  #

          ## what kind of data and output do we have?
          if (data_type=="SSB") {
          output <-getSSB(sim)[timetorun,]   #could change to getBiomass if using survey, also check un
          }

          if (data_type=="catch") {
         output <-getYield(sim)[timetorun,]/1e6
         #' using n . w . dw so g per year per volume (i.e. North Sea since kappa is set up this way).
         #'The data are in tonnes per year so converting to tonnes.
          }

  pred <- log(output)
  dat  <- log(dat)

  # sum of squared errors, here on log-scale of predictions and data (could change this or use other er
   discrep <- pred - dat

   discrep <- (sum(discrep^2))

  # can use a strong penalty on the error to ensure we reach a minimum of 10% of the data (biomass or c
  # if(any(pred < 0.1*dat)) discrep <- discrep + 1e10
```

```
    return(discrep)

   }

## test it
err<-getError(vary = vary, params = params_steady, dat = catchAvg$Catch_1419_tonnes)
```

```
## Convergence was achieved in 24 years.
```

```
# err<-getError(vary,params,dat=rep(100,12),data_type="biomass")
err
```

```
## [1] 85.82931
```

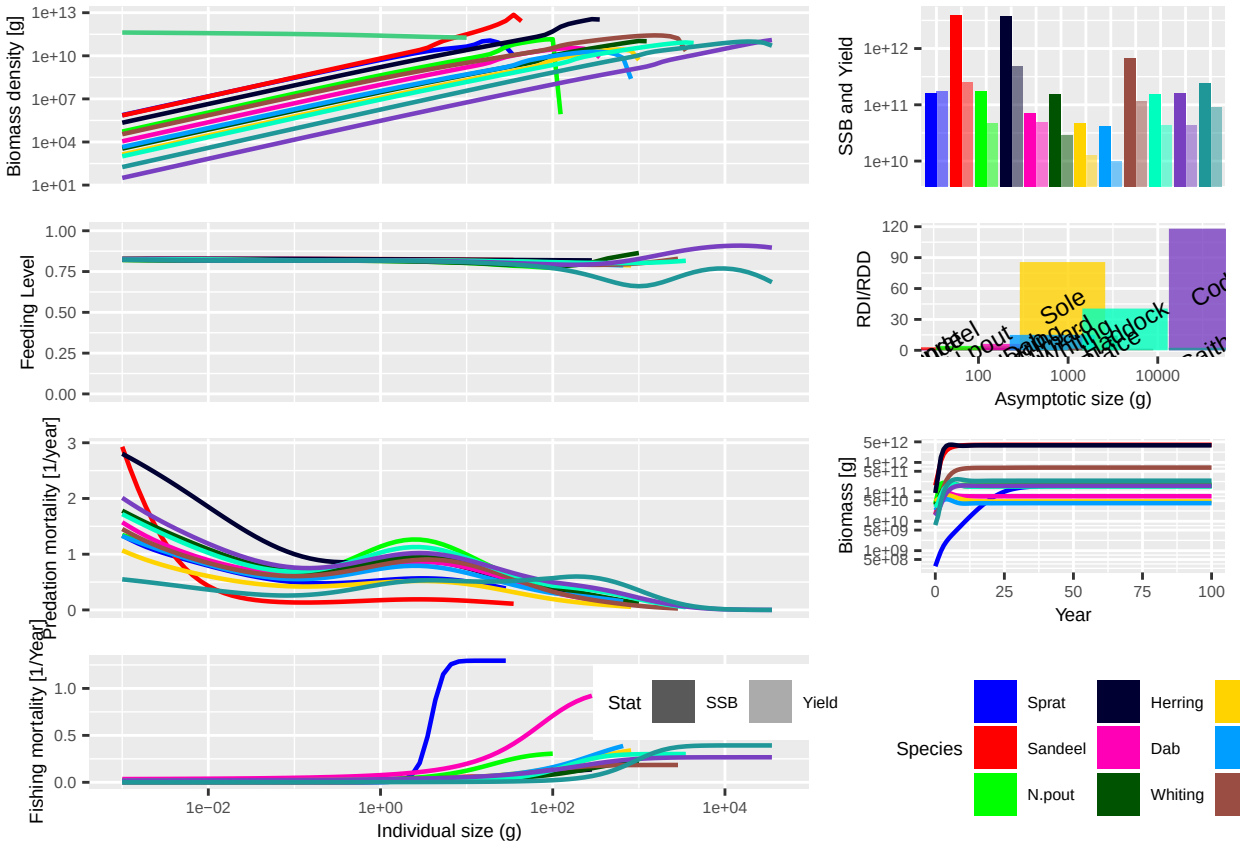
Now, carry out the optimisation. There are several optimisation methods to choose from - we need to select the most robust one to share here. The R package optimParallel seems to be the most robust general R package and has replaced optim. Often this requires repeateing the proceure several times but the advantage of using parallel run is the speed compared to packages such as optimx.

This might take AWHILE. The output is saved as "optim_para_result" if you wish to skip this block.

```
library("parallel")
library("optimParallel")
library("tictoc")

# change kappa and erepro based on shiny exploration, set up initial values based on "close to" equilib
# params_steady already set to erepro = 0.001 and kappa = 10^11

params_optim <- params_guessed
vary <-  log10(params_optim@species_params$R_max)


params_optim@resource_params$kappa<-3.2e11 # better kappa estimated from Rshiny
params_optim<-setParams(params_optim)

noCores <- detectCores() - 1 # keep a spare core

cl <- makeCluster(noCores, setup_timeout = 0.5)
setDefaultCluster(cl = cl)
clusterExport(cl, as.list(ls()))
clusterEvalQ(cl, {
  library(mizerExperimental)
  library(optimParallel)
})

tic()
optim_result <-optimParallel(par=vary,getError,params=params_optim, dat = catchAvg$Catch_1419_tonnes, m
                             parallel=list(loginfo=TRUE, forward=TRUE))

stopCluster(cl)
toc() # 80'' using 47 cores
saveRDS(optim_result,"optim_para_result.RDS")
```

```r
# if previous block not evaluated | have some issue enabling "runtime:shiny" and loading .csv somehow
params_optim <- params_guessed
params_optim@resource_params$kappa<-3.2e11

optim_result <- readRDS("optim_para_result.RDS")
# optim values:
params_optim@species_params$R_max <- 10^optim_result$par

# set the param object
params_optim <-setParams(params_optim)
sim_optim <- project(params_optim, effort = 1, t_max = 100, dt=0.1,initial_n = sim_guessed@n[100,,],ini
saveRDS(sim_optim,"optim_para_sim.RDS")
plotSummary(sim_optim)
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.

## Warning: Removed 24 row(s) containing missing values (geom_path).

## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.

## Warning: Removed 24 row(s) containing missing values (geom_path).

## Warning: Removed 12 rows containing missing values (geom_text).
```



Step 4. Check the level of density dependence.

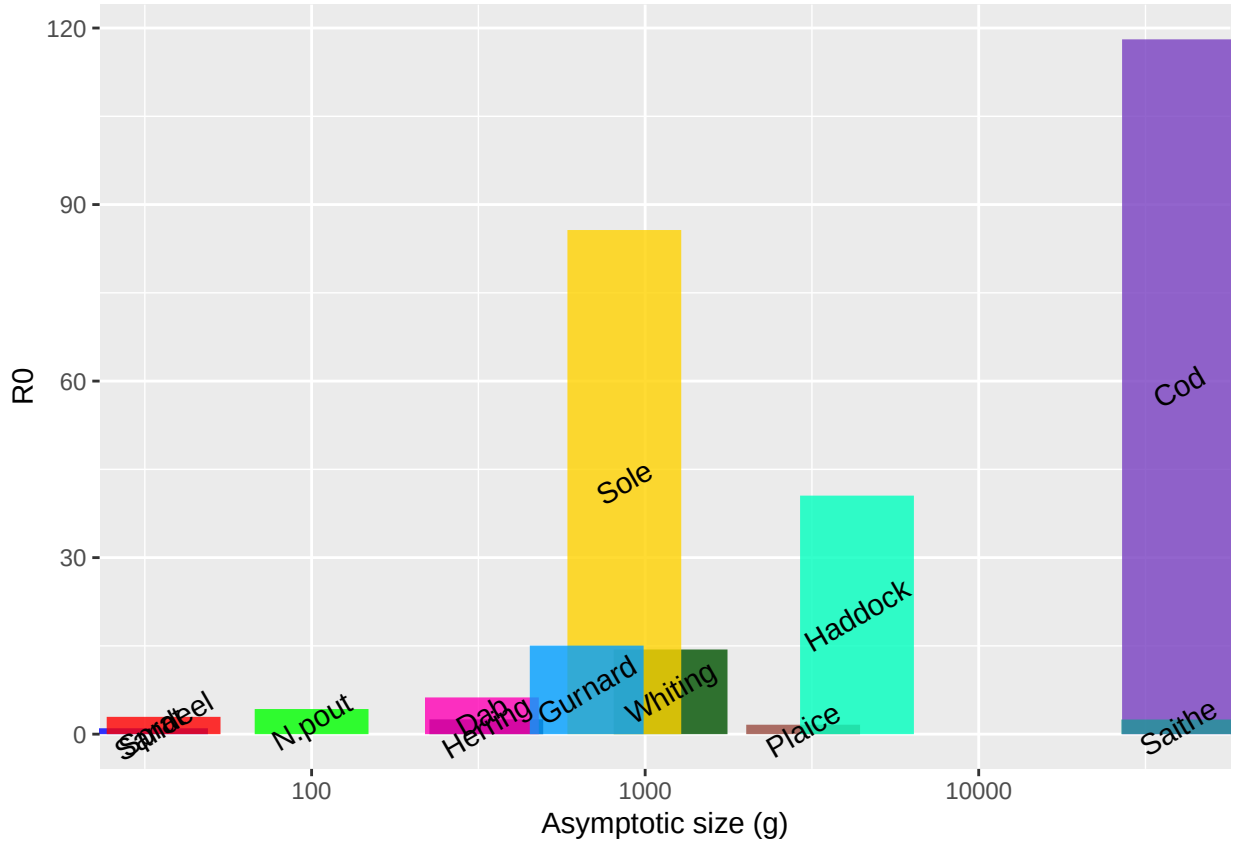
In this section you will:

- check if the RDI/RDD ratio and infer consequences on the ecosystem

```
# plot_dat <- as.data.frame(getRDI(sim_optim@params)/getRDD(sim_optim@params))
# plot_dat$species <- factor(rownames(plot_dat),sim_optim@params@species_params$species)
# colnames(plot_dat)[1] <- "ratio"
#
# ggplot(plot_dat) +
#   geom_bar(aes(x = species, y = ratio, fill = species),stat="identity") +
#   scale_fill_manual(name = "Species", values = sim_optim@params@linecolour) +
#   scale_y_continuous(name = "RDI/RDD") +
#   scale_x_discrete(name = "Species") +
#    theme(panel.background = element_rect(fill = "white", color = "black"),
#          legend.position = "none")
#
# getRDI(sim_optim@params)/getRDD(sim_optim@params)
#
# plot_dat$w_inf <- sim_optim@params@species_params$w_inf
#
# ggplot(plot_dat)+
#   geom_point(aes(x = w_inf, y = ratio, color = species), size = 6, alpha = .8) +
```

```
#   geom_hline(yintercept = 1, linetype = "dashed")+
#   scale_y_continuous(name = "RDI/RDD", limits = c(1,NA)) +
#   scale_x_continuous(name = "Asymptotic size (g)", trans = "log10") +
#   scale_color_manual(name = "Species", values = sim_optim@params@linecolour) +
#   theme(panel.background = element_rect(fill = "white", color = "black"),
#         legend.justification=c(1,1),legend.key = element_rect(fill = "white"))



  plot_dat <- as.data.frame(getRDI(sim_optim@params)/getRDD(sim_optim@params))
  plot_dat$species <- factor(rownames(plot_dat),sim_optim@params@species_params$species)
  colnames(plot_dat)[1] <- "ratio"
  plot_dat$w_inf <- as.numeric(sim_optim@params@species_params$w_inf)

  # trying to have bars at their w_inf but on a continuous scale
  plot_dat$label <- plot_dat$species
  plot_dat2 <- plot_dat
  plot_dat2$ratio <- 0
  plot_dat2$label <- NA
  plot_dat <- rbind(plot_dat,plot_dat2)

ggplot(plot_dat) +
    geom_line(aes(x = w_inf, y = ratio, color = species), size = 20, alpha = .8) +
    geom_text(aes(x = w_inf, y = ratio, label = label),position = position_stack(vjust = 0.5), angle = 3
    scale_color_manual(name = "Species", values = sim_optim@params@linecolour) +
    scale_y_continuous(name = "R0") +
    scale_x_continuous(name = "Asymptotic size (g)", trans = "log10") +
    theme(legend.position = "none")
```

```
## Warning: Removed 12 rows containing missing values (geom_text).
```




```
getRDI(sim_optim@params)/getRDD(sim_optim@params)
```

```
##      Sprat    Sandeel     N.pout    Herring        Dab    Whiting       Sole
##   1.000534   2.940145   4.271215   2.510232   6.241042  14.389434  85.672519
##    Gurnard     Plaice    Haddock        Cod     Saithe
##  15.051169   1.604304  40.538347 118.077148   2.486730
```

```
# seems like there is little density dependence

# # if needed change erepro & plug back into model
 # params@species_params$erepro[] <-1e-3
 # params <- setParams(params)
 # sim <- project(params, effort = 1, t_max = 500, dt=0.1)
 # plot(sim)
```

Is the physiological recruitment, RDI, much higher than the realised recruitment, RDD? High RDI/RDD ratio indicates strong density dependence.

Step 5. Verify the model after the above step by comparing the model with data.

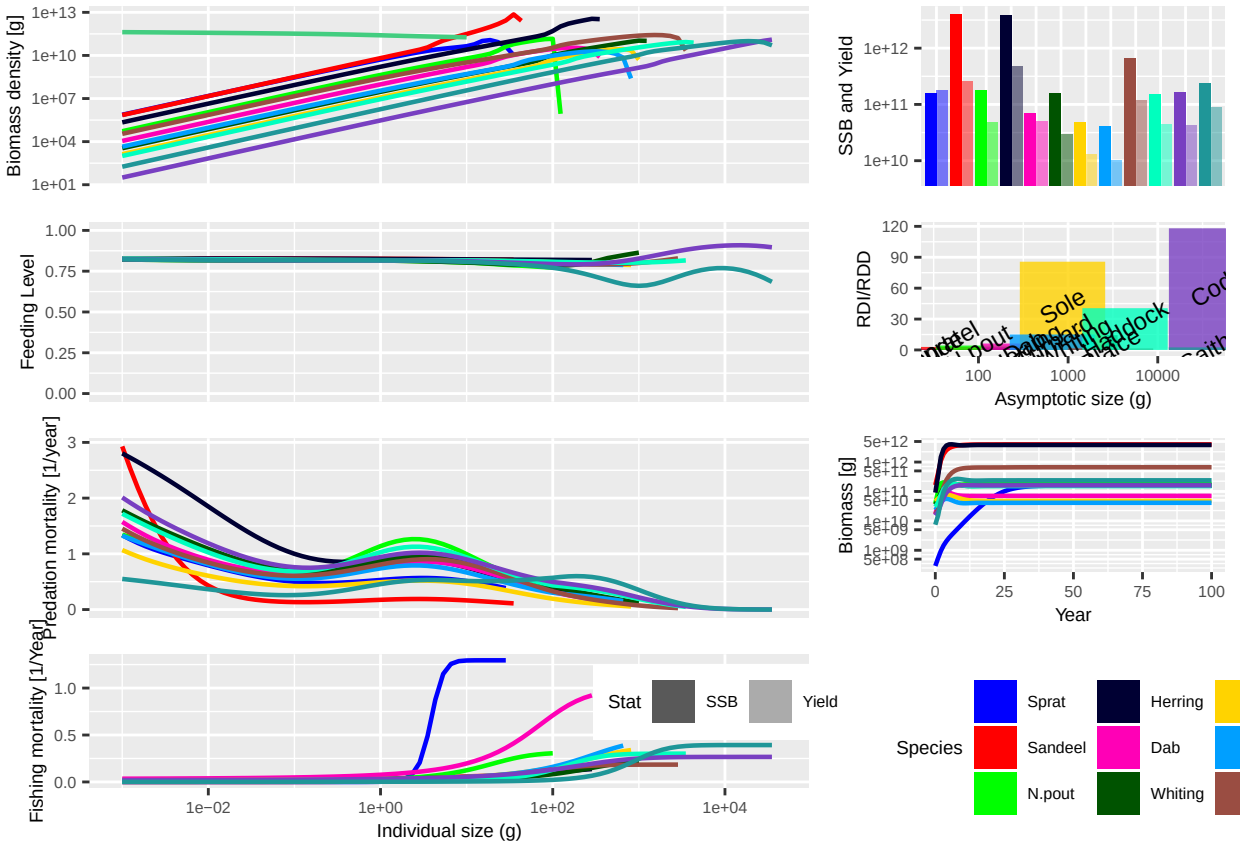
Eg. species biomass or abundance distrubtions, feeding level, naturality mortality, growth, vulnerablity to fishing (fmsy) and catch, diet composition... Many handy functions for plotting these are available here: https://sizespectrum.org/mizer/reference/index.html

```
# hopefully all of this will go on sizespectrum/mizer, in the meantime

plotPredObsYield <-function(sim,dat){
## check obs vs. predicted yield
#sim<-newsim
pred_yield <-melt(getYield(sim)[100,]/1e6)
pred_yield$obs <- dat
pred_yield$species <-row.names(pred_yield)

p <- ggplot() + # plot predicted and observed yields
        geom_point(data = pred_yield,
            aes(x = log10(value), y = log10(obs), color = species)) +
   # plot optimal fit line
        geom_abline(color = "black", slope = 1, intercept = 0) +
  xlab("log10 Predicted Yield") +
  ylab("log10 Observed Yield") #+
 # scale_fill_manual(values = wes_palette(12, "Zissou"))
return(p)
}

plotDiet2 <- function (object, species, xlim = c(1,NA))
{
    params <- validParams(object)
    if (is.integer(species)) {
        species <- params@species_params$species[species]
    }
    diet <- getDiet(params)[params@species_params$species ==
        species, , ]
    prey <- dimnames(diet)$prey
    prey <- factor(prey, levels = rev(prey))
    plot_dat <- data.frame(Proportion = c(diet), w = params@w,
        Prey = rep(prey, each = length(params@w)))
    plot_dat <- plot_dat[plot_dat$Proportion > 0, ]
    ggplot(plot_dat) + geom_area(aes(x = w, y = Proportion, fill = Prey)) +
        scale_x_log10(limits = xlim) + labs(x = "Size [g]") + scale_fill_manual(values = params@linecol
}
```

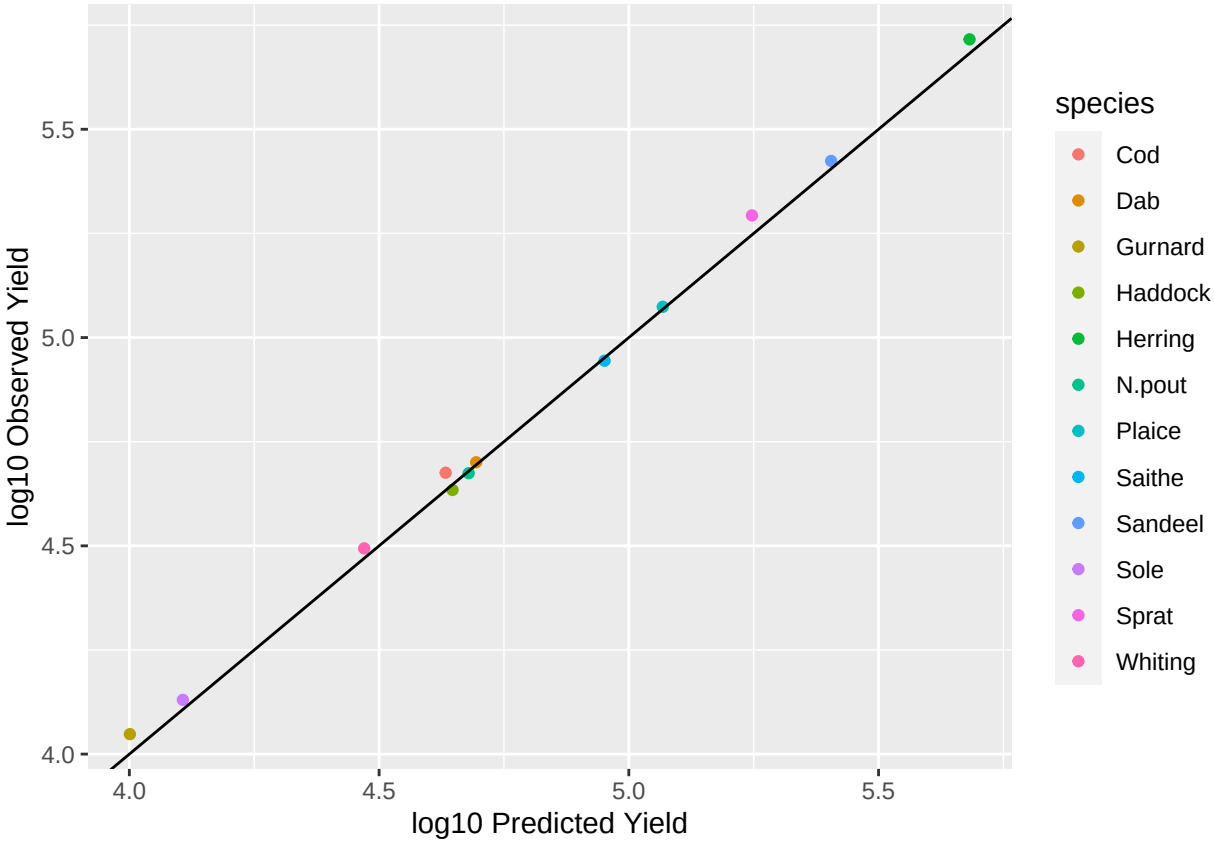
```
plotSummary(sim_optim)
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.

## Warning: Removed 24 row(s) containing missing values (geom_path).

## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.
## Scale for 'x' is already present. Adding another scale for 'x', which will
## replace the existing scale.

## Warning: Removed 24 row(s) containing missing values (geom_path).
```

```
## Warning: Removed 12 rows containing missing values (geom_text).
```

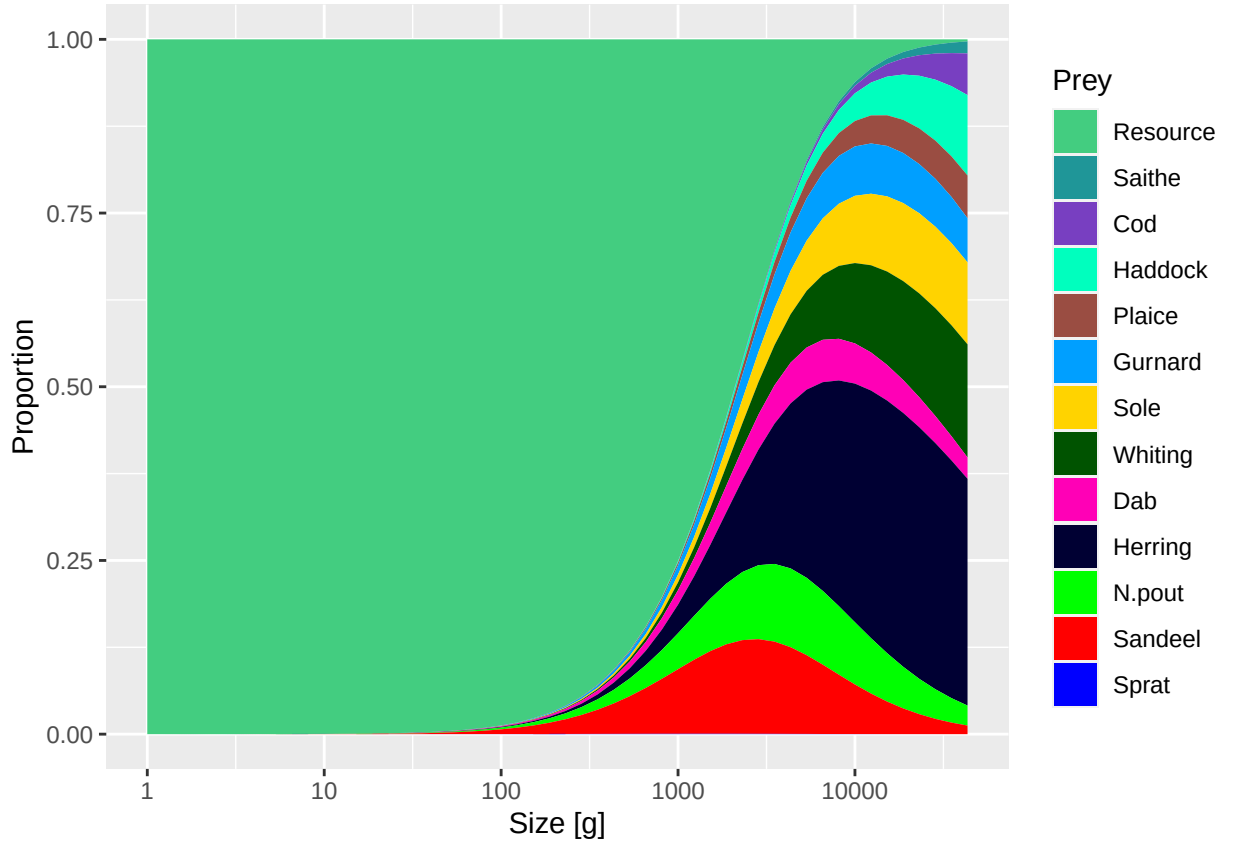



```
plotPredObsYield(sim_optim,catchAvg$Catch_1419_tonnes)
```




```
plotDiet2(sim_optim@params,"Cod")
```

```
## Warning: Removed 421 rows containing missing values (position_stack).
```




```
# interactive plots / won't be displayed in pdf
plotlyBiomass(sim_optim)
```

```
plotlySpectra(sim_optim)
```

```
plotlySpectra(sim_optim,power=2,total = T)
```

```
plotlyGrowthCurves(sim_optim,percentage = T)
```

```
plotlyFeedingLevel(sim_optim)
```

```
plotlyPredMort(sim_optim)
```

```
plotlyFMort(sim_optim)
```

```
# What would happen if we also parameterised the interaction matrix or beta and sigma?
```

Step 6. The final verification step is to force the model with time-varying fishing mortality to assess whether changes in time series in biomassess and catches capture observed trends. The model will not capture all of the fluctuations from environmental processes (unless some of these are included), but should match the magnitude and general trend in the data. We explore this in Example # 2 - Changes through time.