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 to calibrate a size spectrum model 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), growth, vulnerability 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 capture those density dependent effects that are not explicitly modelled. Here, we use a Beverton-Holt type stock recruitment relationship to represent these effects.

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. Calibrate the maximum recruitment, R_{max} by hand by penalising the reproductive capacity of the largest species. Check that the physiological recruitment, RDI, is much higher than the realised recruitment, RDD.
3. Calibrate the maximum recruitment, R_{max} with optimisation, by minimising the error between observed and estimated catches.
4. Calibrate growth by changing the carrying capacity of the background resource spectrum (κ)
5. Calibrate recruitment via $eRepro$

6.Calibrate the food competition

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)

```r
# 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)
library(tictoc)
library(shiny)
library(shinyWidgets)
library(parallel)
library(optimParallel)


# 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
nsParams <- nsParams[order(nsParams$w_inf),] # ordering by asymptotic size for color gradient

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

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

```
## Note: Dimnames of interaction matrix do not match the order of species names in the species data.fram

## 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.
```

```r
params_uncalibrated@species_params$erepro <- .01 # default is 1

# 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_func","gear","interaction_resource","pred_kernel_typ
```

```
##         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
## Dab      5     Dab  324.0    21    191   1.9   Inf 0.536 11.5 17.0 0.010 2.986
## Herring  4 Herring  334.0    99 280540   3.2   Inf 0.606 10.1 20.8 0.002 3.429
## Gurnard  8 Gurnard  668.0    39    283   1.8   Inf 0.266 19.8 29.0 0.004 3.198
## Sole     7    Sole  866.0    78    381   1.9   Inf 0.284 16.4 25.8 0.008 3.019
## Whiting  6 Whiting  1192.0    75     22   1.5   Inf 0.323 19.8 29.0 0.006 3.080
## 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
## Saithe  12  Saithe 39658.6  1076     40   1.1   Inf 0.175 35.3 43.6 0.007 3.075
## Cod     11     Cod 39851.3  1606     66   1.3   Inf 0.216 13.2 22.9 0.005 3.173
##         catchability        h k       ks         z0        gamma     w_mat25
## Sprat     1.29533333 14.46675 0 1.593753 0.18705957 5.652974e-11   11.647460
## Sandeel   0.06510547 25.62741 0 2.936414 0.18171206 3.790575e-11    3.583834
## N.pout    0.31380000 31.20422 0 3.372902 0.12926608 9.750228e-11   20.607045
## Dab       0.97800000 34.87720 0 3.781368 0.08735805 7.579184e-11   18.815128
## Herring   0.18150000 28.36363 0 2.920263 0.08647736 2.514308e-11   88.699888
## Gurnard   0.46250569 20.64990 0 2.193126 0.06863713 4.638552e-11   34.942380
## Sole      0.37383333 24.73805 0 2.567302 0.06294752 5.184323e-11   69.884760
## Whiting   0.24266667 31.77220 0 3.301616 0.05658819 9.927702e-11   67.196884
## Plaice    0.18483333 16.94072 0 1.740765 0.04171321 4.489177e-11   94.075638
## Haddock   0.30150000 41.46028 0 4.196598 0.03685027 7.707429e-11  147.833146
## Saithe    0.39300000 59.95343 0 5.700788 0.01759431 2.435635e-10  964.051303
## Cod       0.26666749 69.40226 0 6.511735 0.01756590 2.325827e-10 1438.909287
##         erepro
## Sprat     0.01
## Sandeel   0.01
## N.pout    0.01
## Dab       0.01
## Herring   0.01
## Gurnard   0.01
## Sole      0.01
## Whiting   0.01
## Plaice    0.01
## Haddock   0.01
## Saithe    0.01
## Cod       0.01
```

w_inf: asymptotic size

w_mat: maturation size (determines when 50% of the population has matured / not sure!)

beta: preferred predator/prey mass ratio

sigma: width of the feeding kernel

R_max: Beverton-Holt density dependence parameter

k_vb: von Bertalanffy growth parameter

l25: length at …

l50: length at …

a: coefficient for age to size conversion

b: constant for age to size conversion

catchability: fisheries efficiency

h: maximum intake rate

k: metabolism constant

ks: metabolism coefficient

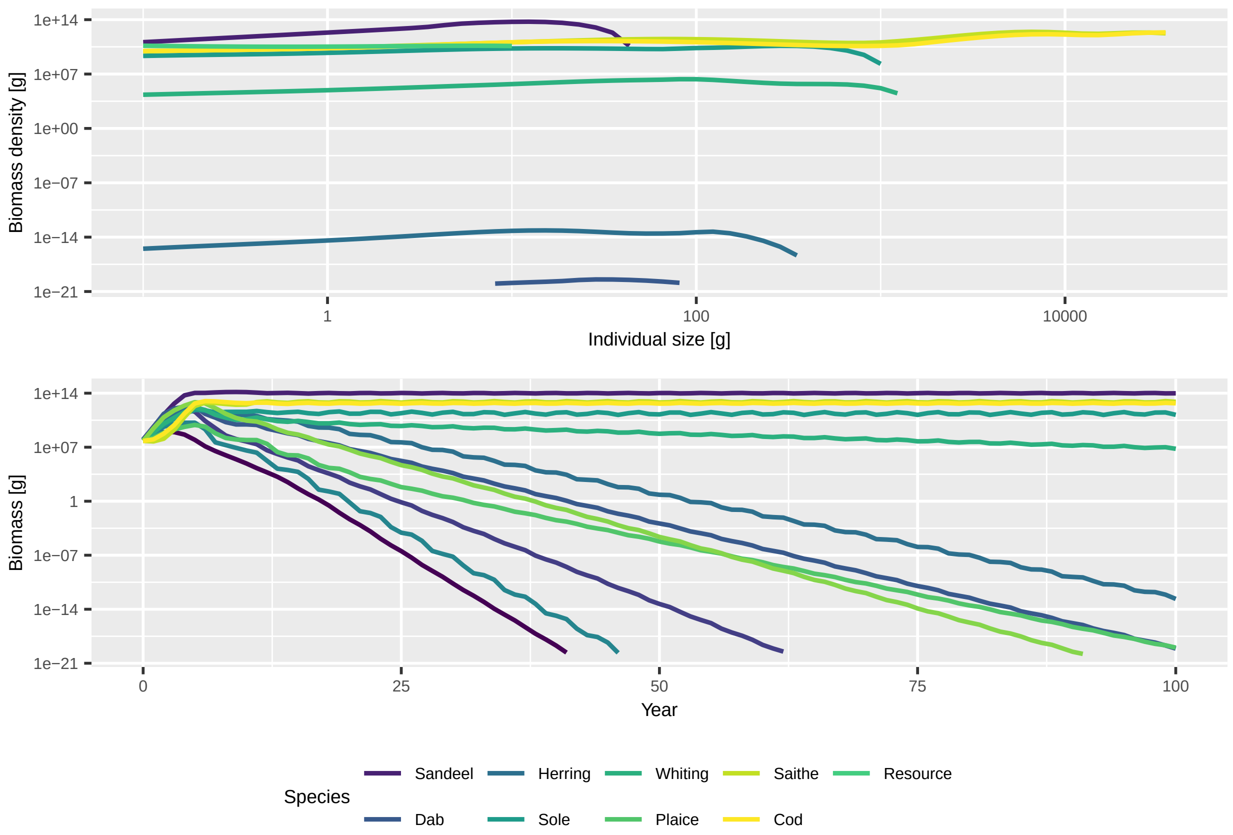
z0: background mortality coefficient

gamma: search volume (obtained from beta and sigma)

w_mat25: weight at which 25% of individuals are mature

erepro: coefficent that weights reproductive output

```
# lets' change the plotting colours, by far the hardest and trickiest part

# looks good but hard to distinguish some species
library(viridis)
params_uncalibrated@linecolour[1:12] <-viridis(dim(params_uncalibrated@species_params)[1])

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

# Gradient over asymptotic size but might get difficult to distinguish species past 9
# colfunc <- colorRampPalette(c("firebrick3","darkturquoise", "orange"))
# colGrad <- colfunc(dim(params_uncalibrated@species_params)[1])
#  #names(colGrad) <- params_uncalibrated@species_params$species[order(params_uncalibrated@species_para
#
# params_uncalibrated@linecolour[1:12] <- colGrad
#
params_uncalibrated@linecolour["Resource"] <-"seagreen3"

# jet.colors <- colorRampPalette(c("#00007F", "blue", "#007FFF", "cyan", "#7FFF7F", "yellow", "#FF7F00"
# params_uncalibrated@linecolour[1:12] <-jet.colors(dim(params_uncalibrated@species_params)[1])

# write color routine for color gradient readable but related to winf (can use my usual one but limited
# maybe have different color depening on species number

# line width related to asymptotic size ?


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

plotSummary(sim_uncalibrated, short = T)
```



The top panel shows the different species size spectrum at the last time step of the simulation while the bottom panel shows the abundance per species through time. These plots show that species do not coexist and several go extinct. This is because there was no external density dependence (R_{max} is set at Inf) and the largest species (Cod and Saithe) are out-competing 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:

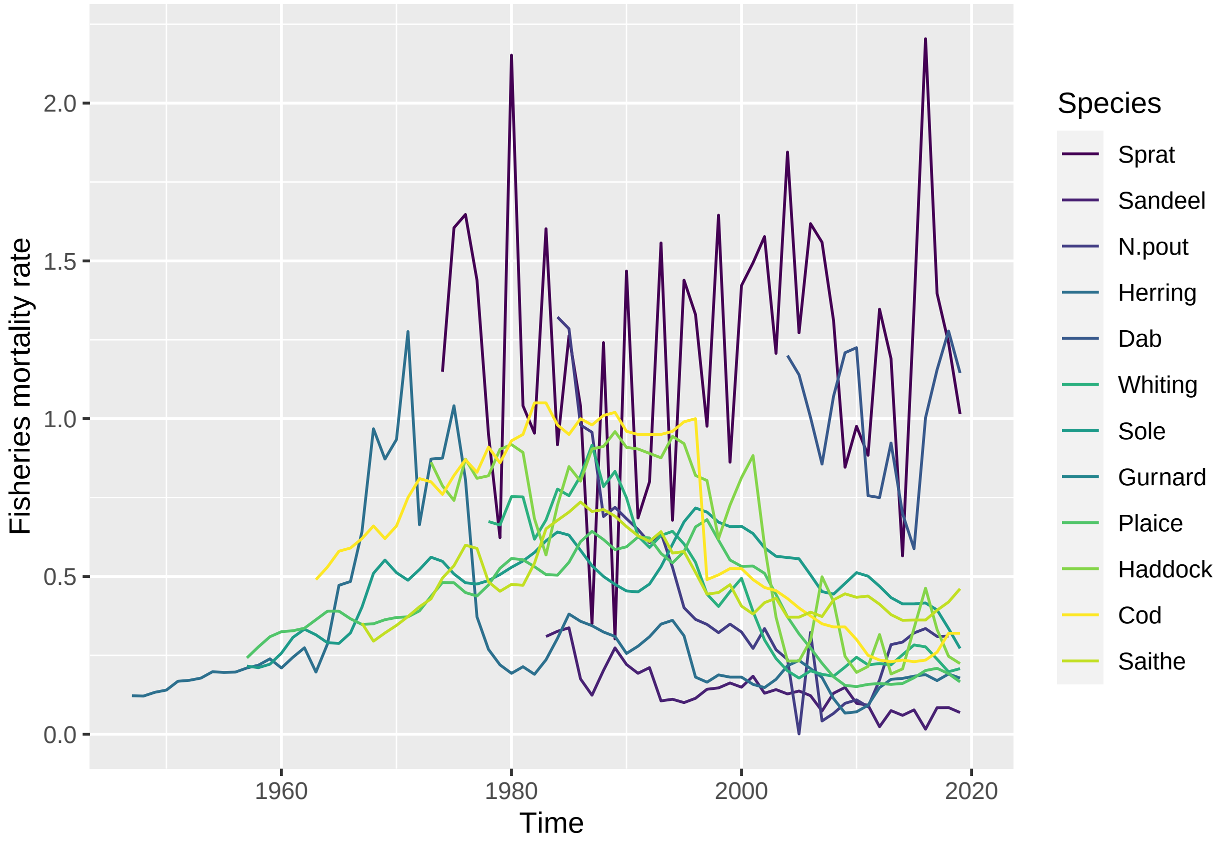
- Download fisheries data and process them in a format comparable to the model output
- Visualise the data

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

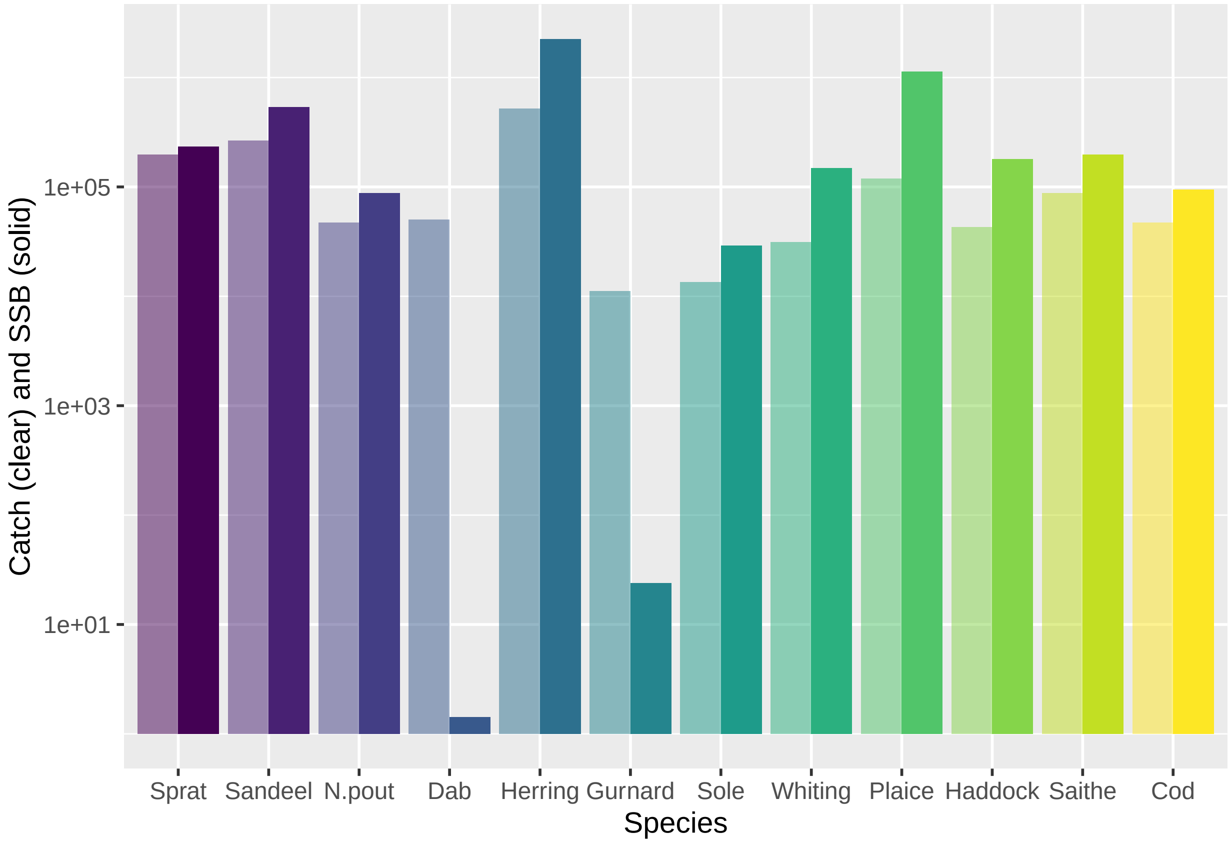
```
# fisheries mortality F
fMat <- read.csv("data/fmat.csv")
fMatWeighted <- read.csv("data/fmatWeighted.csv") # Sandeel and Cod have multiple data base so average

# 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")
```

```
plot_dat <- reshape2::melt(fMatWeighted,"X")
colnames(plot_dat) <- c("Time", "Species", "F")

ggplot(plot_dat)+
  geom_line(aes(x = Time, y = F, color = Species))+
  scale_y_continuous(name = "Fisheries mortality rate") +
  scale_color_manual(values = params_uncalibrated@linecolour)
```



```
plot_dat <- data.frame(catchAvg,ssbAvg)
plot_dat$species.1 <- NULL
colnames(plot_dat) <- c("Species", "average catch", "average SSB")
plot_dat$Species <- factor(as.character(plot_dat$Species),levels = c(as.character(params_uncalibrated@sp
plot_dat <- reshape2::melt(plot_dat,"Species")


  ggplot(plot_dat) +
    geom_bar(aes(x = Species,y = value, fill = Species, alpha = variable), stat = "identity", position =
    # coord_cartesian(ylim = c(0.5*min(plot_dat$value),NA)) +

    scale_y_continuous(trans = "log10", name = "Catch (clear) and SSB (solid)") + #, limits = c(0.5*min
    scale_fill_manual(name = "Species", values = params_uncalibrated@linecolour) +
    scale_alpha_manual(name = "Stat", values = c(0.5, 1), labels = c("Catch", "SSB")) +
    theme(
        legend.position = "none", legend.key = element_rect(fill = "white"))
```



The first plot shows the fishing time-series per species while the second plot shows the average ssb and catch for the period 2014 - 2019.

RF: Dab and Gurnard have some issues, it comes from the data though, the calculations are correct

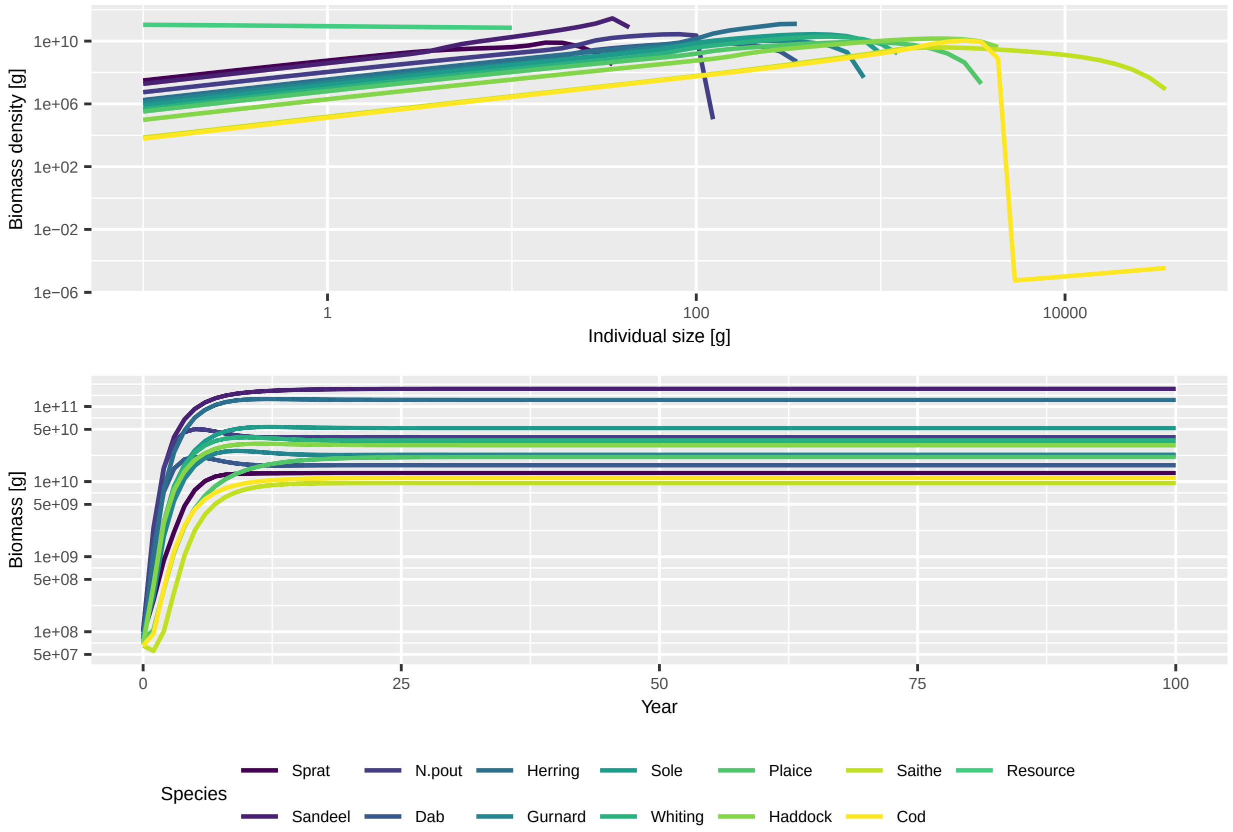
Step 2. Guessing the maximum recruitment

In this section you will:

- guess reasonable R_{max} values which will stop out-competition from a few species
- take a look how it affects the density dependence of each species and their predicted yield

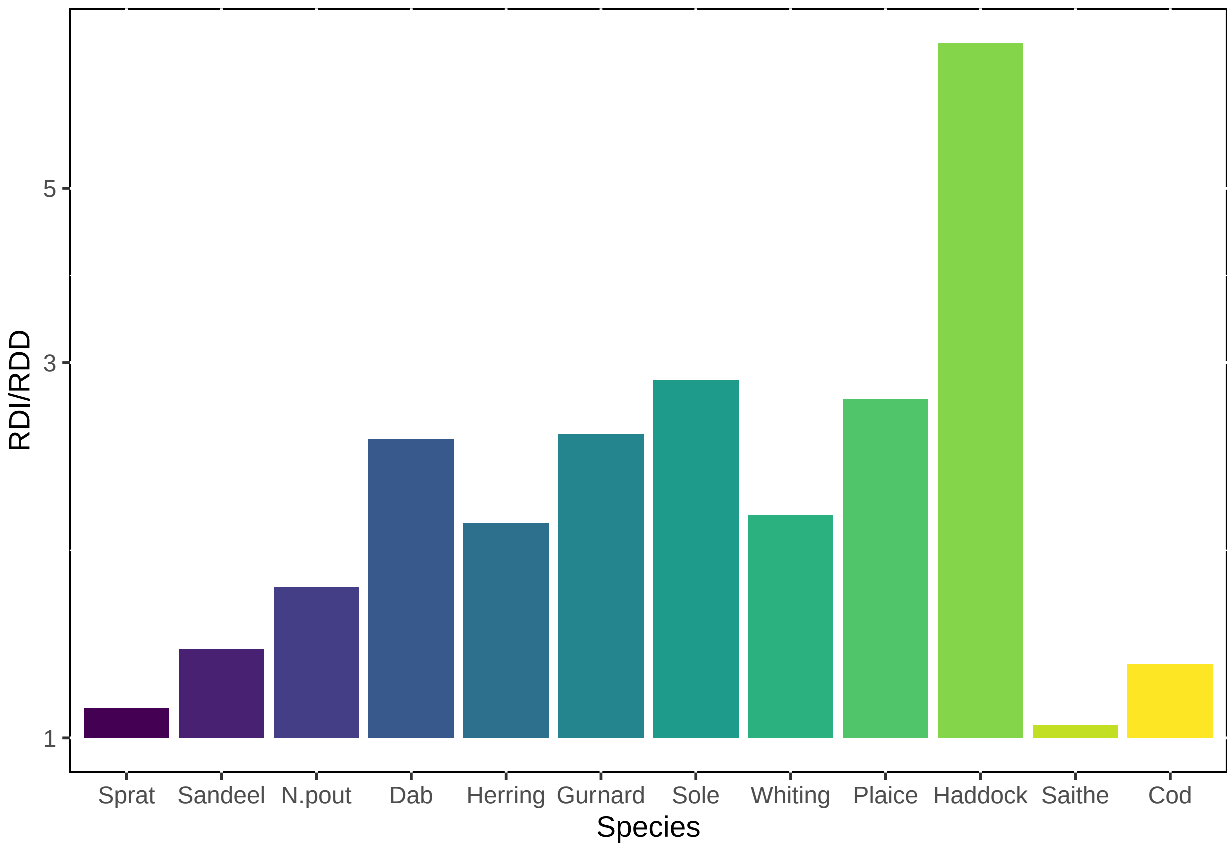
```
# let's start again and replace with the initial pre-calibration "guessed" Rmax
params_guessed <- params_uncalibrated
# penalise the large species with higher density dependence
params_guessed@species_params$R_max <- params_guessed@resource_params$kappa*params_guessed@species_param
# and reduce erepro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

params_guessed <- setParams(params_guessed)
# run with fishing
```

```r
sim_guessed <- project(params_guessed, t_max = 100, effort =1)
plotSummary(sim_guessed, short = T)
```



The ecosystem looks way better. Saithe's largest individuals are having a hard time, but at least species coexist.

```r
  plot_dat <- as.data.frame(getRDI(sim_guessed@params)/getRDD(sim_guessed@params))
  plot_dat$species <- factor(rownames(plot_dat),sim_guessed@params@species_params$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_guessed@params@linecolour) +
  scale_y_continuous(name = "RDI/RDD", trans = "log10") +
  scale_x_discrete(name = "Species") +
   theme(panel.background = element_rect(fill = "white", color = "black"),
        legend.position = "none")
```

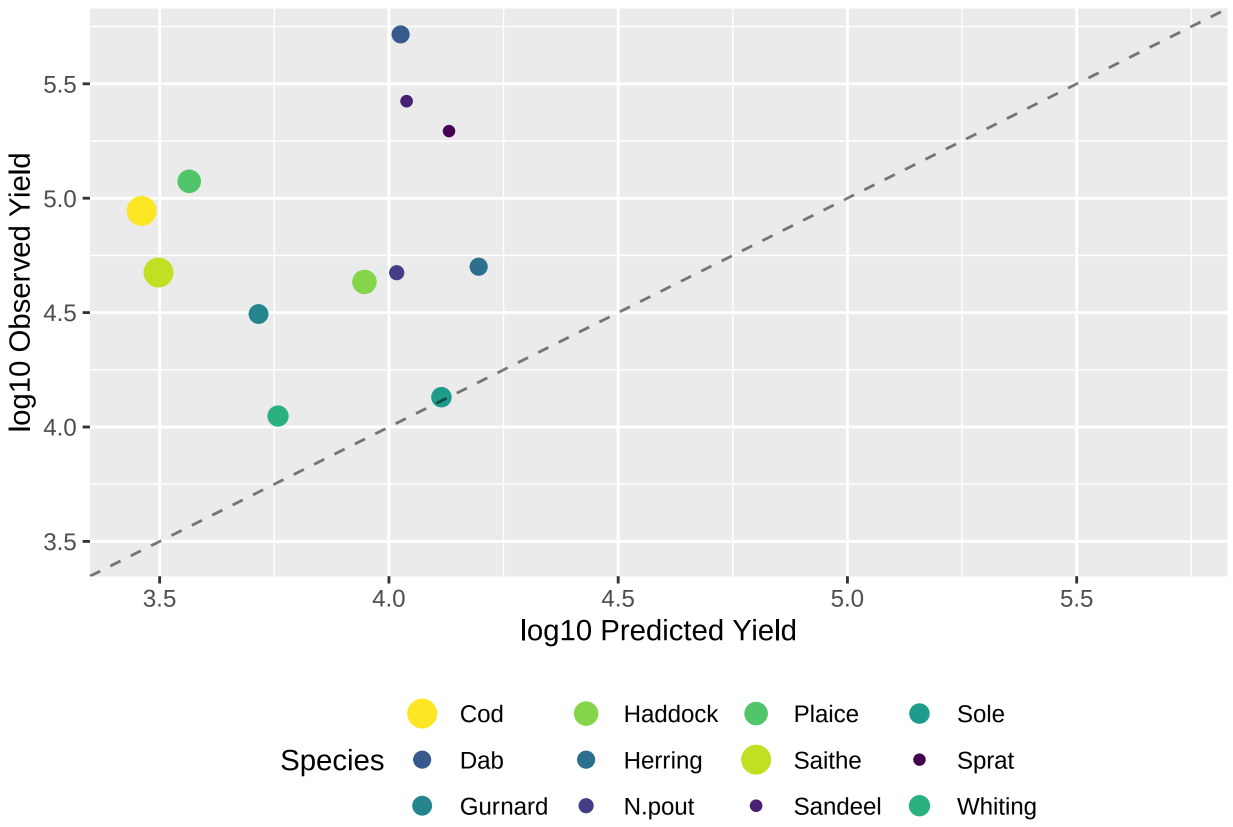


```
getRDI(sim_guessed@params)/getRDD(sim_guessed@params)
```

```
##    Sprat  Sandeel   N.pout      Dab  Herring  Gurnard     Sole  Whiting
## 1.092633 1.297170 1.553506 2.396844 1.874629 2.430816 2.852927 1.920603
##   Plaice  Haddock   Saithe      Cod
## 2.698874 7.647749 1.039356 1.241662
```

Is the physiological recruitment, RDI, much higher than the realised recruitment, RDD? High RDI/RDD ratio indicates strong density dependence, meaning that 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. The largest species that were the most limited by our new R_{max} do not show a strong density dependence. The medium-sized species are the most affected here.

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



This plot show the ratio between observed yield (from the data loaded earlier) and the yield recreated in our model. Only one species matches at the moment (Sole). The size of the dot on the plot is proportional to the species' asymptotic size for clarity.

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_guessed@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_guessed

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

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

```
## Convergence was achieved in 60 years.
```

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

```
## [1] 73.34365
```

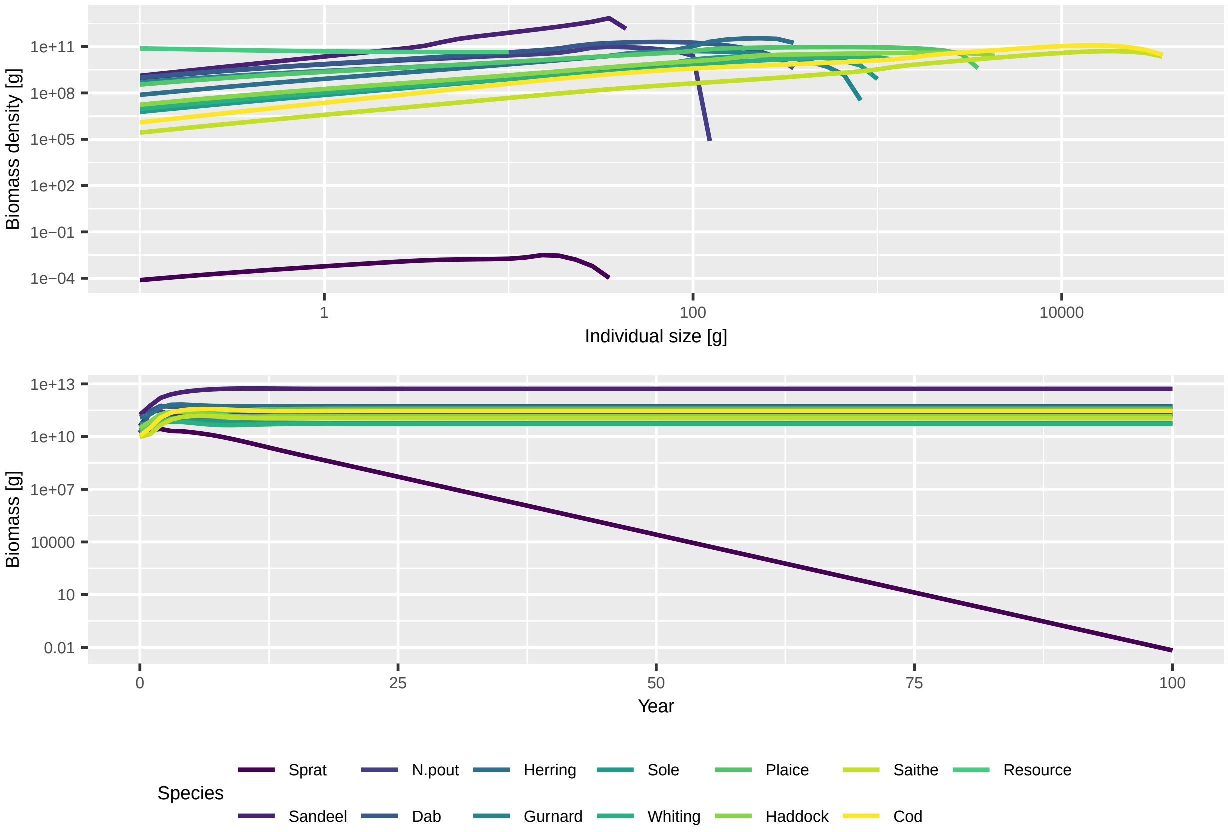
Now, carry out the optimisation. There are several optimisation methods to choose from - we need to select the most robust one to share here. The R package optimParallel seems to be the most robust general R package and has replaced optim. Often this requires repeateing the proceure several times but 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.

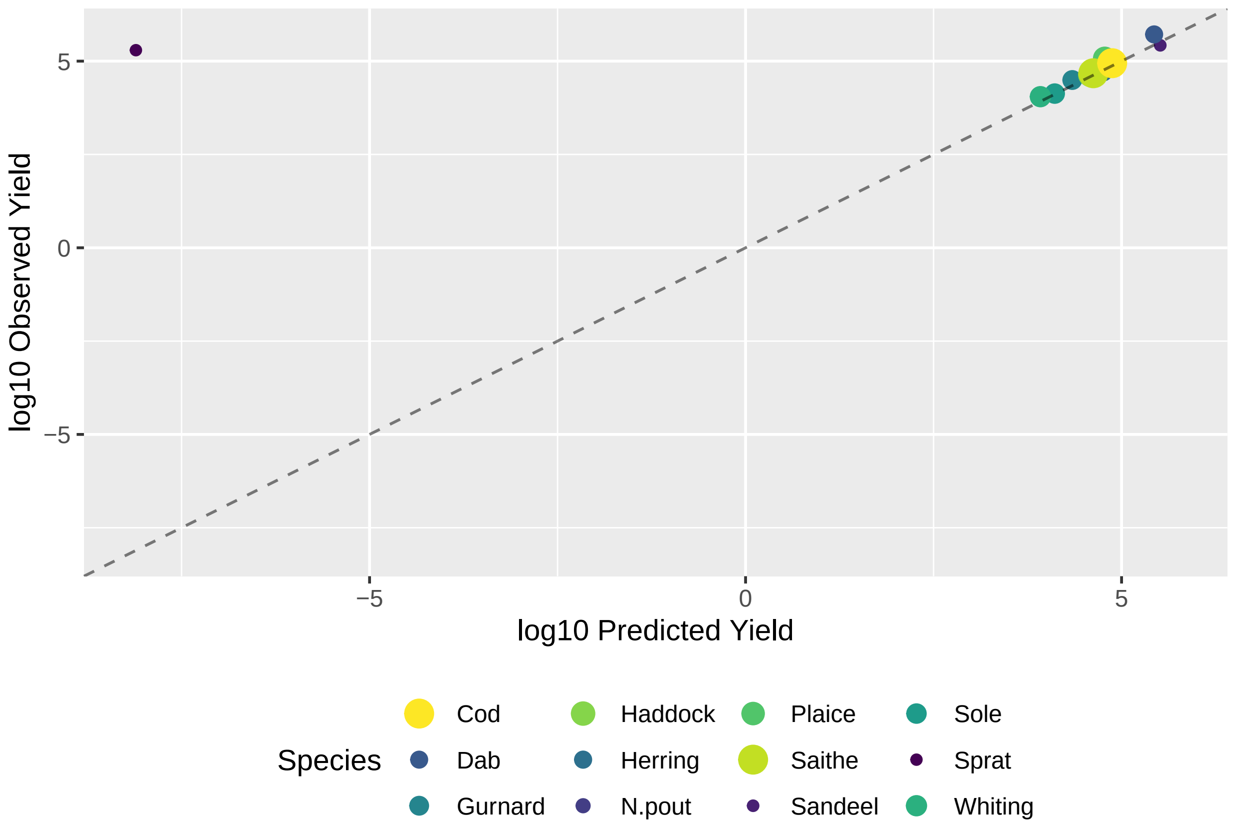
```
# change kappa and erepro based on shiny exploration, set up initial values based on "close to" equilib
# params_steady already set to erepro = 0.001 and kappa = 10^11

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")
```

```
# if previous block was not evaluated
params_optim <- params_guessed
# params_optim@resource_params$kappa<-3.2e11 # old value (keeping it for now cause it works)
```

```
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, short = T)
```



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



Calibrating for best observed/predicted yield makes one species (Sprat) show signs of collapse. We need to look at other parameters to get the community to coexist again.

Step 4. Calibrate the recruitment with eRepro.

In this section you will:

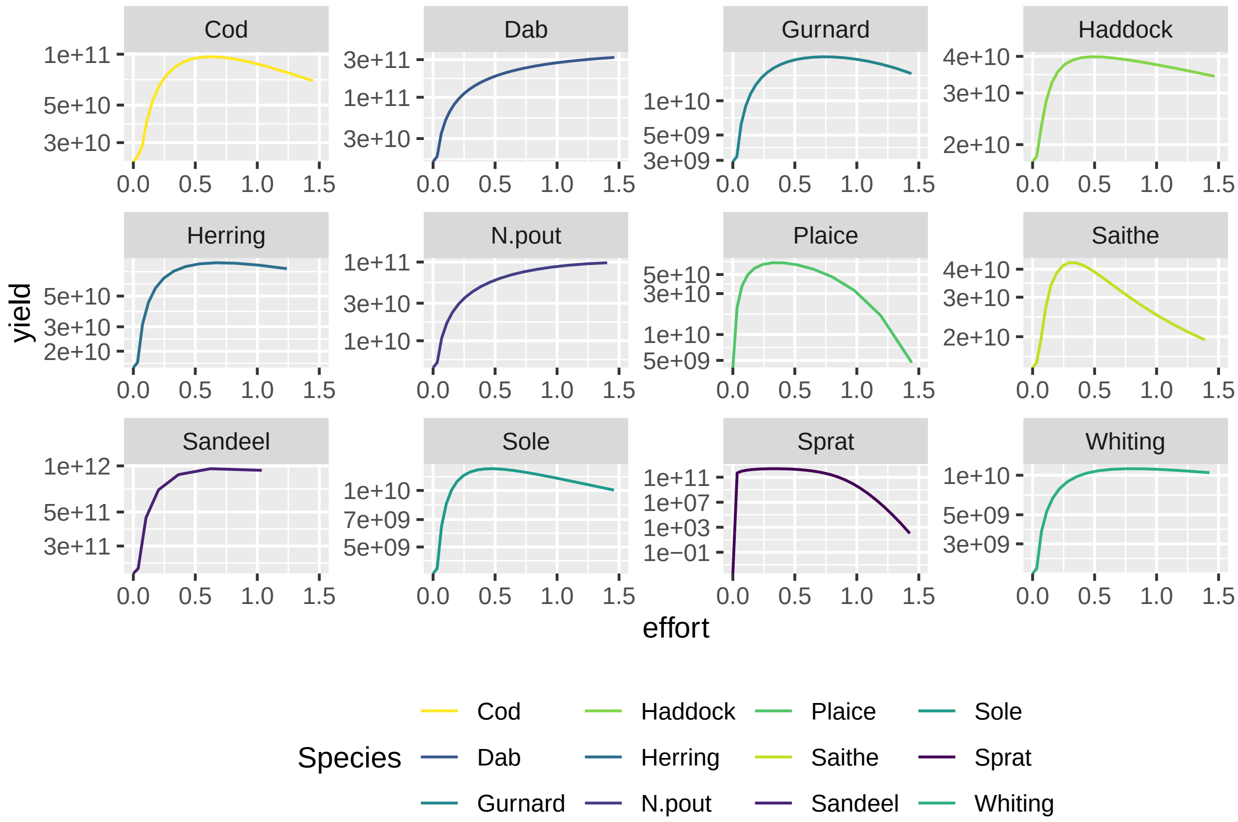
- Look at effect of *erepro* on the reproductive outputs
- Check what impact erepro has on the F_{msy}

eRepro represents the energy conversion efficiency between energy allocated to reproduction and the actual spawn. Lowering *erepro* biologically means higher egg mortality rate or wasteful energy invested into gonads. For example, *eRepro* is currently set to 0.01 meaning, that for every one g of mass allocated to reproduction, $0.1g$ will be used as spawn recruitment, the rest is lost. This coefficient is applied before any density-dependence requirement. Let's use Rshiny to see how varying *eRepro* influences the ecosystem (to play around).

Now let's take a look at the current F_msy per species. To do so we need to set up an ecosystem with one gear per species and vary one gear intensity at a time while the other gear stay constant (default effort value of one).

```
# Can be a bit long to not need to evaluate, just load the next block

plot_dat <- plotFmsy(params_optim, returnData = T, effortRes = 50)

saveRDS(plot_dat, "Fmsy.rds")
```

```
plot_dat <- readRDS("Fmsy.rds")

ggplot(plot_dat) +
geom_line(aes(x = effort , y = yield, color = species))+
  facet_wrap(species~., scales = "free") +
  scale_x_continuous(limits= c(0,1.5))+#, limits = c(1e10,NA))+
  scale_y_continuous(trans = "log10") +
  scale_color_manual(name = "Species", values = params_optim@linecolour) +
    theme(legend.position = "bottom", legend.key = element_rect(fill = "white"))
```

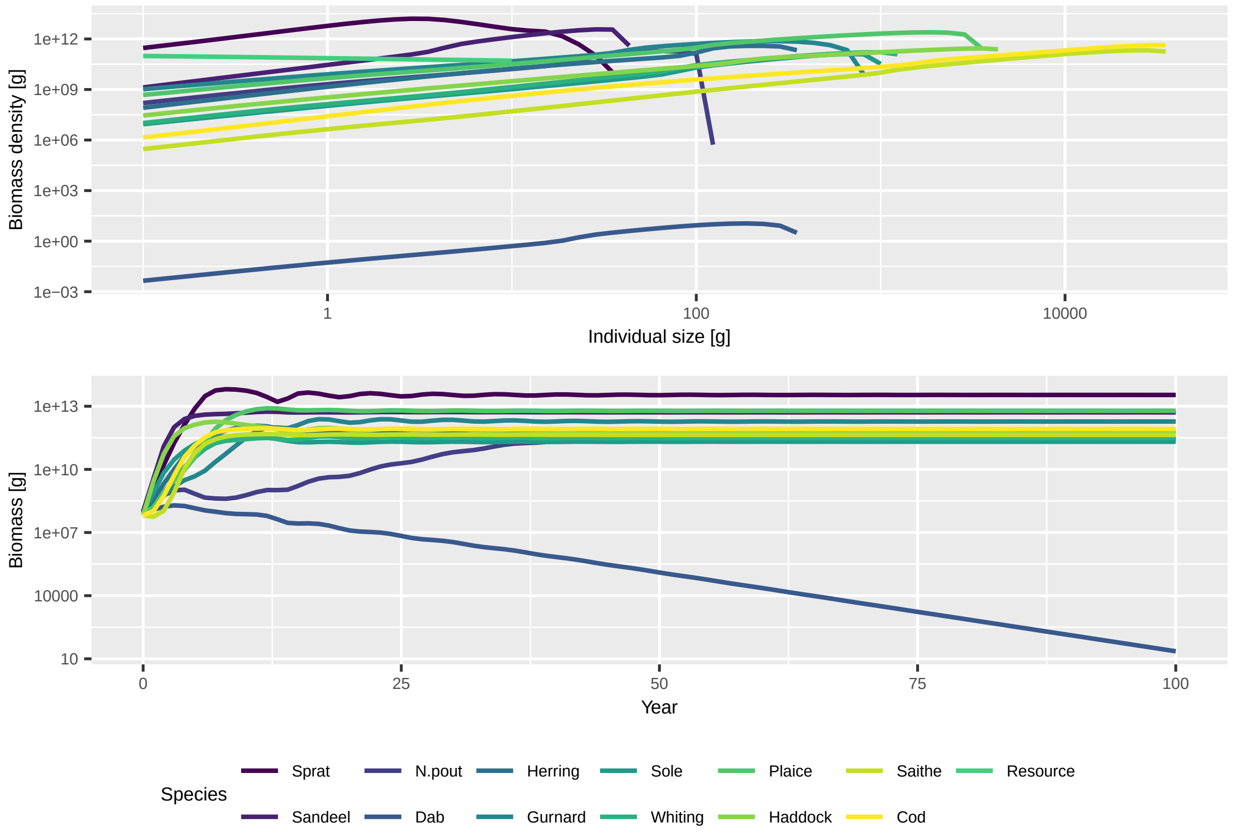


Decreasing *eRepro* is going to move the MSY towards lower effort. Until now we had the same *eRepro* for all species but we can input species-specific values to calibrate our recruitment. To do so, let's use another shiny app again. The next one allows you to change *eRepro* one at a time and see the effect on the species' F_{msy}

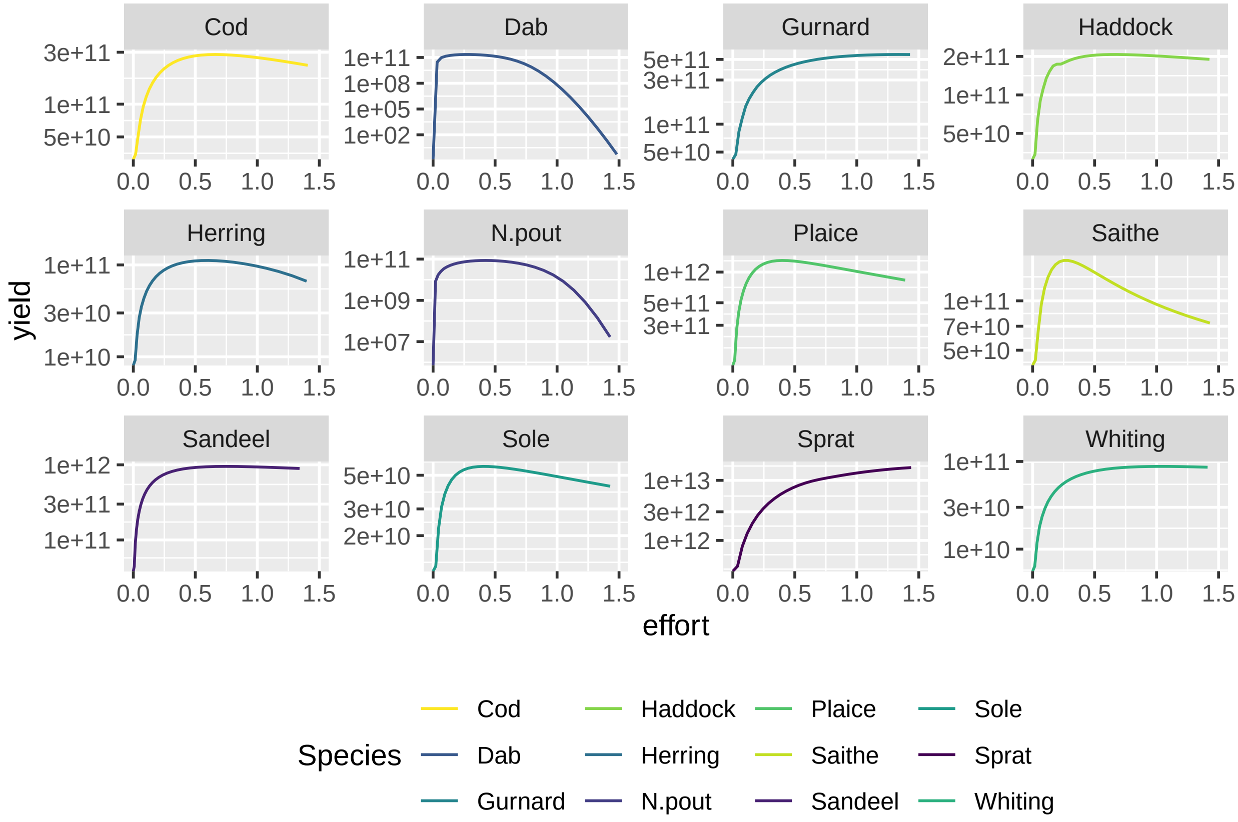
Let's see what our system looks like with species-specific *eRepro*

```
params_optim@species_params$erepro <- 10^(c(-1,-2,-4,-4.5,-3,-3,-2,-3,-2,-2,-2,-2))

params_optim2 <- setParams(params_optim)
sim_optim2 <- project(params_optim2, effort = 1, t_max = 100)
plotSummary(sim_optim2, short = T)
```

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



```r
plotFmsy(params_optim2, effortRes = 30, returnData = F)
```



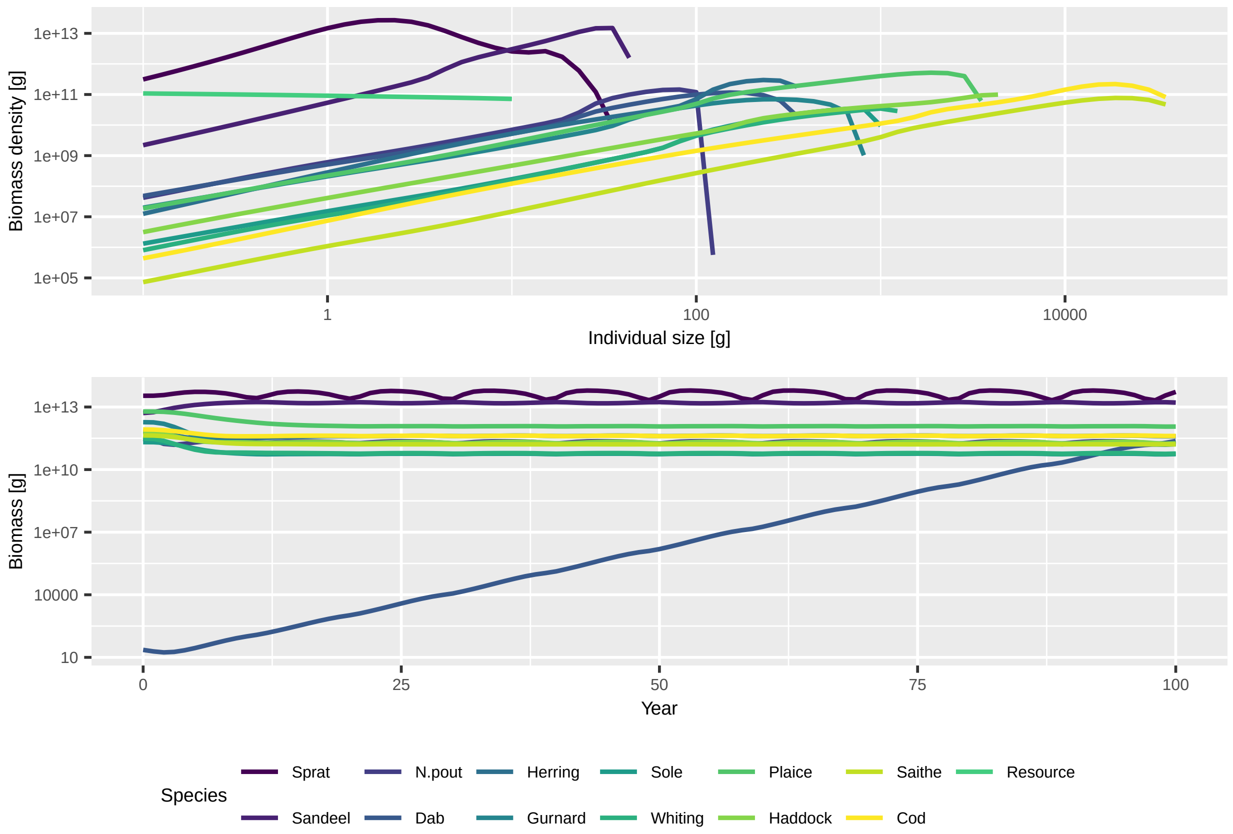
Some trade-offs here, relatively high *eRepro* allows Sprat to coexist with the othe species but he F_{msy} plot looks wrong. The same is true for Dab but I do not want to lower too much *eRepro* either. Instead we are going to run the R_{max} calibration again to see if we improved the yield match.

And now let's use this set of parameters to run the R_{max} calibration again as in step 3.

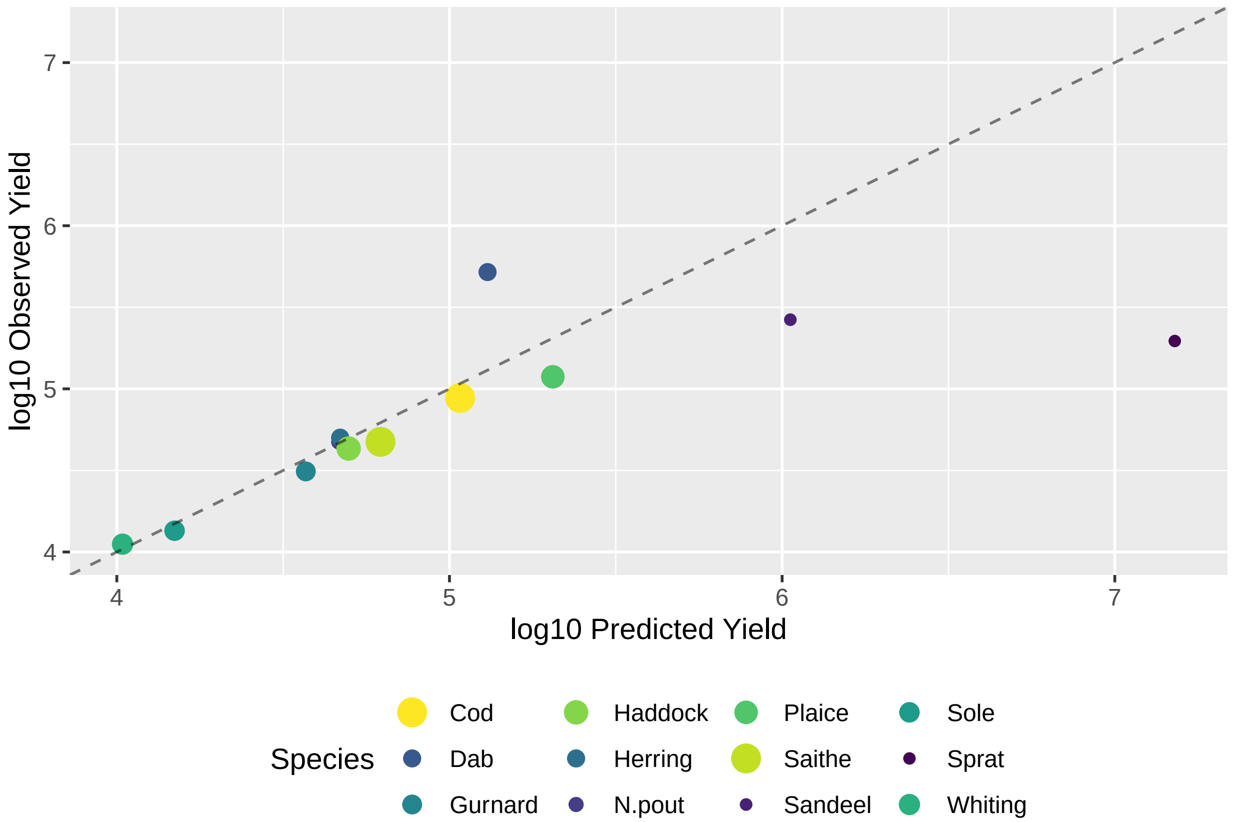
```
# saving the last time step of sim_optim2 in the param object
params_optim2@initial_n <- sim_optim2@n[dim(sim_optim2@n)[1],,]
params_optim2@initial_n_pp <- sim_optim2@n_pp[dim(sim_optim2@n_pp)[1],]

params_calibration <- params_optim2
vary <-  log10(params_calibration@species_params$R_max)
params_calibration<-setParams(params_calibration)

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_calibration, dat = catchAvg$Catch_1419_tonn
                             method   ="L-BFGS-B",lower=c(rep(3,12)),upper= c(rep(15,12)),
                            parallel=list(loginfo=TRUE, forward=TRUE))

stopCluster(cl)
toc() # 80'' using 47 cores
saveRDS(optim_result,"optim_para_result2.RDS")
```

```
optim_result <- readRDS("optim_para_result2.RDS")
# if previous block not evaluated
params_optim2@initial_n <- sim_optim2@n[dim(sim_optim2@n)[1],,]
params_optim2@initial_n_pp <- sim_optim2@n_pp[dim(sim_optim2@n_pp)[1],]

params_calibration <- params_optim2

# optim values:
params_calibration@species_params$R_max <- 10^optim_result$par

# set the param object
params_calibration <-setParams(params_calibration)
sim_optim2 <- project(params_calibration, effort = 1, t_max = 100, dt=0.1, initial_n = params_calibrati
                      initial_n_pp = params_calibration@initial_n_pp)
saveRDS(sim_optim2,"optim_para_sim2.RDS")
plotSummary(sim_optim2, short = T)
```

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



```
plotPredObsYield(sim_optim2,catchAvg$Catch_1419_tonnes)
```



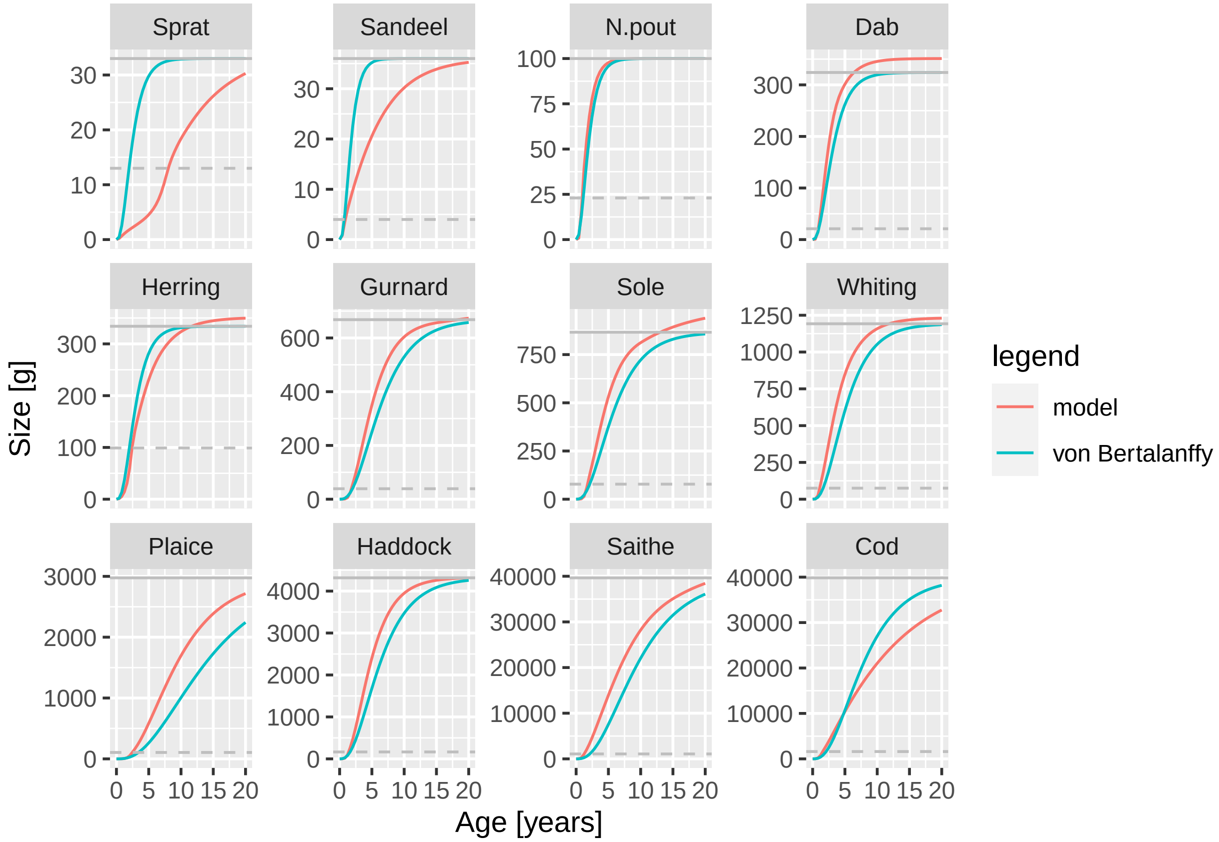
Now the fit is less good overall but we only have 2 outliers, N.pout and Sandeel

Step 5. Calibrating the growth

In this section you will:

- look at the growth curves of each species
- tweak the κ parameter to adjust the growth curves

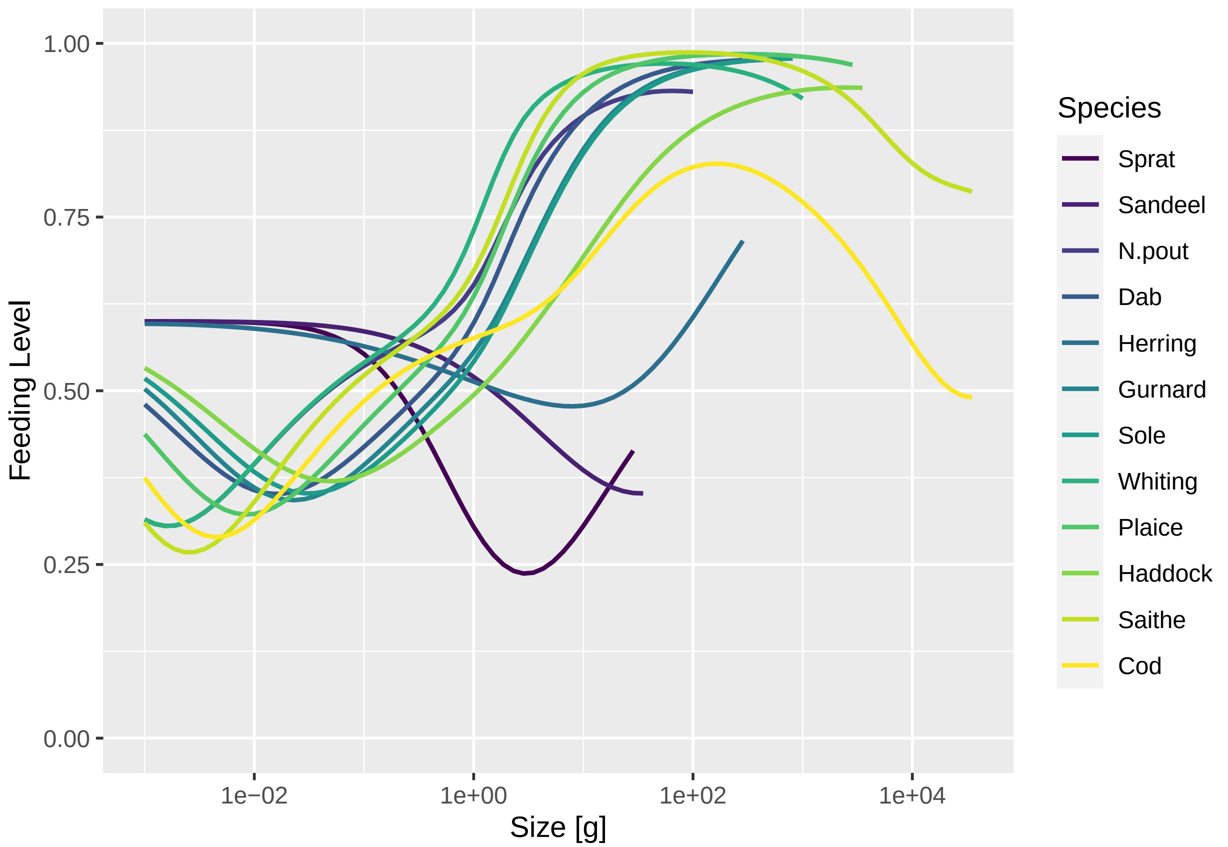
```
plotGrowthCurves(sim_optim2, species_panel = T)
```



```
# Ken's comment: plotGrowthCurves. Legend should be "model", "Observed". Need to discuss about that sin
```

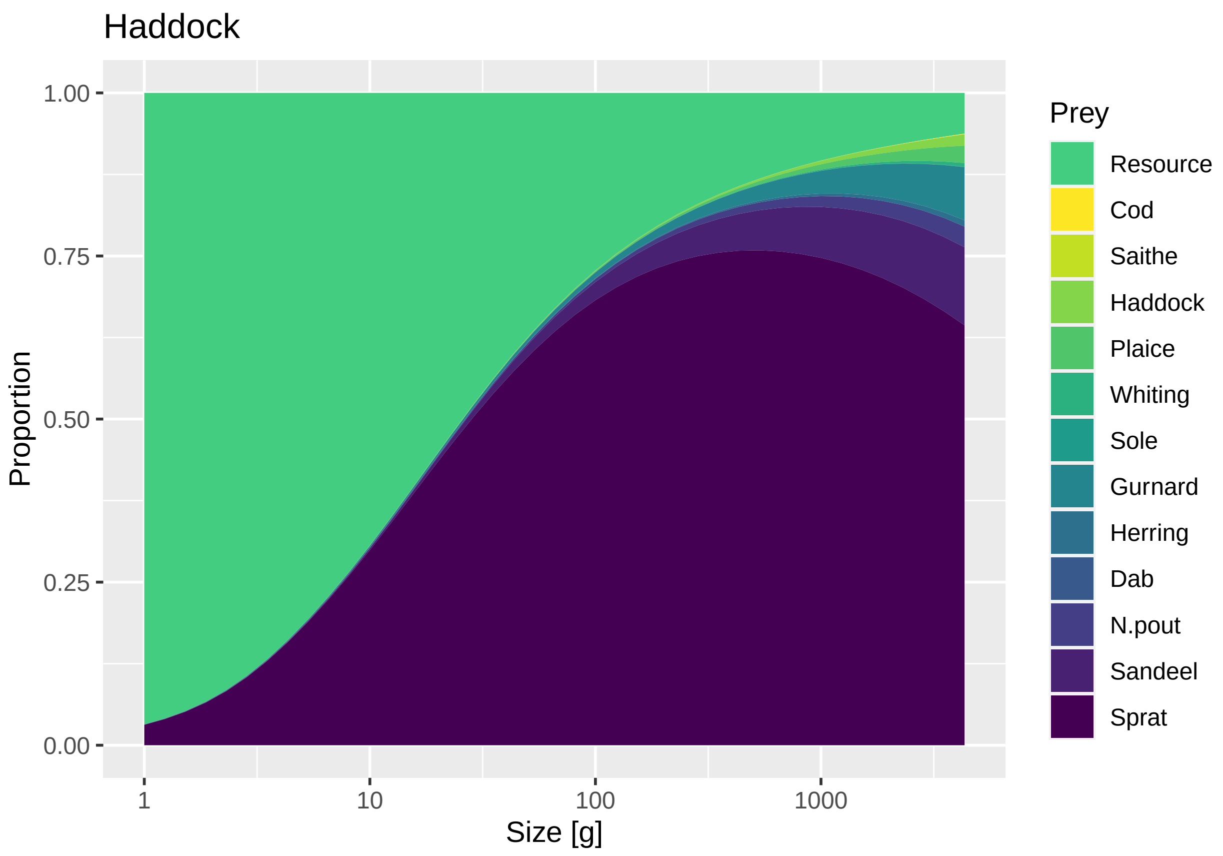
Most species have a growth curve similar to the expected von Bertalanffy growth curve, apart from Spart and Sandeel which have much slower growth than expected. κ, which is the carrying capacity of the background spectrum will affect the food availability and therefore the growth rate (more food means faster growth). However, changing κ will affect all growth curves. If only a few species are off, we need to change the γ parameter (search volume) per species.

```
plotFeedingLevel(sim_optim2, include_critical = F)
```

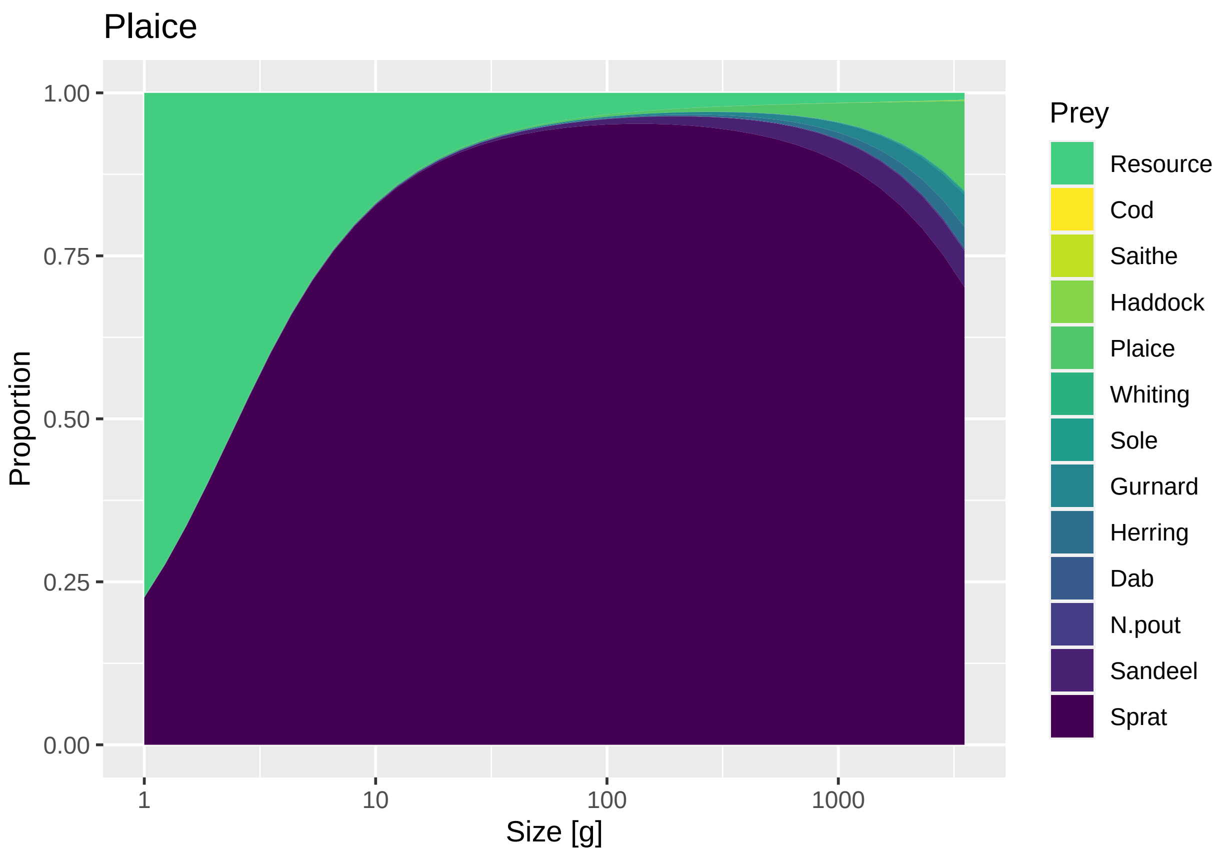


The feeding level here shows that the species reaching the highest size classes have a feeding level close to one, meaning that they feed to satiation. Let's look at their diet to check what makes them to full.

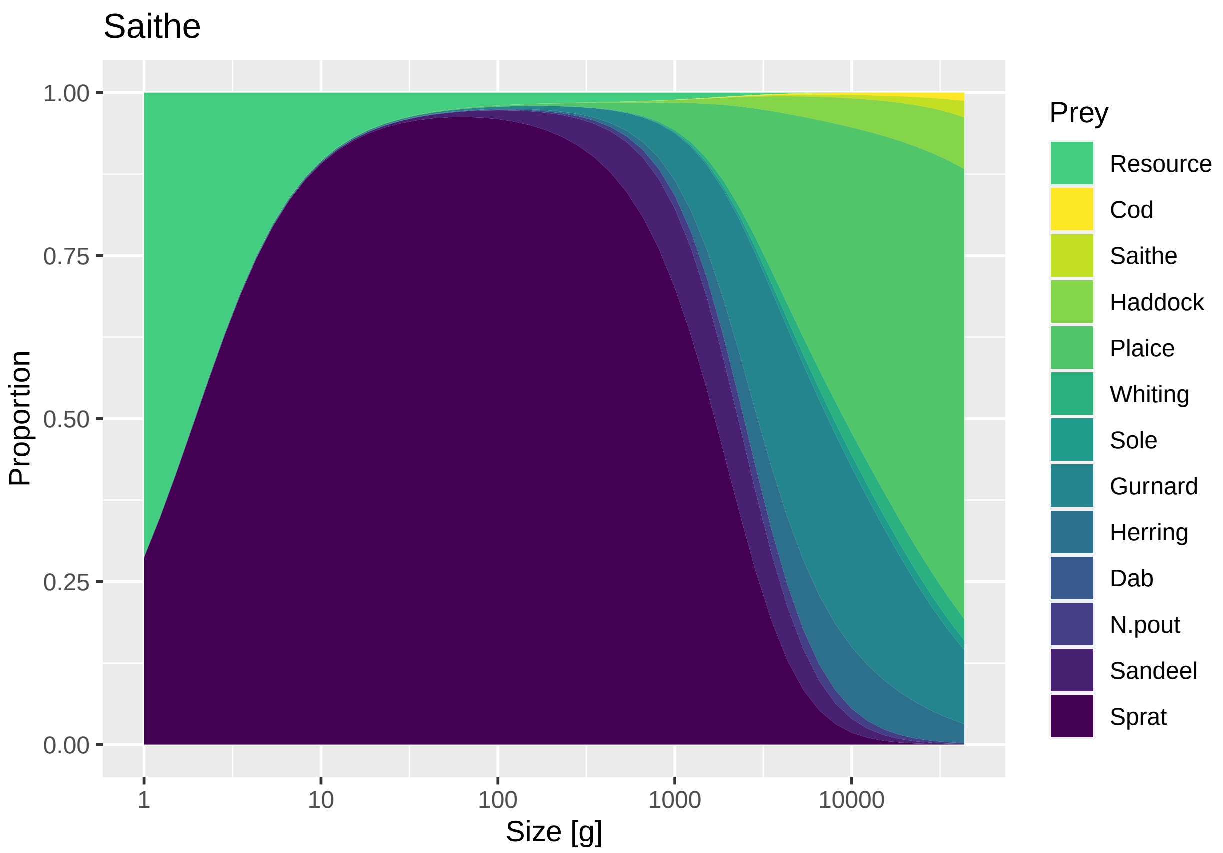
```
plotDiet2(sim_optim2, "Haddock")
```



```
plotDiet2(sim_optim2, "Plaice")
```



```
plotDiet2(sim_optim2, "Saithe")
```



The diets look great as the fish feed on each other and do not rely too much on the background spectrum. We can still reduce the carrying capacity of the background spectrum even more, which could lower the feeding level of the species. Let's do this using the Rshiny app

RF: Rshiny section disabled for pdf

Decreasing slightly κ drives Sprat to extinction probably as it becomes the next choice of food when we reduce the size of the background spectrum (as we saw on the diet plots). So how can we reduce the feeding level of the largest predators while keeping Sprat alive?

Step 6. Calibrate food competition (rPP)

check what happens when we play with rPP

look at food level at larvea size and keep it constant by increasinf kappa when playing with rpp, the rest of the background will adjust itself from this one consaant size

look at resource mortality, choose favorite resource size, that size choose wich rpp give right plankton moratlity

Step 7. Diagnostic plot

```
plotSummary(sim_optim)
```

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

plotDiet2(sim_optim@params,"Cod")

plotGrowthCurves(sim_optim, species_panel = T)
```

```r
# 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?
```