DEB.Model.1.0.1

Ingestion predictions for Biodeposition SW conditions

Emily Roberts

2023-12-02

I'm running Emilien's code and figuring out how to use it to predict assimilation under different environmental conditions.

Table of contents

0.0.1 DEB code developed Summer 2018

Code DEB 3 state variables Emilien Pousse

DEB with pCO2 effect

0.0.2 Load packages, access files and DEB parameters

```
dir_data <- "~/GitHub/EAD-ASEB-Ssolidissima-OA/projects/DEB/data/"
dir_data_biodep <- "~/GitHub/EAD-ASEB-Ssolidissima-OA/projects/Biodeposition/data/"
library(R.matlab)
```

```
R.matlab v3.7.0 (2022-08-25 21:52:34 UTC) successfully loaded. See ?R.matlab for help.
```

```
Attaching package: 'R.matlab'


The following objects are masked from 'package:base':

    getOption, isOpen
```

```r
filename <- paste(dir_data,sep="","results_Ensis_directus.mat")
ParaMat=readMat(filename )
ParaMat$par
```

```
, , 1

       [,1]
z      0.249552
F.m    62.69902
kap.X  0.8
kap.P  0.1
v      0.002167473
kap    0.9751298
kap.R  0.95
p.M    51.09083
p.T    0
k.J    0.002
E.G    2360.587
E.Hb   5.292778e-07
E.Hj   0.0002292468
E.Hp   51.05927
h.a    2.93161e-07
s.G    1e-04
T.A    8000
T.ref  293.15
del.M  0.159872
f      1
t.0    180
d.X    0.09
d.V    0.09
d.E    0.09
d.P    0.09
mu.X   525000
mu.V   5e+05
mu.E   550000
```

```
mu.P  480000
mu.C  0
mu.H  0
mu.O  0
mu.N  0
n.CX  1
n.HX  1.8
n.OX  0.5
n.NX  0.15
n.CV  1
n.HV  1.8
n.OV  0.5
n.NV  0.15
n.CE  1
n.HE  1.8
n.OE  0.5
n.NE  0.15
n.CP  1
n.HP  1.8
n.OP  0.5
n.NP  0.15
n.CC  1
n.HC  0
n.OC  2
n.NC  0
n.CH  0
n.HH  2
n.OH  1
n.NH  0
n.CO  0
n.HO  0
n.OO  2
n.NO  0
n.CN  0
n.HN  3
n.ON  0
n.NN  1
free  list,65
```

```
Param <- list()
```

0.0.2.0.1 Initialize *Spisula solidissima* parameters

Parameters from email correspondence 12/4/23

```
Param$RhoV=15600; #(it is a general number for bivalves)
Param$GammaL1=0.8
Param$GammaL2=0.8
Param$v= 0.011 #cm/d Energy conductance
Param$Kappa=0.9607
Param$KappaX <- 0.8 #K Digestion efficiency of food
Param$Pm=10.64 #13.48
Param$Eg=2360    #2361
Param$Shape=0.37    #0.445
Param$s_M= 17.8                 #7.49598
Param$Pxm= 9.08/.8      #30.6986/0.8
Param$Em=Param$Pxm*Param$KappaX/Param$v

#Arrhenius 5 parameter TPC
Param$Ta <- 9018 #K
Param$T1 <- 293.15 #K
Param$Tl <- 277.3 #K
Param$Th <- 296.5 #K
Param$Tal <- 15601 #K
Param$Tah <- 33775 #K

Param$Lp <- 3 #cm From Pousse et al. 2023
Param$dv <- 0.09
Param$RhoE <- 19600 #J/g
```

Parameters from conversation on GitHub on Jan 10, 2022

```
Param$pHl <- 1011 #uatm Lower pCO2 boundary for ingestion
Param$pHh <- 6778 #uatm Higher pCO2 boundary for ingestion

Param$MpHl <- 700 #uatm Lower pCO2 boundary for maintenance costs
Param$MpHh <- 1300 #uatm Higher pCO2 boundary for maintenance costs
```

0.0.2.0.2 Initialize *Spisula solidissima* pH parameters

```
Param$del_pH=0.01
```

0.0.2.0.3 Set RCP scenario info

```
#for (Sc in 1:5){
#  Sc <- 1
#  if(Sc==1){ScRCP=2.6}
#  if(Sc==2){ScRCP=4.5}
#  if(Sc==3){ScRCP=6}
#  if(Sc==4){ScRCP=8.5}
#if(Sc==5){ScRCP=10}

#For now, just pick a scenario to run.
ScRCP=2.6

# Or use validation
#ScRCP='Validation'
```

0.0.2.0.4 Chose the date when you want the model to start

This will set different temperature and pCO2 forcing variables

```
Oyster='now'# now; 2045; 2070; 2095
Zone="MAB" # MAB; MABmes; GB ### here you can chose if you want to use forcing variables f
#ScRCP=8.5#"Validation"#;4.5#2.6; 4.5; 6; 8.5; Validation
#ScRCP=2.6
```

0.0.3 Files for forcing variables

```
filename <- paste(dir_data,sep="","Temp_data_for_Emilien (copie).csv")
Rechauf=read.csv(filename,header=T,sep=",") # This is a file with the different projected
```

Set up RCP scenarios for environmental data

```
if(ScRCP==2.6){RCP=matrix(20,nrow=5,ncol=3)
RCP[,1]=c(2020,2040,2060,2080,2100)
RCP[,2]=c(412.385321101,439.143730887,441.28440367,430.581039755,420.948012232)
RCP[,3]=c(RCP[2,2]/RCP[1,2],RCP[3,2]/RCP[1,2],RCP[4,2]/RCP[1,2],RCP[5,2]/RCP[1,2],(RCP[5,2
Rechauf=Rechauf[,8]-Rechauf[145,8]
Rechauf[1128]=0.469
}
```

```
if(ScRCP==4.5){RCP=matrix(20,nrow=5,ncol=3)
RCP[,1]=c(2020,2040,2060,2080,2100)
RCP[,2]=c(409.174311927,461.620795107,507.645259939,530.122324159,537.614678899)
RCP[,3]=c(RCP[2,2]/RCP[1,2],RCP[3,2]/RCP[1,2],RCP[4,2]/RCP[1,2],RCP[5,2]/RCP[1,2],(RCP[5,2
Rechauf=Rechauf[,6]-Rechauf[145,6]
}

if(ScRCP==6){RCP=matrix(20,nrow=5,ncol=3)
RCP[,1]=c(2020,2040,2060,2080,2100)
RCP[,2]=c(409.174311927,450.917431193,509.785932722,593.272171254,669.266055046)
RCP[,3]=c(RCP[2,2]/RCP[1,2],RCP[3,2]/RCP[1,2],RCP[4,2]/RCP[1,2],RCP[5,2]/RCP[1,2],(RCP[5,2
Rechauf=Rechauf[,4]-Rechauf[145,4]
}

if(ScRCP==8.5){RCP=matrix(20,nrow=5,ncol=3)
RCP[,1]=c(2020,2040,2060,2080,2100)
RCP[,2]=c(414.525993884,489.449541284,602.905198777,758.103975535,934.709480122)
RCP[,3]=c(RCP[2,2]/RCP[1,2],RCP[3,2]/RCP[1,2],RCP[4,2]/RCP[1,2],RCP[5,2]/RCP[1,2],(RCP[5,2
Rechauf=Rechauf[,2]-Rechauf[145,2]
}
```

```
FuturCO2=c(1,RCP[,3])
Jours=c(1,20,40,60,80,100)
date2=seq(1:100)
resultat=approx(Jours, FuturCO2, xout = date2, method="linear", ties="ordered")$y
factor=1
for (n in 1:100){
  fac=rep(resultat[n],365)
  factor=c(factor,fac)
}

Rechauf=Rechauf[145:length(Rechauf)]
Rechauf=c(Rechauf,Rechauf[984]+(seq(0:216)-1)*(Rechauf[984]-Rechauf[925])/60)
Jours=c(1,c(15,46,75,106,136,167,197,228,259,289,319,350)+(sort(rep(seq(0:99),12))-1)*365)
date2=seq(1:36500)
Rechauf=approx(Jours, Rechauf, xout = date2, method="linear", ties="ordered")$y
Rechauf[36486:36500]=Rechauf[36485]+seq(1:15)*(Rechauf[36485]-Rechauf[36480])
```

Load MAB zone temp, chl, and pCO2 estimates

```
if(Zone=="MAB"){Temp=as.data.frame(read.csv2(paste(dir_data,sep="","MABtemp.csv"),dec='.',

# MAB temp
Temp=as.data.frame(read.csv2(paste(dir_data,sep="","MAB25yearsTemp.csv"),dec='.',sep=',',h
Temp=as.numeric(as.vector(Temp[2:length(Temp[,2]),2]))+273.15
Temp=c(Temp[150:length(Temp)],Temp[1:149])
Temp=(Temp[8767:9132])
Temp=c(Temp[220:365],Temp[1:219])

# MAB chl
Food=as.data.frame(read.csv2(paste(dir_data,sep="","MAB25ychldaily35.csv"),dec='.',sep=','
Food=as.numeric(as.vector(Food[2:length(Food[,2]),2]))
Food=c(Food[150:length(Food)],Food[1:149])
Food=Food[1:9125]

# MAB CO2
filename = paste(dir_data,sep="","MABpCO2_35y.csv")
MABpCO2=read.csv2(filename,sep=',')[,2]
MABpCO2=as.numeric(as.character(MABpCO2))

 FuturCO2=c(440,rep(MABpCO2[397:408]*1.15,100)+seq(1:1200)*0.22)
 Jours=c(1,c(15,46,75,106,136,167,197,228,259,289,319,350)+(sort(rep(seq(0:99),12))-1)*365
 date2=seq(1:36500)
 resultat=approx(Jours, FuturCO2, xout = date2, method="linear", ties="ordered")$y
 MABpCO2=resultat[1:365]
 Param$Xk=1.8

}
```

Load GB estimates

```
if(Zone=="GB"){
Temp=as.data.frame(read.csv2(paste(dir_data,sep="","GB25yTempal.csv"),dec='.',sep=',',head
Temp=as.numeric(as.vector(Temp[2:length(Temp[,2]),2]))+273.15
Temp=c(Temp[150:length(Temp)],Temp[1:149])
Temp=(Temp[8767:9132])
Temp=c(Temp[220:365],Temp[1:219])

Food=as.data.frame(read.csv2(paste(dir_data,sep="","GB25yChlnl.csv"),dec='.',sep=',',heade
Food=as.numeric(as.vector(Food[2:length(Food[,2]),2]))
Food=c(Food[150:length(Food)],Food[1:148])
Food=Food[1:9125]
```

```
filename = paste(dir_data,sep="","GBpCO2_35y.csv")
pCO2=read.csv2(filename,sep=',')[,2]
MABpCO2=as.numeric(as.character(pCO2))
FuturCO2=c(475.8827,rep(MABpCO2[397:408]*1.22,100)+seq(1:1200)*0.22)
Jours=c(1,c(15,46,75,106,136,167,197,228,259,289,319,350)+(sort(rep(seq(0:99),12))-1)*365)
date2=seq(1:36500)
resultat=approx(Jours, FuturCO2, xout = date2, method="linear", ties="ordered")$y
MABpCO2=resultat[1:365]
Param$Xk=1.4

 }
```

Add temperature increase differential to the current temperature in GB or MAB

```
x <- 1:365 #365 days in a year
y <- MABpCO2
lis <- loess(y ~ x, span= 0.25) #loess() fits a polynomial surface, used to interpolate I
  MABpCO2   <- predict(lis, x)

Surp=(factor[2:36501]-1)*max(MABpCO2[1:365])
FuturCO2=rep(MABpCO2,100)*factor[2:36501]
MABpCO2=rep(MABpCO2,100)+Surp

Temp=rep(c(Temp[150:365],Temp[1:149]),100)
Temp=Temp+Rechauf
```

0.0.4 Definition of the temporal vector

Define the temporal vector as a function of RCP scenario, location and model start date

```
if(Oyster=='now'){
  date_start<-"25/12/2020"# Select a start date format = dd/mm/yyyy
  date_end<-"25/12/2045"}# Select a end date format = dd/mm/yyyy}
if(Oyster=='2045'){
  date_start<-"25/12/2045"# Select a start date format = dd/mm/yyyy
  date_end<-"25/12/2070"}# Select a end date format = dd/mm/yyyy}
if(Oyster=='2070'){
  date_start<-"25/12/2070"# Select a start date format = dd/mm/yyyy
  date_end<-"25/12/2095"}# Select a end date format = dd/mm/yyyy}
if(Oyster=='2095'){
```

```
  date_start<-"25/12/2095"# Select a start date format = dd/mm/yyyy
  date_end<-"25/12/2120"}# Select a end date format = dd/mm/yyyy}
```

I'm setting ScRCP now to validation, but earlier in the code it was set to 2.5

```
ScRCP = "Validation"
if(ScRCP=='Validation'){
  Oyster="nul"
  date_start<-"01/01/1994"# Select a start date format = dd/mm/yyyy
  date_end<-"01/01/2019"
  MABpCO2=rep(1,9125)
  pH=MABpCO2

#GB zone:
  if(Zone=="GB"){

  #Temperature
  Temp=as.data.frame(read.csv2(paste(dir_data,sep="",
              "GB25yTempal.csv"),dec='.',sep=',',header=F))
  Temp=as.numeric(as.vector(Temp[2:length(Temp[,2]),2]))+273.15 #Kelvin
  Temp=c(Temp[150:length(Temp)],Temp[1:149])

  #Food
  Food=as.data.frame(read.csv2(paste(dir_data,sep="",
              "GB25yChlnl.csv"),dec='.',sep=',',header=F))
  Food=as.numeric(as.vector(Food[2:length(Food[,2]),2]))
  Food=c(Food[150:length(Food)],Food[1:149])
  }

#MAB zone
  if(Zone=="MAB"){

  #Temperature
    Temp=as.data.frame(read.csv2(paste(dir_data,sep="",
              "MAB25yearsTemp.csv"),dec='.',sep=',',header=F))
    Temp=as.numeric(as.vector(Temp[2:length(Temp[,2]),2]))+273.15
    Temp=c(Temp[150:length(Temp)],Temp[1:149])

  #Food
    Food=as.data.frame(read.csv2(paste(dir_data,sep="",
              "MAB25ychldaily35.csv"),dec='.',sep=',',header=F))
```

```
    Food=as.numeric(as.vector(Food[2:length(Food[,2]),2]))
    Food=c(Food[150:length(Food)],Food[1:149])
  }
  Food=Food[1:9125]
  Temp=Temp[1:9125]

  }
```

Make the temporal vector

```
date_start <- as.numeric(strptime(date_start,tz="UTC",format="%d/%m/%Y"))
date_end <- as.numeric(strptime(date_end,tz="UTC",format="%d/%m/%Y"))

dt   <- 86400 # in secondes (3600s = 1h / 86400s = 1j / 28800 = 8h )
pdt  <- dt / 86400 # time step in day (= unité de paramètres du DEB)
Xnum <- seq(from = date_start,date_end,dt)#vector with each time step (sec)
Xtps <- as.POSIXct(Xnum,origin='1970-1-1',tz="UTC")#vector with each time step (normal dat
ndt  <- length(Xnum)# Number of days
ndt=9125
```

```
if(ScRCP==2.6){Temp2.6=Temp
Rechauf2.6=Rechauf
if(Zone=="GB"){GpCO22.6=MABpCO2}
if(Zone=="MAB"){MpCO22.6=MABpCO2}}
if(ScRCP==4.5){Temp4.5=Temp
Rechauf4.5=Rechauf
if(Zone=="GB"){GpCO24.5=MABpCO2}
if(Zone=="MAB"){MpCO24.5=MABpCO2}}
if(ScRCP==6){Temp6=Temp
Rechauf6=Rechauf
if(Zone=="GB"){GpCO26=MABpCO2}
if(Zone=="MAB"){MpCO26=MABpCO2}}
if(ScRCP==8.5){Temp8.5=Temp
Rechauf8.5=Rechauf
if(Zone=="GB"){GpCO28.5=MABpCO2}
if(Zone=="MAB"){MpCO28.5=MABpCO2}}
if(ScRCP==10){Temp10=Temp
Rechauf10=Rechauf
if(Zone=="GB"){GpCO210=MABpCO2}
if(Zone=="MAB"){MpCO210=MABpCO2}}
```

```
if(Oyster=='now'){
Temp=Temp[1:9125]
pH=MABpCO2[1:9125]}

if(Oyster=='2045'){
  Temp=Temp[9126:18250]
  pH=MABpCO2[9126:18250]}

if(Oyster=='2070'){
  Temp=Temp[18251:27375]
  pH=MABpCO2[18251:27375]}

if(Oyster=='2095'){
  Temp=Temp[27376:36500]
  pH=MABpCO2[27376:36500]}
```

0.0.5 Verification of the forcing variables

```
# Temp=rep(25,25)
# Food = rep(1, 25)
#Check if all the forcing variables have the same length or have missing values
if (length(Temp) <= 1 || length(Temp) != length(Food)) {
  stop("Arguments Temp / Food / T_im have different lengths: ",length(Temp)," ; ",length(F
}
if (TRUE %in% is.na(Temp) || TRUE %in% is.na(Food)) {
  stop(" Arguments Temp, Food and T_im must not have missing values.")
}# Just to confirm that forcing variables are correct
```

0.0.6 Get initial values

```
file_init <- paste(dir_data,sep="","USBiominit.csv")

phys=read.csv2(file_init,sep=",",dec=".")#Load file with initial observation
phys$DFM=rep(0.005,2)
phys$LEN=rep(1,2)
Init=c()
id=seq(1:2)
Eri=c()
```

```
Ei=c()
Vi=c()
DFMi=c()
```

```
for (p in 1:2){
  e=0.2#0.57 #Density of reserve: 1=full, 0=empty, rule: 0.2 winter, 0.6 summer 0.4 fall/s
  Vi[p]   = (phys$LEN[p] * Param$Shape)^3
  Ei[p]=Param$RhoE*(8.18e-6-(Vi[p]*0.09))

  if(Vi[p]>(Param$Lp*Param$Shape)^3){Eri[p]= ((phys$DFM[p] - (Vi[p] *Param$dv)) * Param$Rh
  }else{Eri[p]=0
  Ei[p]= max(c(0.000001,(phys$DFM[p]-Vi[p]*Param$dv)*Param$RhoE))}

  if(Eri[p]<=0){Eri[p]=0}

  DFMi[p] = ((Ei[p] + Eri[p])/Param$RhoE + Param$dv*Vi[p])
}
Init=as.data.frame(cbind(id,Vi,Ei,Eri,DFMi))#,Egoi))
nbAnim <- 2#max(Init$id)
```

Question - what is nbAnim? ID of the animal? ID of simulation?

0.0.7 DEB model simulations

Creation of the results database

Number of returned parameters for fCalcDEB_4VE = 21

```
Result<-array(NA, dim=c(ndt, 20, nbAnim)) # Make a 3D matrix [time, output value, individu
colnames(Result)<-c("jour","E","V","Er","DFM","L","e","PM","Pc1","f","Pg","Pa","Temp",
                    "cT","Xnum","Pl1","Pl2","Pm1","Pj","Pr")#name the parameters
```

Launch DEB for each individual

```
n_iter <- length(Temp)

ErBalance=matrix(0,ncol=n_iter/365,nrow=nbAnim)
```

Here modifying the code for simplicity, since nbAnim is set to 2

for (i in 1:nbAnim) {}

```r
i <- 1

## Initialisation du DEB
  ########################
  V    = Init$Vi[i]
  E    = Init$Ei[i]
  Er   = Init$Eri[i]

  L=Init$Vi[i]^(1/3)/Param$Shape

  ## Initialisation des sorties
  #############################
  # nombre d'itération
  n_iter = length(Temp)
  # Création de vecteurs contenant les variables d'état
  vecteur_E    = vector("numeric",length=n_iter)
  vecteur_V    = vector("numeric",length=n_iter)
  vecteur_Er   = vector("numeric",length=n_iter)
  vecteur_temps = vector("numeric",length=n_iter)
  vecteur_f    = Food/(Food+Param$Xk)

  # Initialization of the state variables
  vecteur_E[1]=E[1]
  vecteur_V[1]=V[1]
  vecteur_Er[1]=Er[1]

  vecteur_temps[1]=0    ####################Utile?
  Balance=0

  vecteur_Pa    = vector("numeric",length=n_iter)
  vecteur_Px    = vector("numeric",length=n_iter)
  vecteur_PM    = vector("numeric",length=n_iter)
  vecteur_Pc1   = vector("numeric",length=n_iter)
  vecteur_Pg    = vector("numeric",length=n_iter)
  vecteur_Pr    = vector("numeric",length=n_iter)
  vecteur_Pj    = vector("numeric",length=n_iter)
  vecteur_Pl1   = vector("numeric",length=n_iter)
  vecteur_Pl2   = vector("numeric",length=n_iter)
  vecteur_Pm1   = vector("numeric",length=n_iter)
  vecteur_Em   = vector("numeric",length=n_iter)#################### utile?
```

0.0.8 Temperature correction

```
cT<-exp((Param$Ta /Param$T1)-(Param$Ta /Temp)) *
  (1+exp((Param$Tal/Param$T1)-(Param$Tal/Param$Tl))+exp((Param$Tah/Param$Th)-(Param$Tah/Pa
  (1+exp((Param$Tal/Temp)-(Param$Tal/Param$Tl))+exp((Param$Tah/Param$Th)-(Param$Tah/Temp
```

0.0.9

```
  ## Start of the calculation loop
  #################################
  #for (n in 1:n_iter){

#Molly's edit: Just grow to 36mm
for (n in 1:481){

    #### Correction of biological functions impacted by Temperature
    Pxm =  Param$Pxm* cT[n]*Param$s_M
    Pm=Param$Pm* cT[n]
    v = Param$v* cT[n] * Param$s_M


    # Compound parameters corrected
    Pam = Pxm * Param$KappaX
    Em  = Pam/v
    KappaG = (Param$dv*Param$RhoV)/Param$Eg
    Vp = (Param$Shape*Param$Lp)^3
    if(pH[n]<=1011){cpH=1}else{cpH=(Param$pHh-pH[n])/(Param$pHh-Param$pHl)}


    Px  <- Pxm * vecteur_f[n] * V^(2/3)  #Ingestion

    Pa  <- Pxm * vecteur_f[n] * V^(2/3) * Param$KappaX *cpH


    Pc1 <- E * (((Param$Eg * (v/V^(1/3))) + Pm) / (Param$Eg + (Param$Kappa * (E/V))))#Mobi


    PM  <- (Pm * V)+(Param$del_pH*(min(max(pH[n]-Param$MpHl,0),Param$MpHh-Param$MpHl)*V^(2

    Pm1 <- min(PM,Param$Kappa * Pc1) #maintenance costs paid
```

```r
Pg  <- Param$Kappa * Pc1 - Pm1 #Growth

Pj  <- min(c(V,Vp)) * Pm * ((1 - Param$Kappa) / Param$Kappa) #Maturation

Pr   <- max(c((1 - Param$Kappa)*Pc1 - Pj,0)) # Reproduction




#SANS GONADE
# In case of long starvation (i.e. maintenance costs cannot be paid):
if(Param$Kappa*Pc1<=PM){# Maintenance costs cannot be paid
  E  = max(0, E   + (Pa - Pc1) * pdt)
  Pl2=1/Param$GammaL2*(PM-(Param$Kappa*Pc1))
  Pr=0
  if(V>=Vp&Er>=Pl2){
    Er=max(0, Er  - Pr-Pl2 * pdt)
    V  = max(0, V)
  }else{
    Pl1=1/Param$GammaL1*(PM-Param$Kappa*Pc1)
    Er=0
    V  = max(0, V   + (-Pl1)/(Param$RhoV*Param$dv) * pdt)
    Pl2=0
  }

}else{
  Pl1=0
  Pl2=0
  # State variables calculation
  E  = max(0, E   + (Pa - Pc1) * pdt)
  V  = max(0, V   + (Pg/Param$Eg-Pl1) * pdt)
  Er = max(0, Er  + (Pr - Pl2) * pdt)
  if(V<Vp){Er=0}}

## More outputs
###############
DFM = (E+Er)/Param$RhoE + Param$dv*V #+ Ego*Param$dgo/Param$EGgo# Dry flesh mass
if(V^(1/3)/Param$Shape>=L){
  L=V^(1/3)/Param$Shape
}
e = (E/V)/Em
```

```
    # State variables allocation
    vecteur_E[n]   = E
    vecteur_V[n]   = V
    vecteur_Er[n]  = Er

    # Output variables
    vecteur_temps[n]= n*pdt
    vecteur_Pa[n]   = Pa
    vecteur_Px[n]   = Px
    vecteur_Pr[n]   = Pr
    vecteur_Pj[n]   = Pj
    vecteur_PM[n]   = PM
    vecteur_Pc1[n]  = Pc1
    vecteur_Pg[n]   = Pg

    vecteur_Pl1[n]  =Pl1
    vecteur_Pl2[n]  =Pl2
    vecteur_Pm1[n]  =Pm1

    Result[n,1:20,i]=c(n,E,V,Er,DFM,L,e,PM,Pc1,vecteur_f[n],Pg,Pa,Temp[n],cT[n],Xnum[n],Pl
    #print(paste("/ day",n))
    }
  print(paste("/ ind.",i)) #Individual 1
```

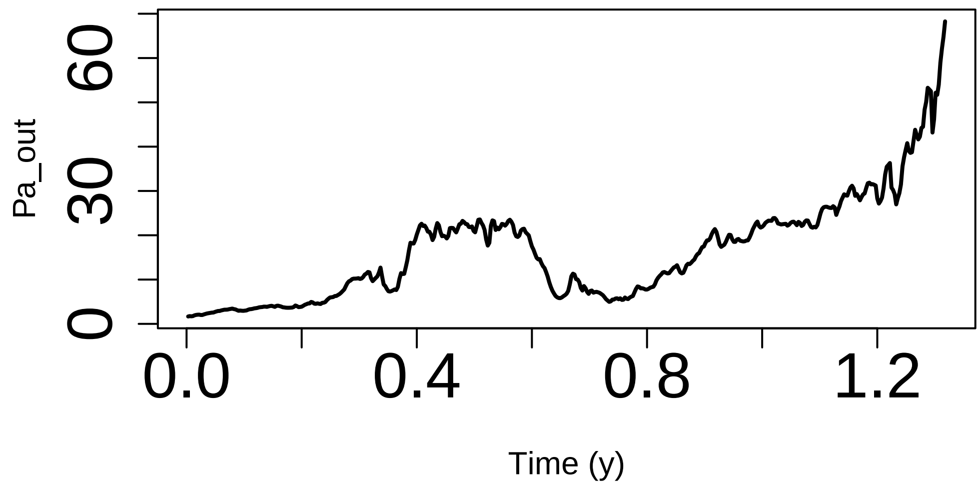
```
[1] "/ ind. 1"
```

Molly's plots and calculations

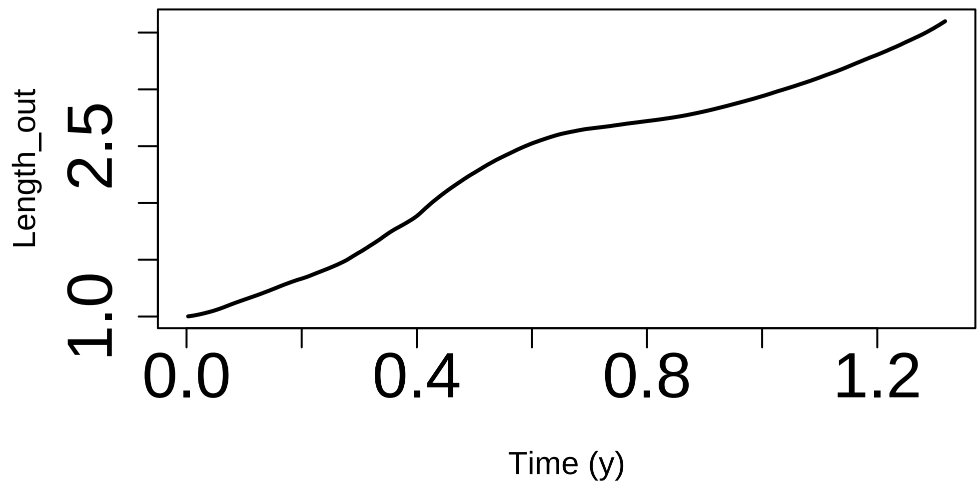
```
Result[1,,1]
```

```
        jour            E            V           Er          DFM            L
1.000000e+00 9.645157e+00 5.089000e-02 0.000000e+00 5.072200e-03 1.001557e+00
           e           PM          Pc1            f           Pg           Pa
2.296062e-01 8.357358e-02 6.692022e-01 4.856340e-01 5.593290e-01 1.666252e+00
        Temp           cT         Xnum          Pl1          Pl2          Pm1
2.787782e+02 1.550680e-01 7.573824e+08 0.000000e+00 0.000000e+00 8.357358e-02
          Pj           Pr
3.418800e-03 2.288085e-02
```

```r
Length_out <- Result[,6,1][1:481]
Pa_out <- Result[,12,][1:481]
Day <-  Result[,1,][1:481]

plot(Day/365,Pa_out,type='l',lwd=2,
         #ylim=c(0,20),xlim=c(0,25),
         #ylab=ylabel,cex.lab=2,
         xlab="Time (y)",cex.axis=2)
```




```r
    #,xlim=c(0,length(Resultmean)/365+0.25))
  #text(0,19.5,"(B)")

plot(Day/365,Length_out,type='l',lwd=2,
         #ylim=c(0,20),xlim=c(0,25),
         #ylab=ylabel,cex.lab=2,
         xlab="Time (y)",cex.axis=2)
```




```
    #,xlim=c(0,length(Resultmean)/365+0.25))
  #text(0,19.5,"(B)")
```

Now for day 482… let's see how this is affected by pH, temp, and food.

First let's create a dataframe that includes all average temperatures. This is "df_summary" from Biodeposition_SW_Summary.qmd

```
SW_cond <- read.csv(paste(dir_data_biodep,"Output/SW_cond_averages.csv", sep = ""), string
```

```
Temp_vec <- SW_cond$Temp+273

cT_vec<-exp((Param$Ta /Param$T1)-(Param$Ta /Temp_vec)) *
  (1+exp((Param$Tal/Param$T1)-(Param$Tal/Param$Tl))+exp((Param$Tah/Param$Th)-(Param$Tah/Pa
  (1+exp((Param$Tal/Temp_vec)-(Param$Tal/Param$Tl))+exp((Param$Tah/Param$Th)-(Param$Tah/Te

pH_vec <- SW_cond$pCO2 #Note these are all below the pCO2 threshold

# Calc correction factor for pH on feeding:
# (Converted to vector calculation rather than for-loop used below. This should be faster.
cpH_vec <- seq(from = 1, t=6, length.out = length(pH_vec))
cpH_vec[pH_vec<=1011] <- 1
```

```
cpH_vec[pH_vec>1011] <- (Param$pHh-pH_vec[pH_vec>1011])/(Param$pHh-Param$pHl)
```

Estimate food based on chlorophyll from biodeposition

```
  vecteur_f   = Food/(Food+Param$Xk)
  Bio_food_Eel_Sep <- 1.109
  vecteur_f_Eel_Sep   <-
    Bio_food_Eel_Sep/(Bio_food_Eel_Sep+Param$Xk)

  vecteur_f_point <- vecteur_f_Eel_Sep
```

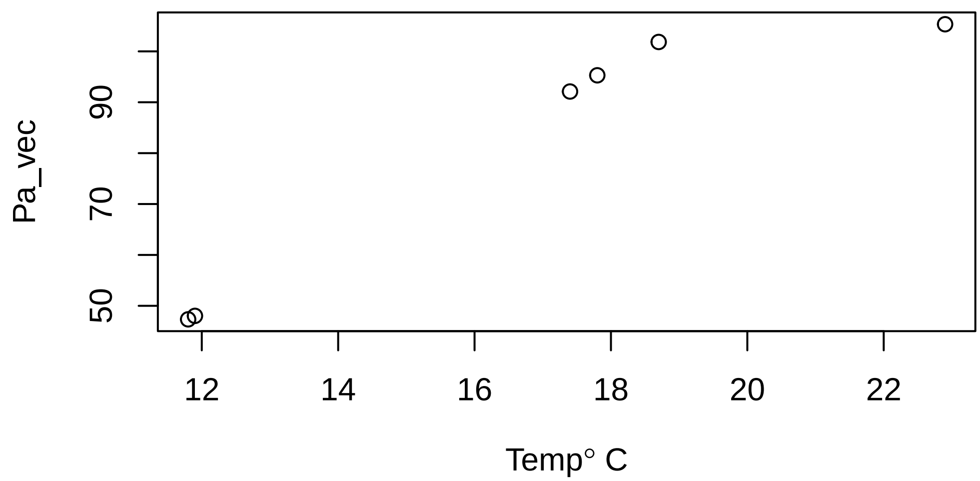
Let's calculate Pa as a function of temperature. I commented out the later parts because there was an issue with each temp output being used to increment length/everything… Just focusing on feeding Pa or Px, depending on what is interesting.

```
n<- 482
    #### Correction of biological functions impacted by Temperature
Pa_vec <- rep(NA, length.out = length(cT_vec))

for (j in 1:length(cT_vec)){

  #j<-1
#New plots as a function of temperature and pCO2
    Pxm =  Param$Pxm* cT_vec[j]*Param$s_M
    Pm=Param$Pm* cT_vec[j]
    v = Param$v* cT_vec[j] * Param$s_M


    # Compound parameters corrected
    Pam = Pxm * Param$KappaX
    Em  = Pam/v
    KappaG = (Param$dv*Param$RhoV)/Param$Eg
    Vp = (Param$Shape*Param$Lp)^3

        # This corrects for pCO2's 1011 and above.


    Px  <- Pxm * vecteur_f_point * V^(2/3)  #Ingestion

    Pa  <- Pxm * vecteur_f_point * V^(2/3) * Param$KappaX *cpH #KappaX is the digesion eff
```

```
    Pa_vec[j]  <- Pxm * vecteur_f_point * V^(2/3) * Param$KappaX *cpH_vec[j]
}
```

Surface specific assimilation (I think) as a function of temperature

```
plot(Temp_vec-273,Pa_vec,type='p',
         #ylim=c(0,20),xlim=c(0,25),
         #ylab=ylabel,cex.lab=2,
         xlab=expression(paste("Temp",degree," C")),cex.axis=1)
```




```
     #,xlim=c(0,length(Resultmean)/365+0.25))
   #text(0,19.5,"(B)")
```

```
SW_cond$Temp_coeff_feed <- cT_vec
SW_cond$pCO2_coeff_feed <- cpH
SW_cond$Pred_ingestion <- Pa_vec
write.csv(SW_cond, paste(dir_data_biodep,"Output/Ingestion_pred_from_DEB.csv", sep = ""))
```