Setting up a zooplankton model using mizer

Patrick Sykes

2021-03-09

Introduction

Here we will recreate the ZooMSS model (version 2) in Heneghan et al. (2020) using mizer.

There are some benefits to doing this. The main things for us are that mizer has a lot of functionality already that we would like to bring to ZooMSS, and a growing community that we can learn from and contribute to.

There are a few important differences between the models (as I found out along the way). The most fundamental of these is that the governing PDE is in a different form in each model (ZooMSS works in absolute abundance over log-weight size classes, and mizer uses normalised abundance and absolute-weight size classes), and hence the numerics work out a bit differently. As we will see, "recreating" ZooMSS in mizer required changing the numerics in mizer to match how they work out in ZooMSS.

Let's get started! We begin with some setup of required packages.

```
#get required packages
library(devtools)
#most up to date master branch of mizer
#install_github("sizespectrum/mizer")
#install_github("astaaudzi/mizer-rewiring", ref = "temp-model-comp")
#documentation here:
#https://sizespectrum.org/mizer/dev/index.html
library(mizer)
require(tidyverse)

#remotes::install_github("sizespectrum/mizerExperimental")
library(mizerExperimental)

#library(plotly)
```

Set-up mizer model

Next let's read in the parameters from ZooMSS.

```
groups <-read_csv("data/TestGroups_mizer.csv")
```

```
## Parsed with column specification:
## cols(
##   species = col_character(),
##   Type = col_character(),
##   FeedType = col_character(),
##   Prop = col_double(),
##   w_min = col_double(),
##   w_inf = col_double(),
##   w_mat = col_double(),
```

```
##   gamma = col_double(),
##   q = col_double(),
##   PPMRscale = col_double(),
##   PPMR = col_double(),
##   FeedWidth = col_double(),
##   GrossGEscale = col_double(),
##   Carbon = col_double(),
##   Repro = col_double(),
##   PlotColour = col_character(),
##   interaction_resource = col_double()
## )
```

```
groups$w_min <- 10^groups$w_min #convert from log10 values
groups$w_inf <- 10^groups$w_inf
groups$w_mat <- 10^groups$w_mat
groups$h <- 1e50 # should be Inf, but that breaks the calculations. Massive value still works out to ef
groups$ks <- 0 #turn off standard metabolism
#todo - ramp up constant repro for coexistence

# read interaction matrix
# get the interaction matrix - actually I think we can leave this out. Default is all 1s, which is the
theta <- readRDS("data/zoomss_inter.rds")[,-1]
```

We will pass these parameters to mizer to set up a new multispecies model.

```
ID <- 223 #index of environmental data to choose
envirofull <- readRDS("data/envirofull_20200317.RDS")
enviro_row <- envirofull[envirofull$cellID==ID,]

#set up the fixed phyoplankton spectrum
phyto_fixed <- function(params, n, n_pp, n_other, rates, dt, kappa=params@resource_params$kappa, lambda=
  npp <- kappa*params@w_full^(1-lambda) / params@dw_full #returns the fixed spectrum at every time step
  npp[params@w_full > params@resource_params$w_pp_cutoff* (1 - 1e-06)] <- 0
  return(npp)
}

mf.params <- newMultispeciesParams(species_params=groups,
                                   interaction=NULL, #NULL sets all to 1, no strict herbivores
                                   no_w = 178, #number of zoo+fish size classes;
                                   min_w_pp = 10^(-14.4), #minimum phyto size. Note: use -14.4, not -14
                                   w_pp_cutoff = 10^(enviro_row$phyto_max), #maximum phyto size
                                   n = 0.7, #The allometric growth exponent used in ZooMSS
                                   z0pre = 1, #external mortality (senescence)
                                   z0exp = 0.3,
                                   resource_dynamics = "phyto_fixed",
                                   kappa = 10^(enviro_row$phyto_int),
                                   lambda = 1-enviro_row$phyto_slope,
                                   #RDD = constantRDD(species_params = groups) #first go at this
                                   #pred_kernel = ... #probably easiest to just import this/pre-calcula
)
```

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

```
#checks that there are as many phytoplankton size classes as ZooMSS
length(which(mf.params@initial_n_pp>0)) == length(seq(-14.5,enviro_row$phyto_max, by = 0.1))
```

```
## [1] TRUE
```

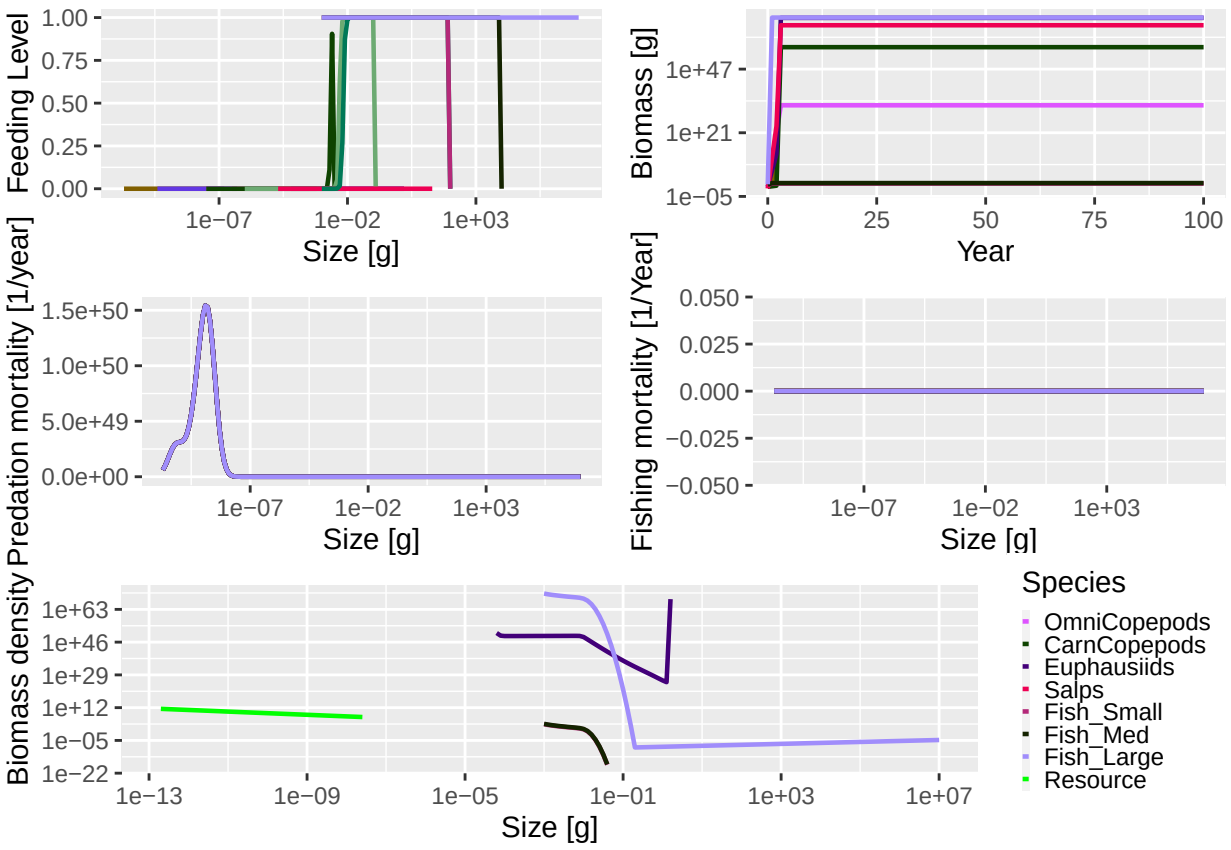
Now do some fiddling to make the new MizerParams object match the ZooMSS parameters. Note: this chunk is adapted from `fZooMSS_setup.R`, found at https://github.com/MathMarEcol/ZooMSS/.

```
mf.params@other_params$temp_eff <-  matrix(2.^((enviro_row$sst - 30)/10), nrow = length(mf.params@speci

setZooMizerConstants <- function(params, Groups, sst){
  #### CALCULATES CONSTANT BITS OF THE MODEL FUNCTIONS FOR EACH GROUP
  SearchVol <- getSearchVolume(params)
  M_sb <- getExtMort(params)
  ZSpre <- 1 # senescence mortality prefactor
  ZSexp <- 0.3 # senescence mortality exponent

  pred_kernel <- getPredKernel(params)
  prey_weight_matrix <- matrix(params@w_full, nrow = length(params@w), ncol = length(params@w_full), by
  pred_weight_matrix <- matrix(params@w, nrow = length(params@w), ncol = length(params@w_full))

  for (i in 1:nrow(params@species_params)) {
    ## Senescence mortality
    if (params@species_params$Type[i] == "Zooplankton") {
      M_sb[i,] <- ZSpre*(params@w/(params@species_params$w_mat[i]))^ZSexp
      M_sb[i, params@species_params$w_inf[i] < params@w * (1 + 1e-06)] <- 0
      M_sb[i, params@species_params$w_mat[i] > params@w * (1 - 1e-06)] <- 0
    }

    if (params@species_params$Type[i] == "Fish") {
      M_sb[i,] <- 0.1*ZSpre*(params@w/(params@species_params$w_mat[i]))^ZSexp
      M_sb[i, params@species_params$w_inf[i] < params@w * (1 + 1e-06)] <- 0
      M_sb[i, params@species_params$w_mat[i] > params@w * (1 - 1e-06)] <- 0
    }

    ### Search volume
    SearchVol[i,] <- (params@species_params$gamma[i])*(params@w^(params@species_params$q[i]))
    SearchVol[i, params@species_params$w_inf[i] < params@w * (1 + 1e-06)] <- 0
    SearchVol[i, params@species_params$w_min[i] > params@w * (1 + 1e-06)] <- 0

    ### Predation Kernels
    if (is.na(params@species_params$PPMRscale[i]) == FALSE){ # If group has an m-value (zooplankton)
      # Calculate PPMR for zooplankton, which changes according to body-size (Wirtz, 2012)
      D.z <- 2*(3*params@w*1e12/(4*pi))^(1/3) # convert body mass g to ESD (um)
      betas <- (exp(0.02*log(D.z)^2 - params@species_params$PPMRscale[i] + 1.832))^3 # Wirtz's equation
      beta_mat <- matrix(betas, nrow = length(params@w), ncol = length(params@w_full))

      # Calculate feeding kernels
      pred_kernel[i, , ] <- exp(-0.5*(log((beta_mat*prey_weight_matrix) /
                                            pred_weight_matrix)/params@species_params$FeedWidth[i])^2) /
        sqrt(2*pi*params@species_params$FeedWidth[i]^2)
      # The feeding kernel of filter feeders is not expected to change  with increasing size so we fix

      # if (param$fixed_filterPPMR == TRUE){
      if (i == 3) {
        pred_kernel[i, , ] <- matrix(pred_kernel[i,44,], nrow = length(params@w), ncol = length(params@
      }
      if (i == 8) {
```

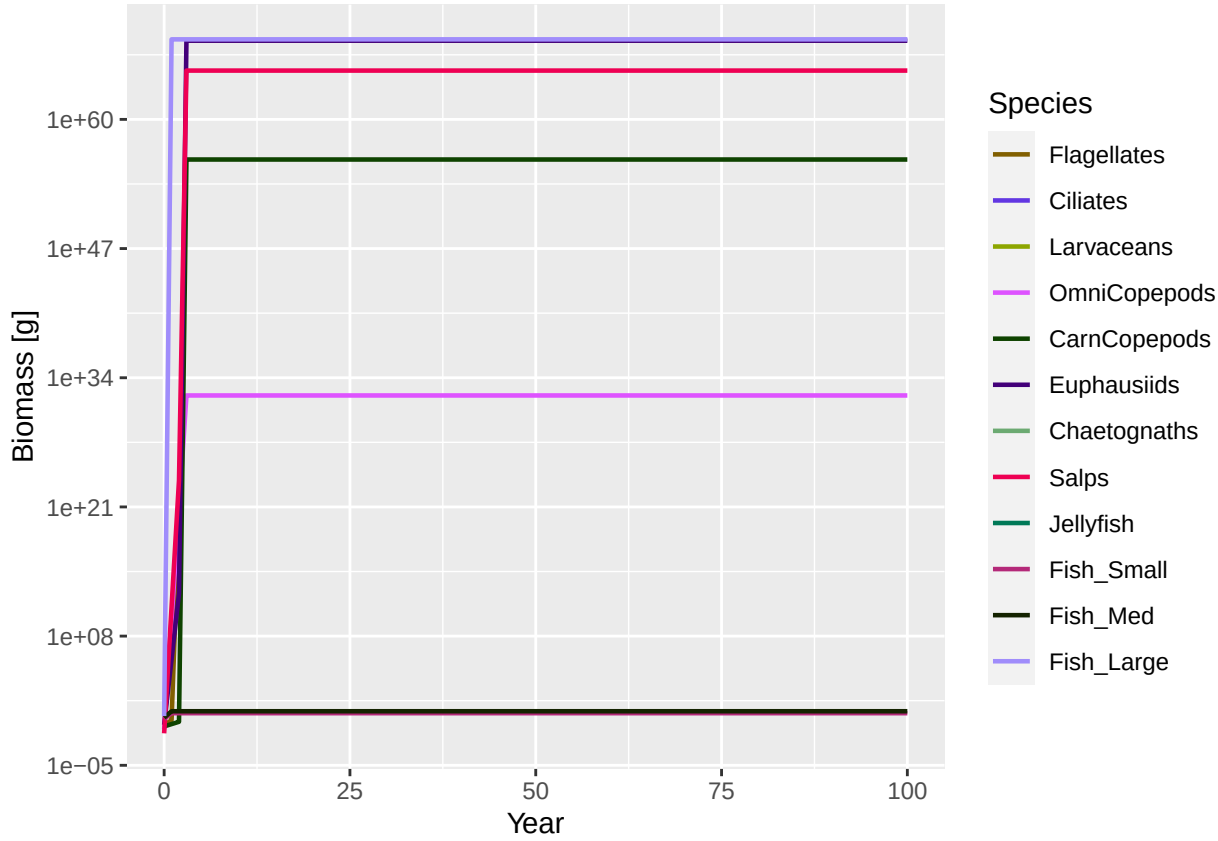
```
        pred_kernel[i, , ] <- matrix(pred_kernel[i,61,], nrow = length(params@w), ncol = length(params@
      }
      # }

    } else { # If group does not have an m-value (fish)
      beta_mat <- matrix(params@species_params$PPMR[i], nrow = length(params@w), ncol = length(params@w

      # Calculate feeding kernels
      pred_kernel[i, , ] <- exp(-0.5*(log((beta_mat*prey_weight_matrix) /
                                          pred_weight_matrix) / params@species_params$FeedWidth[i])^2
        sqrt(2*pi*params@species_params$FeedWidth[i]^2)
    }

  }
  SearchVol[12,178] <- (params@species_params$gamma[12])*(params@w[178]^(params@species_params$q[12]))

  #temperature effect
  M_sb <- params@other_params$temp_eff * M_sb * 10 # Incorporate temp effect on senscence mortality


  params@initial_n_pp <- params@resource_params$kappa * params@w_full^(1 - params@resource_params$lambda
  params@initial_n_pp[params@w_full > params@resource_params$w_pp_cutoff] <- 0


  a_dynam <- (params@resource_params$kappa)*(params@w[1]^(2 - params@resource_params$lambda))#/params@d

  # Initial abundances form a continuation of the plankton spectrum, with a slope of -1
  tempN <- matrix(a_dynam*(params@w)^(-1)/params@dw, nrow = nrow(params@species_params), ncol = length(
  props_z <- params@species_params$Prop[params@species_params$Type == "Zooplankton"] # Zooplankton prop
  tempN[params@species_params$Type == "Zooplankton",] <- props_z * tempN[params@species_params$Type ==
  tempN[params@species_params$Type == "Fish",] <- (1/sum(params@species_params$Type == "Fish")) * tempN

  # For each group, set densities at w > Winf and w < Wmin to 0
  params@species_params$w_min <- params@w[params@w_min_idx]
  tempN[unlist(tapply(round(log10(params@w), digits = 2), 1:length(params@w), function(wx,Winf) Winf < w
  tempN[unlist(tapply(params@w, 1:length(params@w), function(wx,Wmin) Wmin > wx, Wmin = params@species_
  #dimnames(tempN) <- dimnames(params@initial_n)
  params@initial_n[] <- tempN

  SearchVol <- readRDS("data/SearchVol.rds")

  params <- setExtMort(params, z0 = M_sb)
  params <- setSearchVolume(params, search_vol = SearchVol)
  params <- setPredKernel(params, pred_kernel)

  return(params)
}


mf.params <- setZooMizerConstants(params = mf.params, Groups = groups, sst= enviro_row$sst)
```

Try running it:

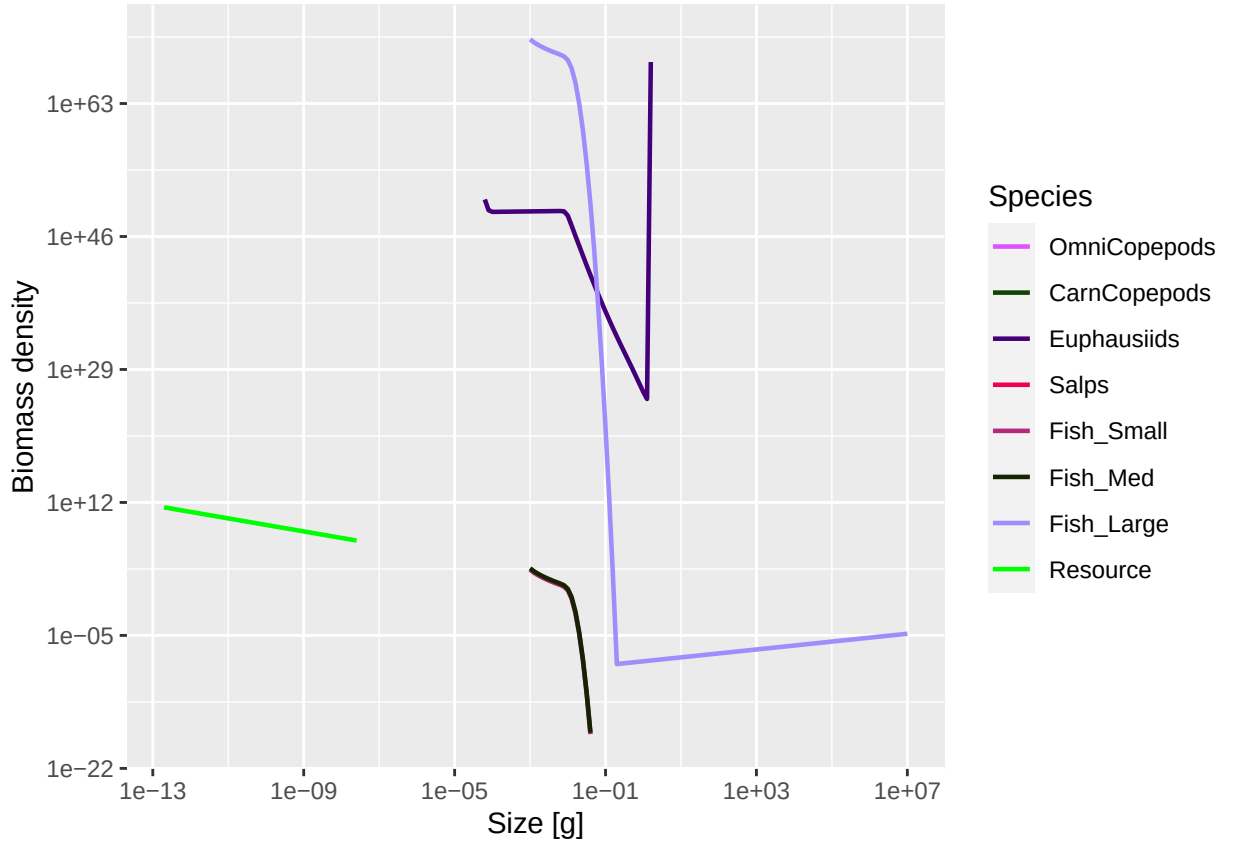
```
sim <- project(mf.params)
plot(sim) #note feeding level means satiation - 0 since there's no satiation in this model.
```




```
plotBiomass(sim)
```




```
#plotlyGrowthCurves(sim,species="macrozooplankton")
#plotlyFeedingLevel(sim)
# feeding level satiation for some groups, except for the seabirds
# macrozooplankton - they are not growing enough,why?
#tuneParams(mf.params)
#plotlyGrowthCurves(sim,percentage = T)
plotSpectra(sim, power = 1)
```



Next thing to do is reproduction. In ZooMSS, this is handled by simply setting the abundance in the smallest size class to be a fixed proportion of the community size spectrum; in short

$$N_i(w_{min}(i)) = prop(i) \sum_{j \neq i} N_j(w_{min}(i)),$$

where $N_i(w)$ is the density of species i in weight class w, and $prop(i)$ is a (fixed) proportion depending on the species.

In mizer, reproduction is linked to metabolism. The abundance in the smallest size class is proportional to the energy available to mature individuals for reproduction - i.e. the energy left over after subtracting metabolic costs (including energy to support growth) from the energy assimilated (by mature individuals) from feeding on prey.

Now, to recreate this in mizer, we need to rewrite mizer's `project_simple()` function. We do this by making a new function, `new_project_simple()`, and using it in place of the default one. Note that there are two key changes in here: to change the boundary conditions ("reproduction") and to account for the conversion from absolute abundance over logged weight classes in ZooMSS to normalised abundance over absolute weight classes in ZooMizer (see the PDF write-up of this - it was a bit of journey to first realise that this could be a problem and then work out how to fix it).

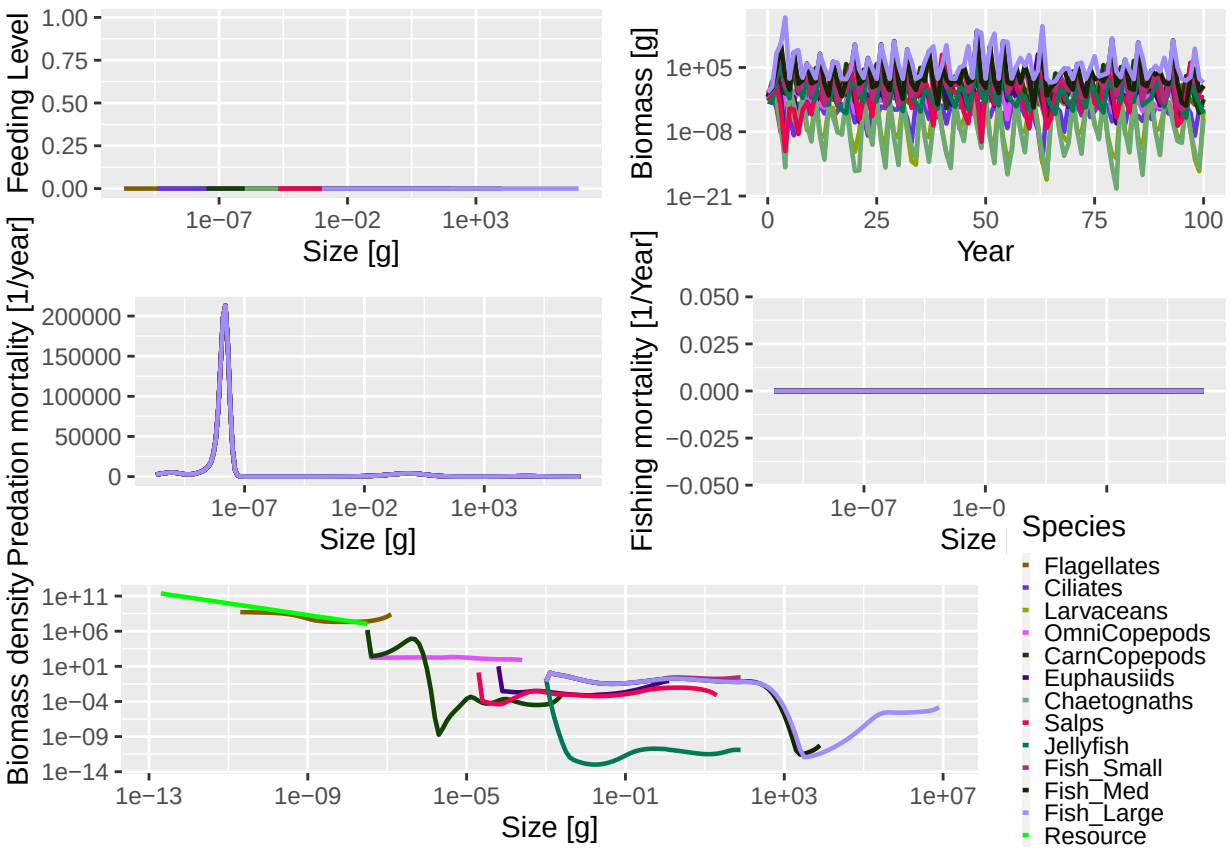
```
new_project_simple <- function(params, n, n_pp, n_other, t, dt, steps,
                               effort, resource_dynamics_fn, other_dynamics_fns,
                               rates_fns, ...) {
  # Handy things
  no_sp <- nrow(params@species_params) # number of species
  no_w <- length(params@w) # number of fish size bins
  idx <- 2:no_w
```

```r
w_max_idx <- params@w_min_idx
for (i in 1:length(w_max_idx)) {
  w_max_idx[i] <- which(round(log10(params@w),2) == round(log10(params@species_params$w_inf[i]),2))
}

fish_grps <- which(params@species_params$Type == "Fish")

if(sum(params@species_params$Type == "Zooplankton") > 1){ # If there's only one zoo group, then you d
  w0idx <- which(params@w_min_idx > min(params@w_min_idx) & is.na(params@species_params$Prop) == FALSE
  w0mins <- params@w_min_idx[w0idx]
  props_z <- params@species_params$Prop[w0idx] # Zooplankton proportions
  }

# Hacky shortcut to access the correct element of a 2D array using 1D notation
# This references the egg size bracket for all species, so for example
# n[w_min_idx_array_ref] = n[,w_min_idx]
w_min_idx_array_ref <- (params@w_min_idx - 1) * no_sp + (1:no_sp)
# Matrices for solver
a <- matrix(0, nrow = no_sp, ncol = no_w)
b <- matrix(0, nrow = no_sp, ncol = no_w)
S <- matrix(0, nrow = no_sp, ncol = no_w)

for (i_time in 1:steps) {
  r <- rates_fns$Rates(
    params, n = n, n_pp = n_pp, n_other = n_other,
    t = t, effort = effort, rates_fns = rates_fns, ...)

  # Update time
  t <- t + dt

  # Update other components
  n_other_current <- n_other  # So that the resource dynamics can still
  # use the current value
  # for (component in names(params@other_dynamics)) {
  #   n_other[[component]] <-
  #     other_dynamics_fns[[component]](
  #       params,
  #       n = n,
  #       n_pp = n_pp,
  #       n_other = n_other_current,
  #       rates = r,
  #       t = t,
  #       dt = dt,
  #       component = component,
  #       ...
  #     )
  # }

  # Update resource
  # n_pp <- resource_dynamics_fn(params, n = n, n_pp = n_pp,
  #                              n_other = n_other_current, rates = r,
  #                              t = t, dt = dt, ...)
  n_pp <- n_pp
```

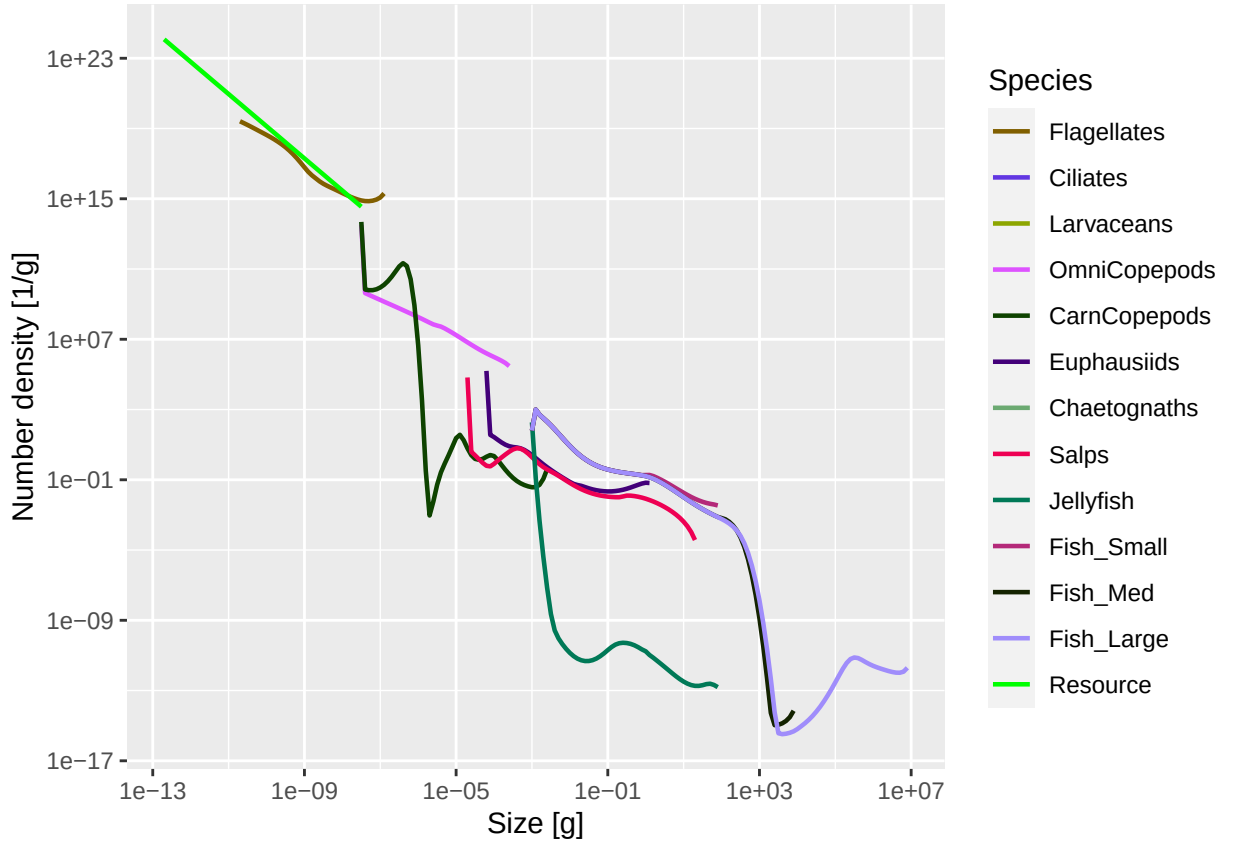
```
    # Iterate species one time step forward:
    # a_{ij} = - g_i(w_{j-1}) / dw_j dt
    a[, idx] <- sweep(-r$e_growth[, idx - 1, drop = FALSE] * dt, 2,
                      params@w[idx-1], "/") * 10^(-0.1) / log(10)
    # b_{ij} = 1 + g_i(w_j) / dw_j dt + \mu_i(w_j) dt
    b[] <- 1 + sweep(r$e_growth * dt , 2, params@w, "/") /log(10) + r$mort * dt * 0.1
    # S_{ij} <- N_i(w_j)
    S[,idx] <- n[, idx, drop = FALSE]
    # Update n
    # for (i in 1:no_sp) # number of species assumed small, so no need to
    #                      vectorize this loop over species
    #     for (j in (params@w_min_idx[i]+1):no_w)
    #         n[i,j] <- (S[i,j] - A[i,j]*n[i,j-1]) / B[i,j]
    # This is implemented via Rcpp
    n <- inner_project_loop(no_sp = no_sp, no_w = no_w, n = n,
                            A = a, B = b, S = S,
                            w_min_idx = params@w_min_idx)


  # Update first and last size groups of n
  #TODO: Make this a little less hacky
  #n[1,1] <- params@species_params$Prop[1]*n_pp[length(params@w_full)-length(params@w)+1]
  if(sum(params@species_params$Type == "Zooplankton") > 1){ # If you only have one zoo group, it will b
    for(i in 1:length(w0idx)){
      w_min_curr <- w0mins[i]
      exclude_mins <- w0idx[which(w0mins == w_min_curr)]
      n[w0idx[i], w_min_curr] <- props_z[i] * sum(n[-exclude_mins, w_min_curr])
    }
  }

  fish_mins <- unlist(params@w_min_idx[params@species_params$Type == "Fish"])


  if(sum(params@species_params$Type == "Fish") > 1 && sum(params@species_params$Type == "Zooplankton") >
    n[fish_grps,fish_mins] <- (1/length(fish_grps))*(colSums(n[-fish_grps,fish_mins]))
  }else{
    n[fish_grps, fish_mins] <- (1/length(fish_grps))*sum(n[-fish_grps, fish_mins])
  }

  for (i in 1:no_sp) {
    n[i, w_max_idx[i]] <- 0
  }


}
  return(list(n = n, n_pp = n_pp, n_other = n_other, rates = r))
}


#assign new project function in namespace
environment(new_project_simple) <- asNamespace('mizer')
assignInNamespace("project_simple", new_project_simple, ns = "mizer")
```

Now let's try that out:

```
sim2 <- project(mf.params, t_max = 100, dt = 001)
plot(sim2)
```




```
plotSpectra(sim2, power = 0)
```



Still missing is the differential prey nutrition based on prey carbon content. We'll do that by editing the `mizerEncounter()` function found in `project_methods.R`, and using `setRateFunction(params, "Encounter", "myEncounter")`. It might be possible to instead edit the chunk above too to incorporate the different method (since we've gone there anyway...)

```
#set up matrix of pred nutrition given prey, dims (pred species) x (prey species)
setassim_eff <- function(groups){
  assim_eff = matrix(groups$GrossGEscale * groups$Carbon, nrow = nrow(groups), ncol = nrow(groups))
  return(t(assim_eff))
}

new_Encounter <- function(params, n, n_pp, n_other, t, ...) {

  # idx_sp are the index values of params@w_full such that
  # params@w_full[idx_sp] = params@w
  idx_sp <- (length(params@w_full) - length(params@w) + 1):length(params@w_full)

  # Note: removed the FFT code because it does not apply to this case.

  # If the feeding kernel does not have a fixed predator/prey mass ratio
  # then the integral is not a convolution integral and we can not use fft.
  # In this case we use the code from mizer version 0.3

  # n_eff_prey is the total prey abundance by size exposed to each
  # predator (prey not broken into species - here we are just working out
  # how much a predator eats - not which species are being eaten - that is
  # in the mortality calculation
```

```
  # \sum_j \theta_{ij} N_j(w_p) w_p dw_p
  assim_prey <- params@other_params$assim_eff * params@interaction
  n_eff_prey <- sweep(assim_prey %*% n, 2,
                      params@w * params@dw, "*", check.margin = FALSE)
  # pred_kernel is predator species x predator size x prey size
  # So multiply 3rd dimension of pred_kernel by the prey biomass density
  # Then sum over 3rd dimension to get consumption rate of each predator by
  # predator size
  # This line is a bottle neck
  phi_prey_species <- rowSums(sweep(
    params@pred_kernel[, , idx_sp, drop = FALSE],
    c(1, 3), n_eff_prey, "*", check.margin = FALSE), dims = 2)
  # Eating the background
  # This line is a bottle neck
  phi_prey_background <- params@other_params$assim_phyto * params@species_params$interaction_resource *
    rowSums(sweep(
      params@pred_kernel, 3, params@dw_full * params@w_full * n_pp,
      "*", check.margin = FALSE), dims = 2)
  encounter <- params@other_params$temp_eff * params@search_vol * (phi_prey_species + phi_prey_backgrou
  dimnames(encounter) <- dimnames(params@metab)

  # Add contributions from other components
  for (i in seq_along(params@other_encounter)) {
    encounter <- encounter +
      do.call(params@other_encounter[[i]],
              list(params = params,
                   n = n, n_pp = n_pp, n_other = n_other,
                   component = names(params@other_encounter)[[i]], ...))
  }

  return(encounter)
}
```

There are a few more things to fix up so that the temperature effect is taken into account, and to ensure that Type I feeding is used:

```
new_PredRate <- function(params, n, n_pp, n_other, t, feeding_level, ...)
{
  n_total_in_size_bins <- sweep(n, 2, params@dw, "*",
                                check.margin = FALSE)
  pred_rate <- sweep(params@pred_kernel, c(1, 2), (1 - feeding_level) * params@other_params$temp_eff *
                     "*", check.margin = FALSE)
  pred_rate <- colSums(aperm(pred_rate, c(2, 1, 3)), dims = 1)
  return(pred_rate)
}

new_EReproAndGrowth <- function(params, n, n_pp, n_other, t, encounter, feeding_level, ...)
{
  return(encounter - params@metab)
}

newFeedingLevel <- function (params, n, n_pp, n_other, t, encounter, ...)
{
  return(encounter * 0) #zero feeding level corresponds to type 1 feeding
```

```
}
```

```
fZooMizer_run <- function(groups, input){

  kappa = 10^(input$phyto_int)
  lambda = 1-input$phyto_slope
  chlo = input$chlo
  sst = input$sst
  dt = input$dt
  tmax = input$tmax

  #data
  # groups$w_min <- 10^groups$w_min #convert from log10 values
  # groups$w_inf <- 10^groups$w_inf
  # groups$w_mat <- 10^groups$w_mat
  # groups$h <- 1e50 # should be Inf, but that breaks the calculations. Massive value still works out t
  # groups$ks <- 0 #turn off standard metabolism
  #todo - ramp up constant repro for coexistence

  mf.params <- new_newMultispeciesParams(species_params=groups,
                                         interaction=NULL, #NULL sets all to 1, no strict herbivores
                                         #min_w = 10^(-10.7),
                                         #max_w = 10^7* (1 + 1e-06),
                                         #no_w = 178, #number of zoo+fish size classes;
                                         w_full = 10^seq(from = -14.5, to = (log10(max(groups$w_inf)) +
                                         #min_w_pp = 10^(-14.4), #minimum phyto size. Note: use -14.4,
                                         w_pp_cutoff = 10^(input$phyto_max)* (1 + 1e-06), #maximum phyt
                                         n = 0.7, #The allometric growth exponent used in ZooMSS
                                         z0pre = 1, #external mortality (senescence)
                                         z0exp = 0.3,
                                         resource_dynamics = "phyto_fixed",
                                         kappa = kappa,
                                         lambda = lambda,
                                         RDD = constantRDD(species_params = groups), #first go at this
                                         #pred_kernel = ... #probably easiest to just import this/pre-c
  )

  mf.params@species_params$w_min <- groups$w_min  #fix Mizer setting the egg weight to be one size larg
  #mf.params@initial_n[] <- readRDS("data/initialn.RDS")

  temp_eff <-  matrix(2.^((sst - 30)/10), nrow = length(mf.params@species_params$species), ncol = lengt

  mf.params@other_params$assim_eff <- setassim_eff(groups)
  cc_phyto <- 0.1 #carbon content of phytoplankton
  mf.params@other_params$assim_phyto <- mf.params@species_params$GrossGEscale * cc_phyto #assimilation

  mf.params@other_params$temp_eff <-  matrix(2.^((sst - 30)/10), nrow = length(mf.params@species_params

  mf.params <- setZooMizerConstants(params = mf.params, Groups = groups, sst = input$sst)
  #mf.params@initial_n[] <- readRDS("data/initialn.RDS")

  #mf.params <- setParams(mf.params)

  # mf.params <- setRateFunction(mf.params, "PredRate", "new_PredRate")
```

```
  mf.params <- setRateFunction(mf.params, "EReproAndGrowth", "new_EReproAndGrowth")
  mf.params <- setRateFunction(mf.params, "FeedingLevel", "newFeedingLevel")
  mf.params <- setRateFunction(mf.params, "Encounter", "new_Encounter")
  mf.params <- setRateFunction(mf.params, "PredRate", "new_PredRate")
  mf.params <- setReproduction(mf.params, repro_prop = matrix(0, nrow = nrow(mf.params@psi), ncol = nco


  #mf.params <- setmort_test(mf.params, sst)
  M_sb <- getExtMort(mf.params)
  M_sb[] <- readRDS("data/mu_b.RDS")
  temp_eff <-  matrix(2.^((sst - 30)/10), nrow = length(mf.params@species_params$species), ncol = lengt
  M_sb <- temp_eff * M_sb *10 # Incorporate temp effect on senscence mortality

  mf.params <- setExtMort(mf.params, z0 = M_sb)

  sim <- project(mf.params, dt = dt, t_max = tmax, t_save = 1) #TODO: make t_save an input to fZooMizer

  return(sim)
}
```

(There's a bit more going on behind the scenes - usually you can't directly specify `w_full`, but I made changes to the `newMultispeciesParams` and `emptyParams` functions in order to allow this. I'm planning to roll this change back soon.)

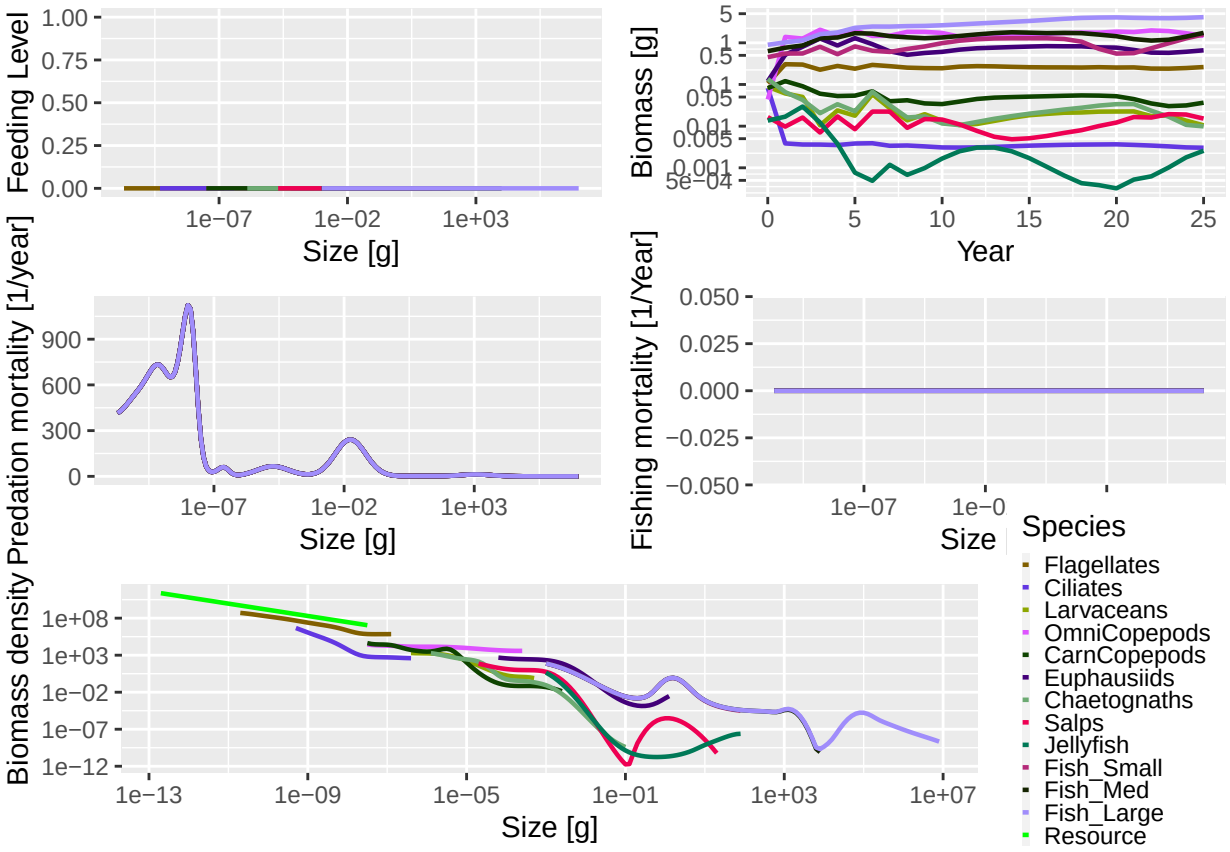
So we load the new versions of mizer functions and then run the new function:

```
environment(new_project_simple) <- asNamespace('mizer')
assignInNamespace("project_simple", new_project_simple, ns = "mizer")

environment(new_newMultispeciesParams) <- asNamespace('mizer')
assignInNamespace("newMultispeciesParams", new_newMultispeciesParams, ns = "mizer")

environment(new_emptyParams) <- asNamespace('mizer')
assignInNamespace("emptyParams", new_emptyParams, ns = "mizer")

enviro_row$tmax <- 25

sim3 <- fZooMizer_run(groups, enviro_row)
```

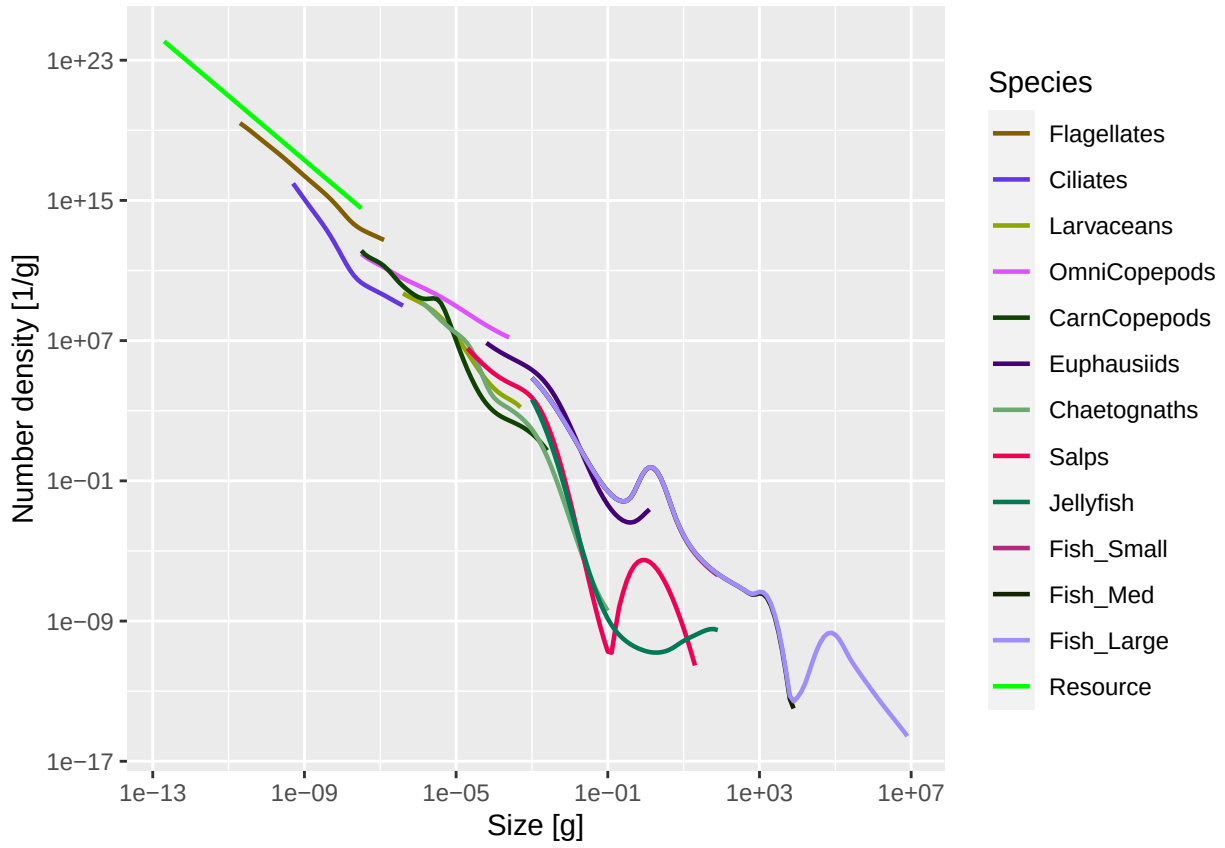
```
## Note: Using z0 = z0pre * w_inf ^ z0exp for missing z0 values.
```

```
## Warning: Unknown or uninitialised column: `constant_reproduction`.
```

```
plot(sim3)
```



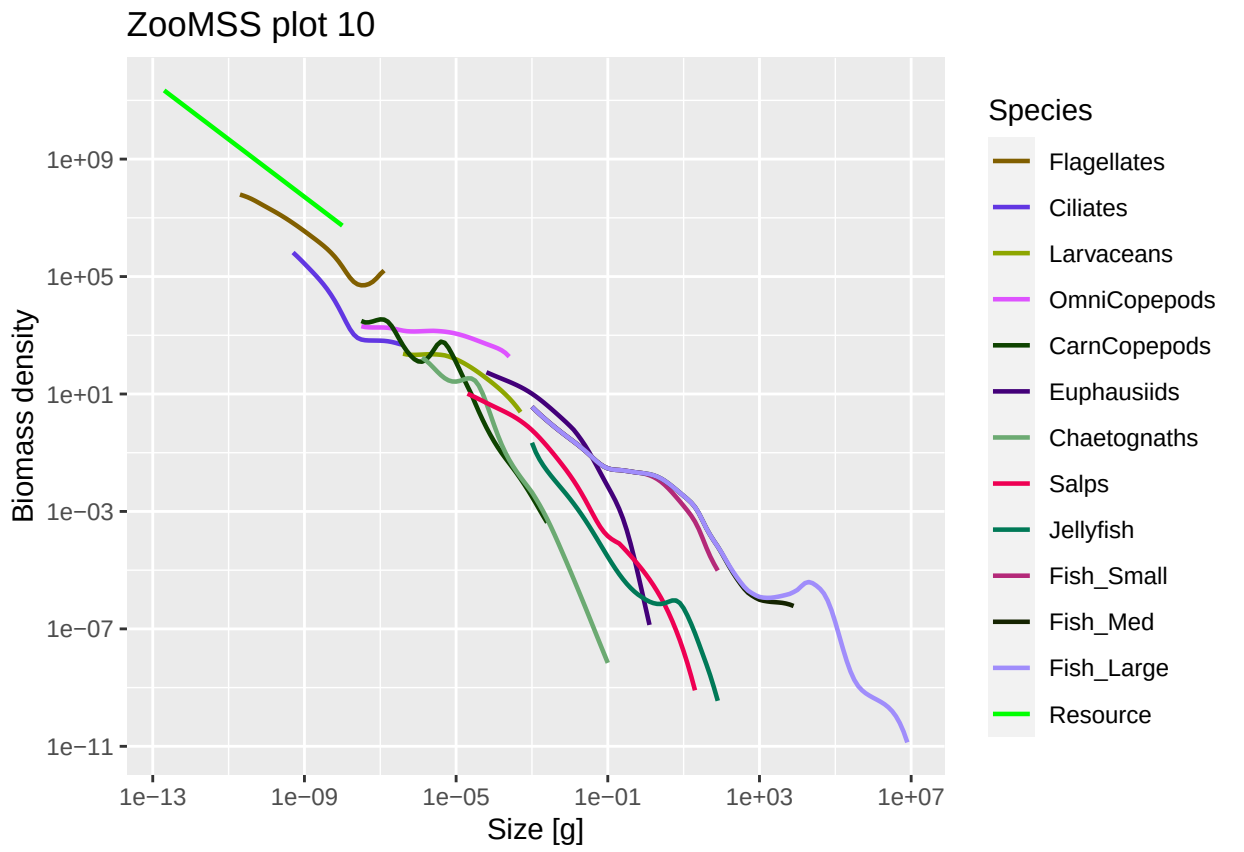

```
plotSpectra(sim3, power = 0)
```



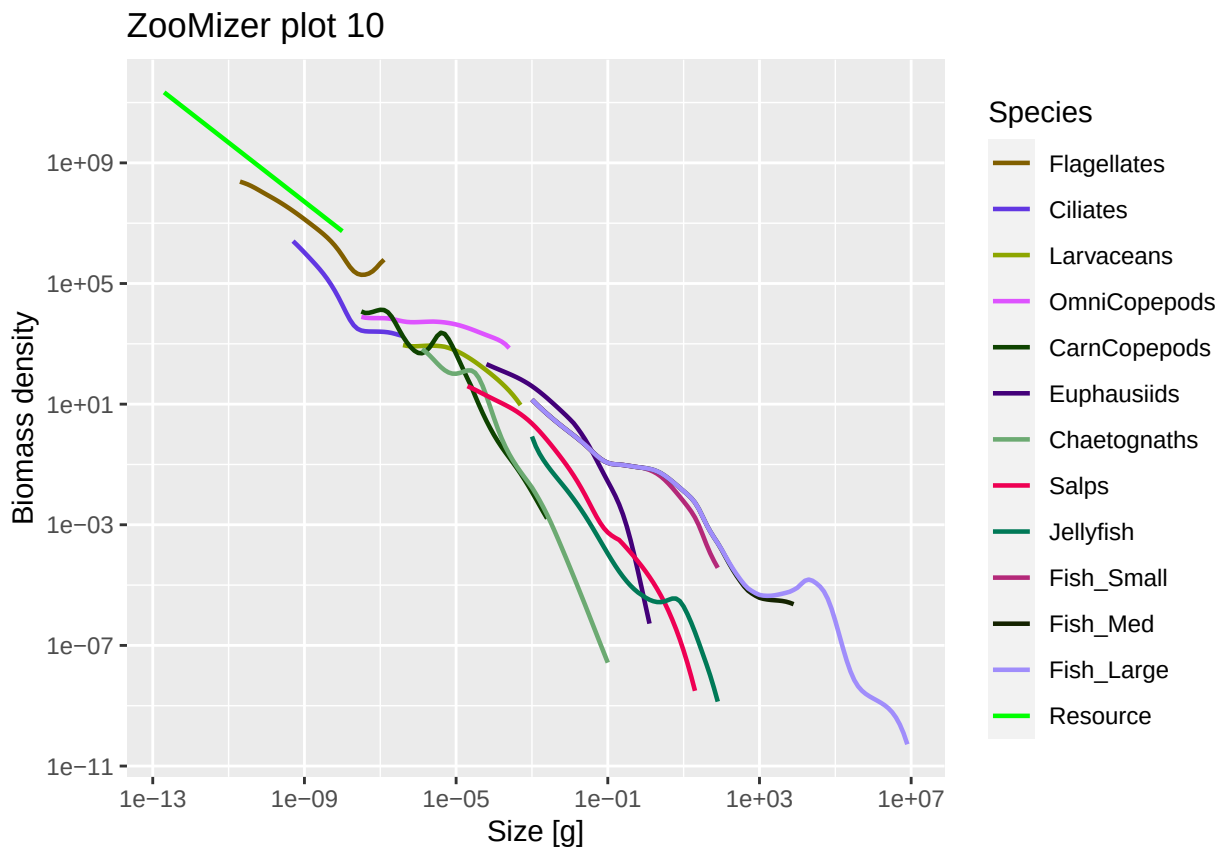
Using a bit of a hack, we can compare this over several grid cells of both models, just to make sure everything works:

```
require(ggplot2)

hack <- sim2 #making a "fake" MizerSim object that we can use plots on.

#match ZooMSS data to that collected from zoomizer
enviro <- readRDS("data/enviro_test20.RDS")
zoomssgrid <- readRDS("Output/res_ZooMizer50yrs.RDS")
zoomssgrid <- zoomssgrid[rank(enviro$CellID)]

w <- matrix(hack@params@w, nrow = nrow(hack@params@species_params), ncol = length(hack@params@w), byrow
dw <- matrix(hack@params@dw, nrow = nrow(hack@params@species_params), ncol = length(hack@params@dw), by

#load in ZooMizer data for comparison
zoomizergrid <- readRDS("test_grid_50yrs.RDS")

tmax <- dim(zoomizergrid[[1]]@n)[1]

pzoomizer <- list()
for (i in 1:length(zoomizergrid)) {
  zoomizergrid[[i]]@n[tmax,,] <- apply(as.array(zoomizergrid[[i]]@n[ceiling(tmax/2):tmax,,]),c(2,3),'mea
  pzoomizer[[i]] <- plotSpectra(zoomizergrid[[i]], time_range = tmax-1, power = 1)+labs(title = paste("
}

#fill fake MizerSim object with ZooMSS data and plot
```

```
pzoomss <- list()
for (i in 1:length(zoomssgrid)) {
   hack@n[i,,] <- zoomssgrid[[i]] / w
   hack@n_pp[i,] <- zoomizergrid[[i]]@n_pp[1,]
   pzoomss[[i]] <- plotSpectra(hack, time_range = i-1, power = 1)+labs(title = paste("ZooMSS plot",i))
}




#have a look
pzoomss[[ceiling(length(zoomizergrid))/2]]
```

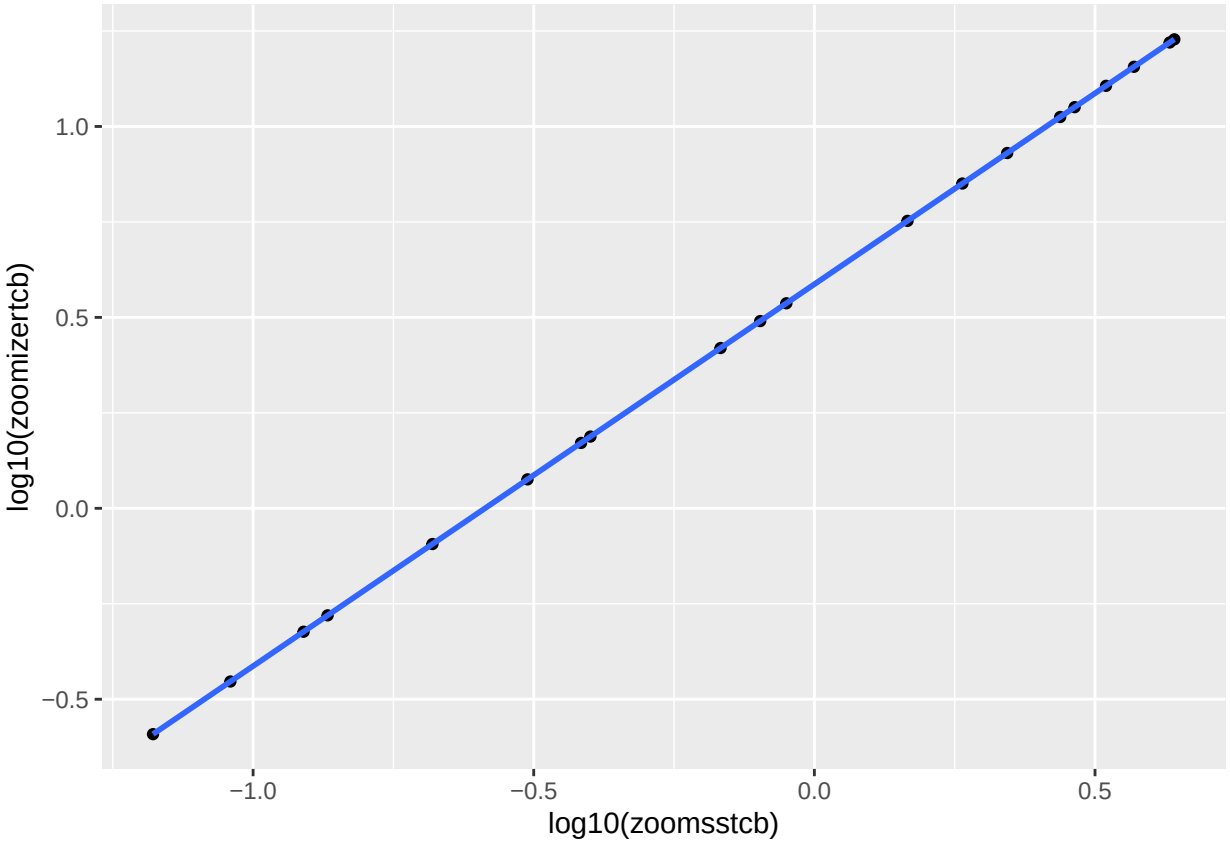



```
pzoomizer[[ceiling(length(zoomizergrid))/2]]
```




```
#compare them all
# p <- list(length(zoomizergrid)*2)
# for (i in 1:length(zoomizergrid)) {
#     p[[2*i-1]] <- pzoomss[[i]]
#     p[[2*i]] <- pzoomizer[[i]]
# }
#p

testdf <- data.frame(zoomsstcb = numeric(length(zoomizergrid)),zoomizertcb = numeric(length(zoomizergri
for (i in 1:nrow(testdf)) {
testdf$zoomsstcb[i] = sum(hack@n[i,,]*w*dw)
testdf$zoomizertcb[i] = sum(zoomizergrid[[i]]@n[tmax,,]*w*dw)
}

(scatter <- ggplot(testdf, aes(x=log10(zoomsstcb), y=log10(zoomizertcb)))+
  geom_point()+
  geom_smooth(method = 'lm'))
```

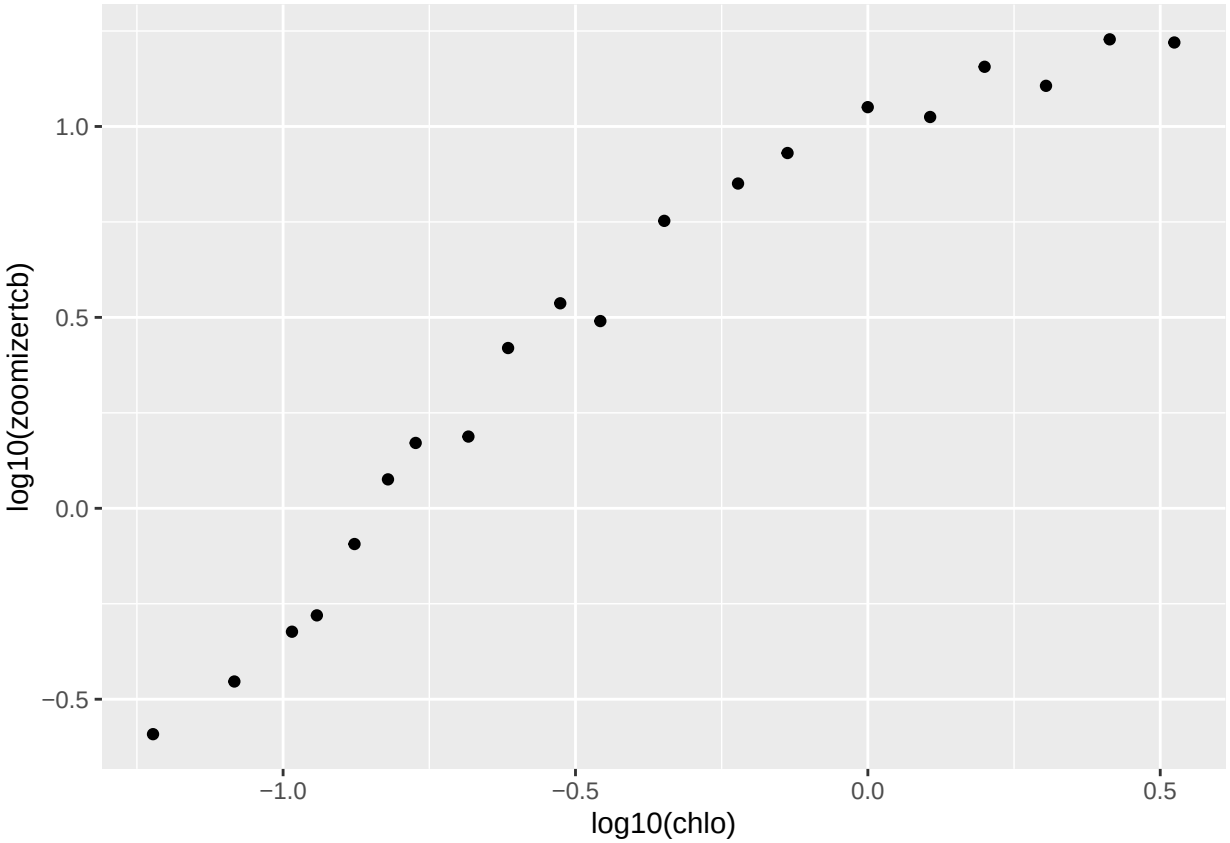
```
## `geom_smooth()` using formula 'y ~ x'
```




```
ggsave("zoomss_zoomizer_comparison_tcb.png",scatter)
```

```
## Saving 6.5 x 4.5 in image
## `geom_smooth()` using formula 'y ~ x'
```

```
testdf$chlo <- enviro$chlo[1:nrow(testdf)]
(chlotcb <- ggplot(data=testdf, aes(x=log10(chlo),y=log10(zoomizertcb)))+
  geom_point())
```




```
ggsave("zoomizer_tcb_vs_chlo.png", chlotcb)
```

```
## Saving 6.5 x 4.5 in image
```

```
fit <- lm(testdf$zoomizertcb~testdf$zoomsstcb)
```

Next steps

There are a few benefits to using (Zoo)Mizer for the next steps in our model development.

- **Modularity**: it's easy to customise the functions used for ingestion (feeding response type), growth, mortality, reproduction, etc. without changing anything else.
- **Fish and fishing**: the functionality already exists in mizer to model different fishing rates and selectivity types, catchability, fishing effort and so on.
- **Phytoplankton**: the phytoplankton abundance function can easily be changed to use semi-chemostat (default) or logistic models, fixed spectrum as in ZooMSS, inputs from ESMs and so on.
- **Extra slots in the `MizerParams` and `MizerSim` classes**: we can use the `other_params` slot to load in extra parameters and include unstructured resources (such as detritus) in the `n_other` slot. Additional slots can also be added by editing the `emptyParams()` function.