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$h <- 1e50 # should be Inf, but that breaks the calculations. Massive value still works out to effectively unlimited feeding as allowed in ZooMSS - not currently being used anyway.  
groups$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 same as in ZooMSS. Included for completeness; it may be useful in future to keep this in.  
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$kappa, lambda=params@resource\_params$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 -14.5, otherwise it makes an extra size class  
 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-calculate it, once dimensions are worked out  
)

## 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))

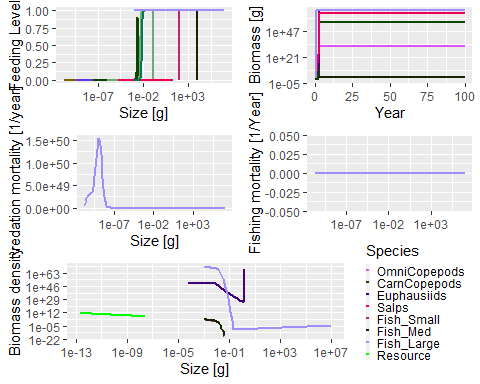
## [1] TRUE

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/>.

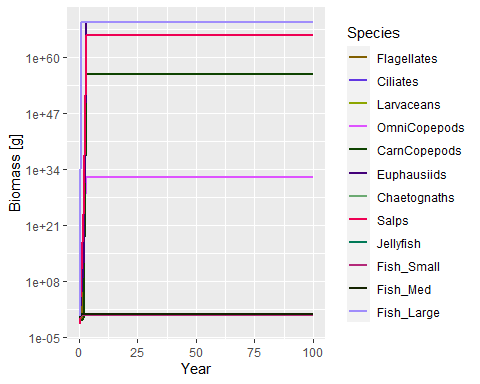
mf.params@other\_params$temp\_eff <- matrix(2.^((enviro\_row$sst - 30)/10), nrow = length(mf.params@species\_params$species), ncol = length(mf.params@w))  
  
setZooMizerConstants <- function(params, Groups, sst){  
 #### CALCULATES CONSTANT BITS OF THE MODEL FUNCTIONS FOR EACH GROUP  
 SearchVol <- 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), byrow = TRUE)  
 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 it here  
  
 # if (param$fixed\_filterPPMR == TRUE){  
 if (i == 3) {  
 pred\_kernel[i, , ] <- matrix(pred\_kernel[i,44,], nrow = length(params@w), ncol = length(params@w\_full), byrow = TRUE)  
 }  
 if (i == 8) {  
 pred\_kernel[i, , ] <- matrix(pred\_kernel[i,61,], nrow = length(params@w), ncol = length(params@w\_full), byrow = TRUE)  
 }  
 # }  
  
 } 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\_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)  
 }  
  
 }  
 SearchVol[12,178] <- (params@species\_params$gamma[12])\*(params@w[178]^(params@species\_params$q[12])) #adding last size class by hand  
  
 #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@dw\_full  
 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@dw[1] # calculate coefficient for initial dynamic spectrum, so that N(w\_phyto) equals N(w\_dynam) at w[1]  
  
 # 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(params@w), byrow = TRUE)  
 props\_z <- params@species\_params$Prop[params@species\_params$Type == "Zooplankton"] # Zooplankton proportions  
 tempN[params@species\_params$Type == "Zooplankton",] <- props\_z \* tempN[params@species\_params$Type == "Zooplankton",] # Set abundances of diff zoo groups based on smallest size class proportions  
 tempN[params@species\_params$Type == "Fish",] <- (1/sum(params@species\_params$Type == "Fish")) \* tempN[params@species\_params$Type=="Fish",] # Set abundandances of fish groups based on smallest size class proportions  
  
 # For each group, set densities at w > Winf and w < Wmin to 0  
 params@species\_params$w\_min <- params@w[params@w\_min\_idx]  
 tempN[unlist(tapply(round(log10(params@w), digits = 2), 1:length(params@w), function(wx,Winf) Winf < wx, Winf = log10(params@species\_params$w\_inf)))] <- 0  
 tempN[unlist(tapply(params@w, 1:length(params@w), function(wx,Wmin) Wmin > wx, Wmin = params@species\_params$w\_min))] <- 0  
 #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:

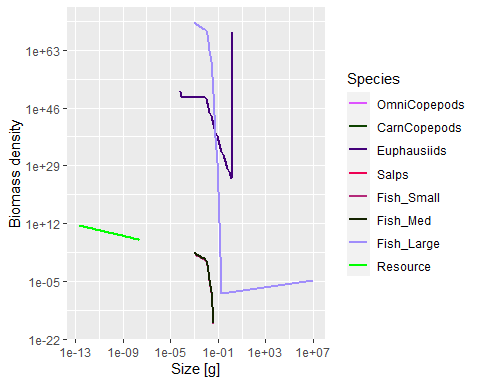
sim <- project(mf.params)  
plot(sim) #note feeding level means satiation - 0 since there's no satiation in this model.



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

where is the density of species in weight class , and is a (fixed) proportion depending on the species.

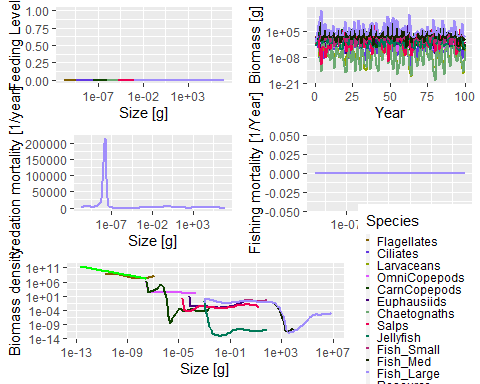
In mizer, reproduction is linked to metabolism. The abundance in the smallest size class is proportional to the energy available to mature individuals for reproduction - i.e. the energy left over after subtracting metabolic costs (including energy to support growth) from the energy assimilated (by mature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

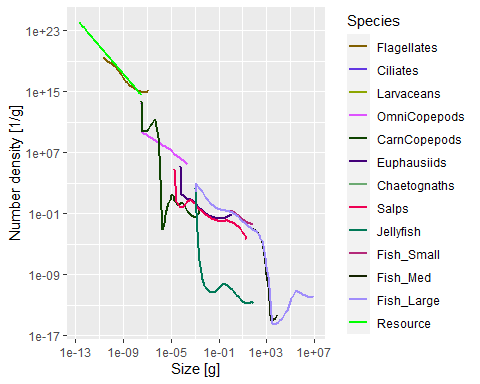
new\_project\_simple <- function(params, n, n\_pp, n\_other, t, dt, steps,  
 effort, resource\_dynamics\_fn, other\_dynamics\_fns,  
 rates\_fns, ...) {  
 # Handy things  
 no\_sp <- nrow(params@species\_params) # number of species  
 no\_w <- length(params@w) # number of fish size bins  
 idx <- 2:no\_w  
 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$Type == "Zooplankton") > 1){ # If there's only one zoo group, then you do not need w0idx. All this stuff gives you info about all zoo groups except the smallest zoo group.  
 w0idx <- which(params@w\_min\_idx > min(params@w\_min\_idx) & is.na(params@species\_params$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 be locked to phyto spectrum so you do not need to do this  
 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") > 1){  
 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\_background)  
 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 \* params@search\_vol \* n\_total\_in\_size\_bins,  
 "\*", 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$h <- 1e50 # should be Inf, but that breaks the calculations. Massive value still works out to effectively unlimited feeding as allowed in ZooMSS  
 # groups$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$w\_inf)) + 0.1), by = 0.1),  
 #min\_w\_pp = 10^(-14.4), #minimum phyto size. Note: use -14.4, not -14.5, otherwise it makes an extra size class  
 w\_pp\_cutoff = 10^(input$phyto\_max)\* (1 + 1e-06), #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 = kappa,  
 lambda = lambda,  
 RDD = constantRDD(species\_params = groups), #first go at this  
 #pred\_kernel = ... #probably easiest to just import this/pre-calculate it, once dimensions are worked out  
 )  
  
 mf.params@species\_params$w\_min <- groups$w\_min #fix Mizer setting the egg weight to be one size larger for some groups.  
 #mf.params@initial\_n[] <- readRDS("data/initialn.RDS")  
  
 temp\_eff <- matrix(2.^((sst - 30)/10), nrow = length(mf.params@species\_params$species), ncol = length(mf.params@w))  
  
 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 efficiency when eating phytoplankton  
  
 mf.params@other\_params$temp\_eff <- matrix(2.^((sst - 30)/10), nrow = length(mf.params@species\_params$species), ncol = length(mf.params@w))  
  
 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 = ncol(mf.params@psi)))  
  
  
 #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 = length(mf.params@w))  
 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\_run  
  
 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.)

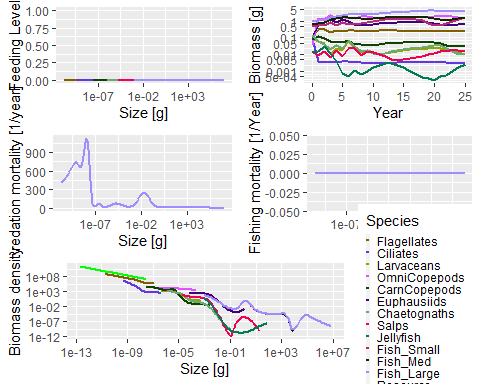
So we load the new versions of mizer functions and then run the new function:

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)

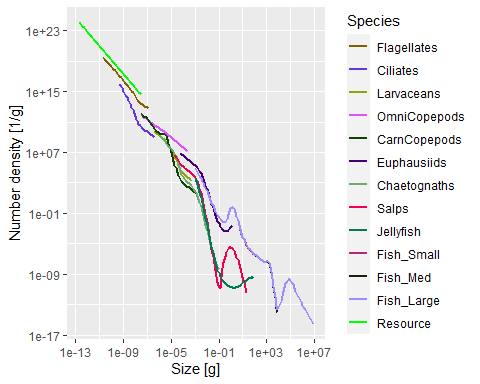
## Note: Using z0 = z0pre \* w\_inf ^ z0exp for missing z0 values.

## Warning: Unknown or uninitialised column: `constant\_reproduction`.

plot(sim3)

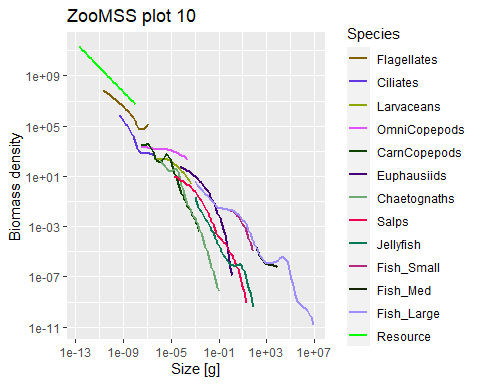


plotSpectra(sim3, power = 0)

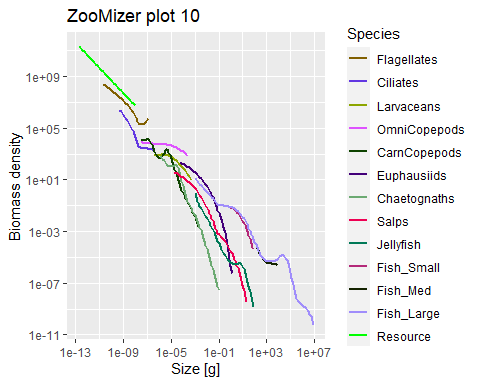


Using a bit of a hack, we can compare this over several grid cells of both models, just to make sure everything works:

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 = TRUE)  
dw <- matrix(hack@params@dw, nrow = nrow(hack@params@species\_params), ncol = length(hack@params@dw), byrow = TRUE)  
  
#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),'mean') #apples-to-apples comparison  
 pzoomizer[[i]] <- plotSpectra(zoomizergrid[[i]], time\_range = tmax-1, power = 1)+labs(title = paste("ZooMizer plot",i))  
}  
  
#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]]

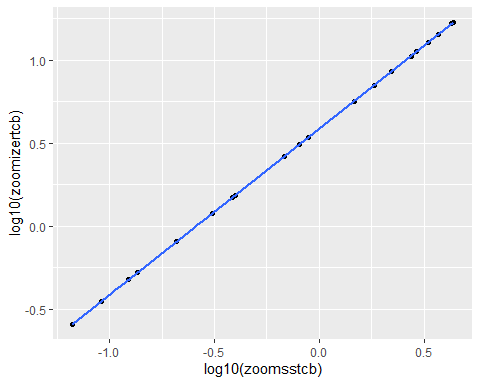


pzoomizer[[ceiling(length(zoomizergrid))/2]]



#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(zoomizergrid)))  
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'))

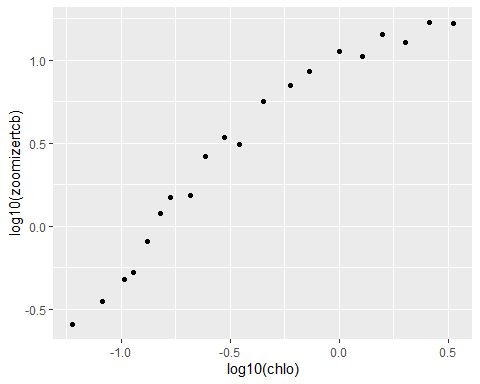
## `geom\_smooth()` using formula 'y ~ x'



ggsave("zoomss\_zoomizer\_comparison\_tcb.png",scatter)

## Saving 5 x 4 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 5 x 4 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.