Daphnia DVM Model Source Code

Bennett McAfee

5/2/2021

Table of Contents

Preface

The following source code is representative of how the diel vertical migration model functions and was used in the creation of the thesis paper. This is not the final code that will be used in the public R package. Minor alterations will be made to allow the code to work in package format. Specifically, the largest changes for the final code will use a different method of translating the size of a Daphnia to index coordinates that allows for

greater variation in the x critical and capacity values, and also use a different method of storing the Daphnia images put into the output of the DaphniaDVM function.

List of Dependencies

This package is intended to work with the TidyVerse and requires the following packages:

dplyr, rlang, ggplot2, ggimage

```
library(ggplot2)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(ggimage)
library(rlang)
```

Private functions

These functions are intended for use by the developer to allow the public functions, intended for user use, to run.

chop

```
basicchop <- function(formula, lower, upper){
  value <- "An error has occurred"
  if (formula > upper){value <- upper}
  if (formula >= lower & formula <= upper){value <- formula}
  if (formula < lower){value <- 0}
  return(value)
}
```

Critcompare

```
critcompare <- function(value, critical){
  if (value > critical){
    return(1)
  } else {return(0)}
}
```

init_fone

```
init_fone <- function(x_crit, x_max, data, depth){

  F1_temp <- integer((x_max - x_crit)/5)

  for (x in seq(from = x_crit, to = x_max, by = 5)){
    u <- x / 5
    F1_temp[u] = critcompare(value = x, critical = x_crit)
  }

  F1_temp2 <- data %>%
    mutate(patch = 2 * round({{depth}}/2))
  F1 <- data.frame("Size" = seq(from = x_crit, to = x_max, by = 5))

  for (row in 1:nrow(F1_temp2)){
    label1 <- as.character(F1_temp2[row, "patch"])

    F1[[label1]] <- F1_temp
  }


  return(F1)
}
```

init_fzero

```
init_fzero <- function(x_crit, x_max){

  F0 = integer(((x_max - x_crit)/5)+1)

  return(F0)
}
```

compute_v

```
# Round x' and x" values to integers?
compute_v <- function(x, alpha, beta, lambda, upsilon, x_crit, x_max, F1,
depth){
  require(rlang)
  x <- {{x}}
  F1 <- {{F1}}
  depth <- {{depth}}
  xfit <- F1[x/5, as.character(depth)]

  # calculating x' and x"
  xp1 <- basicchop(formula = {{x}} - {{alpha}} + {{upsilon}}, lower =
{{x_crit}}, upper = {{x_max}})
  xp2 <- basicchop(formula = {{x}} - {{alpha}}, lower = {{x_crit}}, upper =
{{x_max}})
```

```
  # Retrieve info about x at depth from F1

  rowrefup <- {{x}}/5+1
  if (rowrefup >= {{x_max}}/5){
    rowrefup <- {{x_max}}/5
  }


  rowrefdown <- {{x}}/5-1
  if (rowrefdown <= 0){
    rowrefdown <- 1
  }

  if (xp1 >= x){
    fitxp1plus <- F1[rowrefup, as.character(depth)]
  } else {
    fitxp1plus <- F1[rowrefdown, as.character(depth)]
  }



  fitxp2plus <- F1[rowrefdown, as.character(depth)]


  # Interpolate fitness values
  prop_xp1 <- (xp1 - {{x}}) / 5
  fitxp1 <- ((fitxp1plus-xfit) * prop_xp1) + xfit

  prop_xp2 <- (xp2 - {{x}}) / 5
  fitxp2 <- ((fitxp2plus-xfit) * prop_xp2) + xfit



  # Fitness Calculation
  surv_prob <- (1 - {{beta}})
  starv_prob <- (({{lambda}} * fitxp1) + ((1 - {{lambda}}) * fitxp2))

  v <- surv_prob * starv_prob

  return(v)
}
```

get_patches

```
get_patches <- function(data, depth, food_prob, fish_abun, chl, temp, k,
lsurf, x){
  require(dplyr)
  require(rlang)
```

```
  lengthx <- 10^((log10(x) - log10(10.674)) / 2.093)

  data_sum <- data %>%
    mutate(patch = 2 * round({{depth}}/2)) %>%
    group_by(patch) %>%
    summarise(food_prob = mean({{food_prob}}), predn = mean({{fish_abun}}),
chl = mean({{chl}}), temp = mean({{temp}}), .groups = "drop_last")

  RDSAT <- 0.16504 #RD MODEL SAT. LEVEL & HALFSAT
  RDHALF <- .035
  SSPEED <- 0.15 #DATA FROM CLARK&LEVY-88(m/s & sec.)
  HANDLE <- 1.8
  PDENS <- 10000 #PREY DENSITY SET TO 10/L (10000/m3)
  ASSIM <- 0.6

  data_out <- data_sum %>%
    group_by(patch) %>%
    summarise(Alpha = .007 * ({{x}}^.81) * (exp(.06931 * temp) / 2),
              Light = {{lsurf}} * (exp(-{{k}}*2*patch)),
              React = RDSAT * lengthx * (Light / (RDHALF + Light)),
              Denom1 = pi * SSPEED * PDENS * HANDLE * (React^2) +
0.000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000001,
              Frate = (1/HANDLE) / (1+(1/Denom1)),
              Consum = Frate * (predn/1000) * 108000,
              Beta = Consum / PDENS,
              Lambda = food_prob,
              ups_coeff = (chl * (.029 * temp - .028)) / (.41 * (.3 * chl +
.42)),
              Upsilon = ASSIM * (ups_coeff * (lengthx / 2) ^ (2.43 - .044 *
temp)) /2,
              .groups = "drop_last"
              )

  return(data_out)




}
```

max_v

```
# 'patches' must be a data frame with columns "Depth", "Alpha", "Beta",
"Lambda", "Upsilon"
max_v <- function(x_crit, x_max, F0, F1, D, data, depth, food_prob,
fish_abun, chl, temp, k, lsurf){
  require(rlang)

  F1_ref <- F1
```

```
  for (x in seq(from = x_crit, to = x_max, by = 5)){
    u <- x / 5
    vm <- 0 # vm is the max fitness per patch
    patches <- get_patches({{data}}, {{depth}}, {{food_prob}}, {{fish_abun}},
{{chl}}, {{temp}}, {{k}}, lsurf, x)
    for (i in 1:dim(patches)[1]){
      alpha = pull(patches[i,"Alpha"])
      beta = pull(patches[i,"Beta"])
      lambda = pull(patches[i,"Lambda"])
      upsilon = pull(patches[i,"Upsilon"])
      xdepth = pull(patches[i,"patch"])

      v <- compute_v(x = x, alpha = alpha, beta = beta, lambda = lambda,
upsilon = upsilon, x_crit = x_crit, x_max = x_max, F1 = F1, depth = xdepth)

      depthref <- (xdepth/2) + 2


      F1_ref[u, depthref] <- v



      if (isTRUE(v > vm) == TRUE){
        vm <- v
        F0[u] <- v
        D[u] <- pull(patches[i, "patch"])
      }

    }
  }
  F1 <- F1_ref

  final_out <- list("F1" = F1, "F0" = F0, "D" = D)
  return(final_out)

}
```

DVM Simulation

This function is used to call one DVM simulation using the parameters set by the user.

```
dvmsimulation <- function(n_timesteps, critical, capacity, data, depth,
food_prob, fish_abun, chl, temp, k, lsurf, label = "DVM Simulation:"){

  # STEP 0 - Change global significant figures options

  init_sigfigs <- getOption("digits")
  options(digits = 22)
```

```
  # STEP 1 - initialize fitness arrays
  # cat("\r", "Initializing fitness arrays")

  F0 <- init_fzero(x_crit = critical, x_max = capacity)
  D <- init_fzero(x_crit = critical, x_max = capacity)
  F1 <- init_fone(x_crit = critical, x_max = capacity, data = {{data}}, depth
= {{depth}})

  complete_output <- data.frame(matrix(ncol=4,nrow=0))


  # STEP 2 - Iterate over timesteps, getting max fitness values

  for (t in 1:n_timesteps){

    maxv_list <- max_v(x_crit = {{critical}}, x_max = {{capacity}}, F0 = F0,
F1 = F1, D = D, data = {{data}}, depth = {{depth}}, food_prob =
{{food_prob}}, fish_abun = {{fish_abun}}, chl = {{chl}}, temp = {{temp}}, k =
{{k}}, lsurf = {{lsurf}})
    F1 <- maxv_list$F1
    F0 <- maxv_list$F0
    D <- maxv_list$D


    # STEP 3 - Print/log fitness value and optimal index with t
    new_df <- data.frame("Timestep" = rep(t, length(F0)), "Size" =
seq(critical, capacity, 5), "Fitness" = F0, "Depth" = D)
    complete_output <- rbind(complete_output, new_df)



    # STEP 4 - Display progress
    label = {{label}}
    # cat("\r", label, "Timesteps complete:", t, "of", n_timesteps)
  }

  # STEP 5 - display results and return global options to prior values
  options(digits = init_sigfigs)
  return(complete_output)

}
```

Visualization Functions

DVMSim Prepper

```
singlecurvetable <- function(dvmsim, maxdepth){
  require(dplyr)

  maxdepth <- {{maxdepth}}
  last_timestep <- dvmsim %>% filter(Timestep == max(Timestep))

  depths <- seq(0, maxdepth, by = 2)
  proportionsDX <- integer(length = length(depths))

  for (i in 1:dim(last_timestep)[1]){
    opt_depth = pull(last_timestep[i,], var = "Depth")
    curve <- dnorm(depths, mean = opt_depth, sd = 3.841476) # SD comes from
mean of Sparkling Lake field data 6th Aug 1991
    proportionsDX <- proportionsDX + curve
  }

  final_prop <- data.frame("Depth" = depths, "Abs_Count" = proportionsDX)
  final_prop <- final_prop %>%
    mutate(Count = Abs_Count / sum(Abs_Count))
  return(final_prop)
}
```

Comparison visualization

```
dvmfigure <- function(day, night, day_lg, day_sm, night_lg, night_sm){
  require(ggplot2)
  require(dplyr)
  require(ggimage)

  # Known Data

  day$Time = "Day"
  night$Time = "Night"
  combined_df <- rbind(day, night)


  # Predicted Data
  sum_com_df <- combined_df %>% group_by(Depth) %>% summarise(Total =
max(Count), .groups = "drop_last")

  # Image Links:
  # Red:
https://drive.google.com/uc?export=download&id=1qWv097oYGA3jNXomYllfT_8Cw44uc
mPO
  # Blue:
https://drive.google.com/uc?export=download&id=1zJ57MAc_E6i5gxk7NqzT9obpU2UXe
```

```
oUN

   predicted_df_lg <- data.frame(Depth = c(day_lg, night_lg), Placement =
c((sum_com_df$Total[sum_com_df$Depth ==
day_lg])+((sum_com_df$Total[sum_com_df$Depth == day_lg])*0.3),
(sum_com_df$Total[sum_com_df$Depth ==
night_lg])+((sum_com_df$Total[sum_com_df$Depth == night_lg])*0.4)), Image =
c("https://drive.google.com/uc?export=download&id=1qWv097oYGA3jNXomYllfT_8Cw4
4ucmPO",
"https://drive.google.com/uc?export=download&id=1zJ57MAc_E6i5gxk7NqzT9obpU2UX
eoUN"))
   predicted_df_sm <- data.frame(Depth = c(day_sm, night_sm), Placement =
c((sum_com_df$Total[sum_com_df$Depth ==
day_sm])+((sum_com_df$Total[sum_com_df$Depth == day_sm])*0.1),
(sum_com_df$Total[sum_com_df$Depth ==
night_sm])+((sum_com_df$Total[sum_com_df$Depth == night_sm])*0.2)), Image =
c("https://drive.google.com/uc?export=download&id=1qWv097oYGA3jNXomYllfT_8Cw4
4ucmPO",
"https://drive.google.com/uc?export=download&id=1zJ57MAc_E6i5gxk7NqzT9obpU2UX
eoUN"))


   ggplot(data=combined_df, aes(x=Depth, y=Count, fill = Time))+
     geom_bar(alpha = 0.3, stat="identity", position = "identity")+
     labs(x="Depth (m)", y="Daphnia Abundance (% of Total Population)")+
     coord_flip() + scale_x_reverse(limits = c(NA,-1), breaks =
seq(0,max(combined_df$Depth), 2))+
     theme_bw()+
     geom_image(data = predicted_df_lg, aes(x = Depth, y = Placement, image =
Image), size = 0.07, inherit.aes = FALSE)+
     geom_image(data = predicted_df_sm, aes(x = Depth, y = Placement, image =
Image), size = 0.05, inherit.aes = FALSE)


}
```

Public Functions

Lux to PPFD/Microeinstein conversion

This function is for the convenient conversion of Lux units of light to PPFD aka Microeinsteins. This is useful for putting the surface light in proper units.

```
luxtoPPFD <- function(lux, calibrationfactor = 0.0185){
  answer = lux * calibrationfactor
  return(answer)
}
```

```
luxtoPPFD(100000)
```

```
## [1] 1850
```

DaphniaDVM - The Primary Function

This function is the primary function for use by the user. DaphniaDVM runs two DVM simulations, one for day and one for night, and outputs the results in the form of a ggplot figure by default, or in the form of a dataframe if the kstest_out argument is set to TRUE.

```
DaphniaDVM <- function(n_timesteps, critical, capacity, data, depth,
food_prob, fish_abun, chl, temp, k, light_day, light_night, kstest_out =
FALSE){
  require(ggplot2)
  require(ggimage)
  require(dplyr)
  require(rlang)

  # cat("\n", "Simulating Daytime")

  dvmsimday <- dvmsimulation(n_timesteps = {{n_timesteps}}, critical =
{{critical}}, capacity = {{capacity}}, data = {{data}}, depth = {{depth}},
food_prob = {{food_prob}}, fish_abun = {{fish_abun}}, chl = {{chl}}, temp =
{{temp}}, k = {{k}}, lsurf = {{light_day}}, label = "Day Sim")

  dvmsimday2 <- dvmsimday %>% filter(Timestep == {{n_timesteps}})
  daphnia_day_lg <- dvmsimday2$Depth[dvmsimday2$Size == {{capacity}}]
  daphnia_day_sm <- dvmsimday2$Depth[dvmsimday2$Size == ({{critical}} + 5)]

  # cat("\n", "Simulating Nighttime")

  dvmsimnight <- dvmsimulation(n_timesteps = {{n_timesteps}}, critical =
{{critical}}, capacity = {{capacity}}, data = {{data}}, depth = {{depth}},
food_prob = {{food_prob}}, fish_abun = {{fish_abun}}, chl = {{chl}}, temp =
{{temp}}, k = {{k}}, lsurf = {{light_night}}, label = "Night Sim")

  dvmsimnight2 <- dvmsimnight %>% filter(Timestep == {{n_timesteps}})
  daphnia_night_lg <- dvmsimnight2$Depth[dvmsimnight2$Size == {{capacity}}]
  daphnia_night_sm <- dvmsimnight2$Depth[dvmsimnight2$Size == ({{critical}} +
5)]

  # cat("\n", "Generating Distributions")

  if (max(dvmsimday$Depth) >= max(dvmsimnight$Depth)){
    max_fig_depth <- max(dvmsimday$Depth)
  } else {
    max_fig_depth <- max(dvmsimnight$Depth)
  }
```

```
  curve_table_day <- singlecurvetable(dvmsim = dvmsimday, maxdepth =
max_fig_depth) # max(dvmsimday$Depth)
  curve_table_night <- singlecurvetable(dvmsim = dvmsimnight, maxdepth =
max_fig_depth)

  kstest_out <- {{kstest_out}}
  if (kstest_out == FALSE) {

    # cat("\n", "Generating Visuals")

    final_out <- dvmfigure(day = curve_table_day, night = curve_table_night,
day_lg = daphnia_day_lg, day_sm = daphnia_day_sm, night_lg =
daphnia_night_lg, night_sm = daphnia_night_sm)

    } else if (kstest_out == TRUE) {

    # cat("\n", "Generating Data Frame")
    curve_table_day <- curve_table_day %>% mutate(Time = "Noon")
    curve_table_night <- curve_table_night %>% mutate(Time = "Midnight")
    final_out <- rbind(curve_table_day, curve_table_night)

    } else {
    final_out <- "Error: kstest_out must be a boolean"
  }



  # cat("\n", "Simulation Complete")

  return(final_out)
}
```

Examples of the DVM Simulation

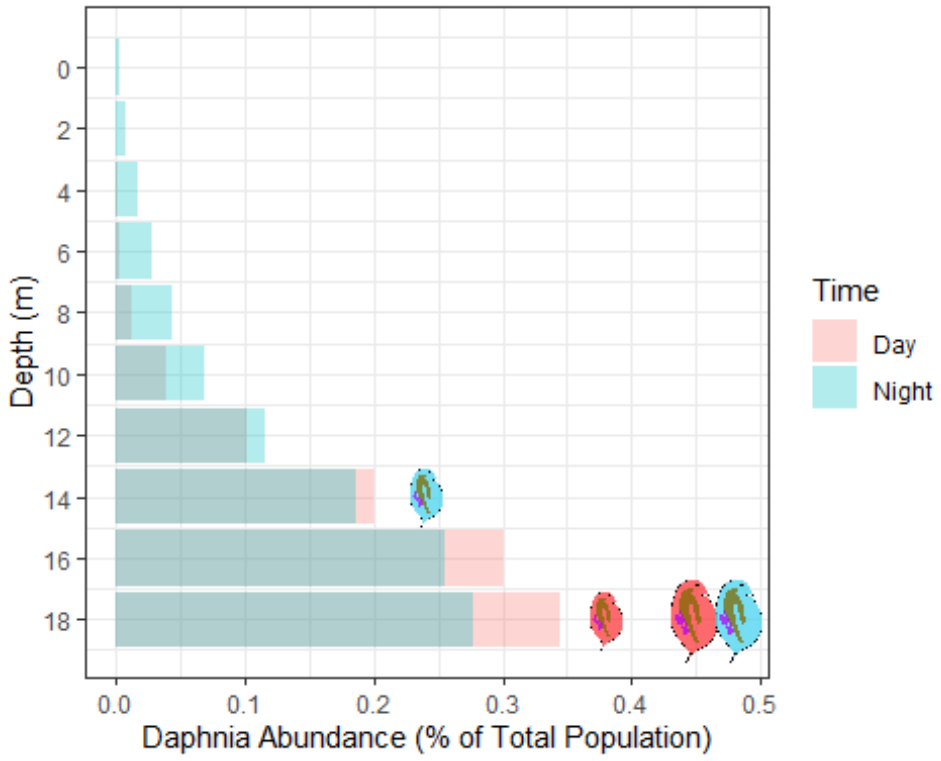
Trout Lake, May of 1992

```
trmay92_df <- data.frame("DEPTH" = c(0, 2, 4, 6, 8, 10, 12, 14 ,16, 18, 20,
22, 24, 26, 28, 30, 32),
                         "TEMP" = c(8.9, 8.8, 8.8, 8.7, 8.2, 7.9, 6.7, 6.6,
6.6, 6.4, 6, 5.6, 5.3, 5.2, 5.2, 5.2, 5.1),
                         "CHLORO" = c(10.6, 11, 12.1, 12.4, 12.8, 11.5, 9.8,
10.25, 10.6, 10.6, 10.7, 9, 8.3, 7.5, 6.9, 6, 5.5),
                         "PREDATORS" = c(6.31, 1.5, 0.9, 3.61, 8.57, 13.52,
25.7, 22.1, 21.2, 5.4, 1.4, 1.4, 1.8, 3, 4.5, 0, 0),
                         "FOOD_PROB" = c(0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5))
```

```
trmay_surflight <- luxtoPPFD(105253)

trmay92_plot <- DaphniaDVM(n_timesteps = 12, critical = 5, capacity = 60,
data = trmay92_df, depth = DEPTH, food_prob = FOOD_PROB, fish_abun =
PREDATORS, chl = CHLORO, temp = TEMP, k = 0.48, light_day = trmay_surflight,
light_night = 0.15, kstest_out = FALSE)
trmay92_plot
```

Sparkling Lake, 1991

```
Sparkling91_df <- data.frame("DEPTH" = c(0, 2, 4, 6, 8, 10, 12, 14 ,16, 18),
                             "TEMP" = c(21.3, 21.2, 21, 20.1, 12.4, 9.2, 7.5,
6.2, 5.8, 5.6),
                             "CHLORO" = c(2, 2, 4, 4, 4, 10, 6.5, 4.25, 3.5,
3.5),
                             "PREDATORS" = c(105, 105, 105, 45, 45, 91, 91, 32,
32, 1),
                             "FOOD_PROB" = c(0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
0.5, 0.5, 0.5))

Sparkling91_surflight <- 270
Sparkling91_k <- 0.316098166

DaphniaDVM(n_timesteps = 12, critical = 5, capacity = 60, data =
Sparkling91_df, depth = DEPTH, food_prob = FOOD_PROB, fish_abun = PREDATORS,
chl = CHLORO, temp = TEMP, k = 0.316098166, light_day = 270, light_night =
0.15, kstest_out = FALSE)
```


Depth (m)
0
2
4
6
8
10
12
14
16
18
0.0
0.1
0.2
0.3
0.4
0.5
Daphnia Abundance (% of Total Population)
Time
Day
Night