Urosalpinx growth rate (length)

Andrew Villeneuve

2/25/2020

# The Study

For this part of the study, we measured growth rates of juvenile snails. We measured snails within two days of them hatching, and grew them in tea strainers separated by population for 24 days. Nine replicates per population were distributed in six temperatures, with three groups of three subreplicates in each temperature/population treatment. The end length was recorded, and we substracted the starting length from the beginning length to get the growth rate (over 24 days).

# Metadata

* code
  + Unique code for each indiviudal snail, corresponding to population, temperature treatment. First digit = temperature (1=16,2=20,3=24,4=26, 5=28, 6=30), second digit = site (1=Willapa, 2=Humboldt, 3=Great Bay, 4=Woods Hole, 5=Oyster, 6=Beaufort, 7=Folly Beach, 8=Skidaway), third digit = tupperware bin number (1-3), fourth digit = snail replicate (1-3)
* pop
  + Source population of each snail. See data table below for list of site abbreivations with site.
* temp
  + Common garden temperature the snails were raised in for 24 days. Degrees C
* hatch
  + hatch date of each snail from it’s egg case (m/dd/yyyy)
* exp.date
  + Date on which hatchling snails were placed in the common garden experiment. Not more then 2 days from the hatch date. (m/dd/yyyy)
* grow.date
  + End date where growth measurements were taken. 24 days after exp.date, therefore no more then 26 days post hatch (m/dd/yyyy)
* alive
  + Tracks if snails survived the experient. m marks missing, n marks no, y marks yes
* rem.oysters
  + Was there a surplus of food at the end of the experiment? n marks no, y marks yes
* cal.length.start
  + caliper length of hatchlings upon entering the experment. We took photos of snails before entering snails into the experiment, and then used ImageJ to extract snail sizes. Size in mm
* cal.length.end
  + caliper length of hatchling at the end of the experiemnt. We took caliper measurements of the snails, as well as verifying the measurements using a subset of photographs in ImageJ. Size in mm
* wt
  + End weight of snail. Note that no initial starting weight was recorded. Weight in g.
* ran.out
  + Did the snail ever run out of food during the consumption experiment? 1 for yes, 0 for no
* bin
  + Bin number, controls subreplication. The third digit of the code.
* oce
  + Ocean (Atlantic or Pacific)

## Broken Stick regression

It’s hard to compare TPC against one another. One method we’ve settled on is the use of broken stick regression to allow us to quantify the shape of the reaction as well as the thermael optima (x) and the maximal trait performance (y).

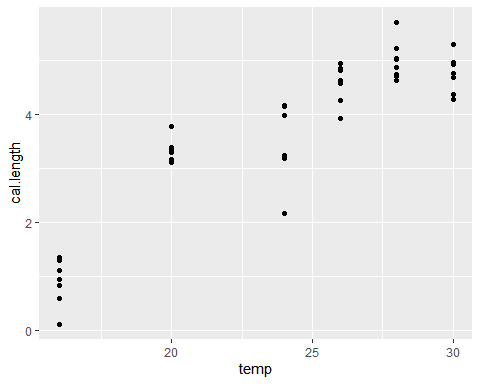
There are two methods to go about this. First is to use the segmented package, a relatively straightforward way to create piecewise regressions.

I will compare the methods with one population that has been “problematic” in regards with the segmented package.

# Segmented

Here is a great resource for getting started with the segmented package: <https://www.r-bloggers.com/r-for-ecologists-putting-together-a-piecewise-regression/>

#focusing on growth from one population, Great Bay. Filter, and plot raw data  
gb<-filter(growth.alive,pop=="Great Bay")  
ggplot(gb,aes(x=temp,y=cal.length))+geom\_point()



#create our model of caliper length over temperature  
m.gb<-glm(cal.length~temp,gb,family="gaussian")

What’s going on here? I am telling the segmented package to create a broken stick model.

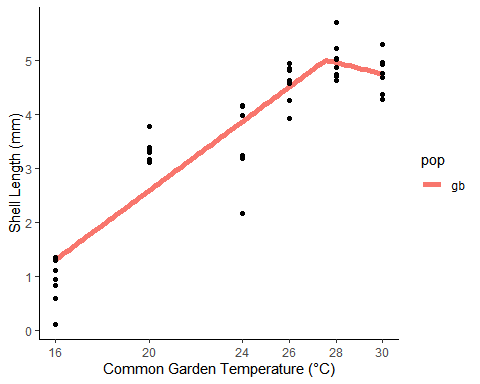
* m.gb - the linear model object
* seg.Z - equation with no response variable, indicating the continuous predictor over which the breakpoint is estimated to lie
* psi - starting predictor value for the package to start looking for the breakpoints. Can be a vector in the case of multiple breakpoints. Choose a psi value close to your eyeballed breakpoint
* n.boot - number of bootstrap iterations. I chose a high number here because of the possible bimodal breakpoint in this data.
* n.psi - not shown, but default at 1. Sets the number of expected breakpoints, ie can be multiple.

seg.gb<-segmented(m.gb,seg.Z = ~temp, psi=24,n.boot=5000)  
summary(seg.gb)

##   
## \*\*\*Regression Model with Segmented Relationship(s)\*\*\*  
##   
## Call:   
## segmented.glm(obj = m.gb, seg.Z = ~temp, psi = 24, n.boot = 5000)  
##   
## Estimated Break-Point(s):  
## Est. St.Err  
## psi1.temp 27.562 0.751  
##   
## Meaningful coefficients of the linear terms:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -3.80488 0.62864 -6.053 3.34e-07 \*\*\*  
## temp 0.31996 0.02876 11.125 4.24e-14 \*\*\*  
## U1.temp -0.42427 0.15154 -2.800 NA   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
## (Dispersion parameter for gaussian family taken to be 0.3486455)  
##   
## Null deviance: 91.474 on 45 degrees of freedom  
## Residual deviance: 14.643 on 42 degrees of freedom  
## AIC: 87.887  
##   
## Convergence attained in 1 iter. (rel. change 0)

great, we have our segmented model and we can see that we have a signficant deviation from 0 for slopes. Note that this does not tell us if the presence of a breakpoint is signficant. Now, we should plot the segmented model.

xmin<-min(growth.alive$temp,na.rm=T)  
xmax<-max(growth.alive$temp,na.rm=T)  
predicted.gb<-data.frame(temp=seq(xmin,xmax,length.out=100))  
predicted.gb$cal.length<-predict(seg.gb,predicted.gb)  
predicted.gb$pop<-"gb"  
predicted.gb$oce<-"a"  
  
ggplot(predicted.gb,aes(x=temp,y=cal.length))+geom\_line(aes(group=pop,x=temp,y=cal.length,color=pop,linetype=pop,size=pop))+  
 ylab("Shell Length (mm)")+xlab("Common Garden Temperature (°C)")+theme\_classic()+  
 scale\_x\_continuous(breaks=c(16,20,24,26,28,30))+geom\_point(data=gb,aes(x=temp,y=cal.length))+scale\_y\_continuous(breaks=c(0,1,2,3,4,5))



#extract breakpoint points  
brkpts\_seg<-data.frame(matrix(,nrow=1,ncol=6))  
colnames(brkpts\_seg)<-c("pop","brkptx","brkpty","brkptx\_se","se\_lower","se\_higher")  
brkpts\_seg[1,1]<-"gb\_segmented"  
brkpts\_seg[1,2]<-seg.gb$psi[[2]]  
brkpts\_seg[1,3]<-(seg.gb$psi[[2]]\*coef(seg.gb)[[2]])+(coef(seg.gb)[[1]])  
brkpts\_seg[1,4]<-seg.gb$psi[[3]]  
brkpts\_seg[1,5]<-brkpts\_seg[1,2]-brkpts\_seg[1,4]  
brkpts\_seg[1,6]<-brkpts\_seg[1,2]+brkpts\_seg[1,4]  
  
brkpts\_seg

## pop brkptx brkpty brkptx\_se se\_lower se\_higher  
## 1 gb\_segmented 27.56216 5.014004 0.7513959 26.81076 28.31355

Depending on the result of the segmented model, we either see a breakpoint at the point I want around 27 or earlier at a ‘false’ breakpoint. I can rerun and get the desired result back. Two issues with this: 1) It’s not dependable enough. I want to set the limits of x. 2) I can get either the standard error or the confidence interval from the breakpoint, but I want to get the variance in y so that I can add error bars to when I extract breakpoint x and y and plot by latitude.

# mcp package

What are the benefits of the mcp package? It gives a clearer picture into how the machine iterates and finds the best breakpoints, and is great for datasets like this one where you might need to control where the package looks for a breakpoint. It is much slower than segmented, particularly with large datasets. See below links for tips how to speed up.

Two good resources for mcp (one of which is my query on stackoverflow) <https://stackoverflow.com/questions/60419033/segmented-package-breakpoints-are-variable-and-finding-standard-errors-on-breakp> <https://lindeloev.github.io/mcp/>

The method for creating a segmented model in mcp is a little different than segmented. First, you have to define (create) the model.

#create global model  
model\_mcp=list(cal.length~1+temp, #line with intercept  
 ~0+temp) #Joined slope  
  
#use mcp to create model fit for each population  
  
#GB  
fitgb=mcp::mcp(model\_mcp,data=gb,iter=10000,cores=3)

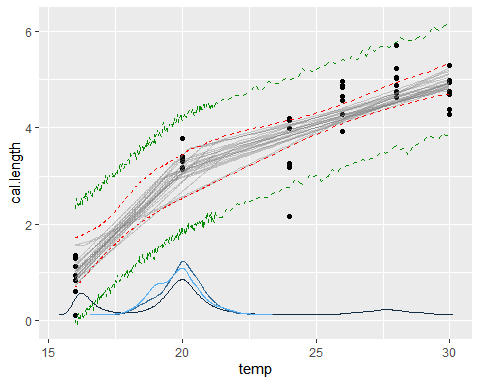
## Parallel sampling (`cores` > 1) sometimes err on R versions below 3.6.1. You have R version 3.5.1 (2018-07-02). Consider upgrading if it fails or hangs.

## Parallel sampling in progress...

## Finished sampling in 10.7 seconds

Here, I’ve made my fit for the great bay data over 10000 iterations. Cores=3 is a recommended set up . Let’s plot.

plot(fitgb, q\_fit = T, q\_predict = T)



The gray lines here are 25 random samples from the posterior, ie 25 random model fits from our iterations. Red lines are our fitted intervals, green are our prediction intervals. The blue lines show the frequency of breakpoint occurance over our iterations. What we can see here are two potential breakpoints over the common garden temperatures. This explains why sometimes segmented choses to make the breakpoint occur around 20C instead of 27.5C, which is the “ecologically correct” point we are looking for. The nice thing about mcp is that it allows the user much more control over where the package looks for the breakpoint over your predictor variable.

Here, we assign some priors by limiting the iterative breakpoint searching between 22C and 30C.

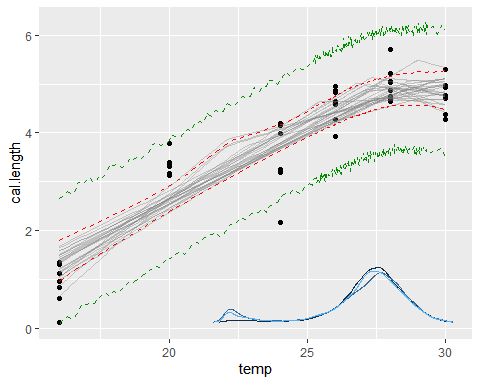
##bimodal. Let's target the "real" breakpoint by assigning priors  
fitgb2=mcp::mcp(model\_mcp,data=gb,prior=list(cp\_1="dunif(22,30)"),iter=10000,cores=3)

## Parallel sampling (`cores` > 1) sometimes err on R versions below 3.6.1. You have R version 3.5.1 (2018-07-02). Consider upgrading if it fails or hangs.

## Parallel sampling in progress...

## Finished sampling in 2.6 seconds

plot(fitgb2, q\_fit = T, q\_predict = T)



Success! This cleans up the breakpoint analysis quite a bit. Now, let’s extract values from this mcp method and compare to the segmented regression.

sgb<-summary(fitgb2)

## Family: gaussian(link = 'identity')  
## Iterations: 30000 from 3 chains.  
## Segments:  
## 1: cal.length ~ 1 + temp  
## 2: cal.length ~ 1 ~ 0 + temp  
##   
## Population-level parameters:  
## name mean lower upper Rhat n.eff  
## cp\_1 27.152 22.72 29.60 1.1 195  
## int\_1 -3.614 -5.08 -2.01 1.1 110  
## sigma\_1 0.607 0.48 0.74 1.0 8195  
## temp\_1 0.313 0.24 0.38 1.1 95  
## temp\_2 -0.017 -0.24 0.20 1.0 2184

#breakpoint\_x  
sgb[1,2]

## [1] 27.1523

#intercept  
sgb[2,2]

## [1] -3.613863

#segment\_1 slope  
sgb[4,2]

## [1] 0.312518

#segment\_2 slope  
sgb[5,2]

## [1] -0.01743018

#breakpoint\_y  
sgb[1,2]\*sgb[4,2]+sgb[2,2]

## [1] 4.87172

brkpts\_seg[2,1]<-"gb\_mcp"  
brkpts\_seg[2,2]<-sgb[1,2]  
brkpts\_seg[2,3]<-sgb[1,2]\*sgb[4,2]+sgb[2,2]  
brkpts\_seg[2,4]<-"NA"  
brkpts\_seg[2,5]<-sgb[1,3]  
brkpts\_seg[2,6]<-sgb[1,4]  
  
brkpts\_seg

## pop brkptx brkpty brkptx\_se se\_lower se\_higher  
## 1 gb\_segmented 27.56216 5.014004 0.751395868182744 26.81076 28.31355  
## 2 gb\_mcp 27.15230 4.871720 NA 22.72377 29.60490

Looks pretty good between the two methods, with mcp giving larger SE range than segmented.

So, what model to chose? segmented is very intuitive, and is much easier to predict values from and plot (as far as I can tell). I used segmented to extract my breakpoints and plot my fitted segmented models, but I used mcp to troubleshoot data as needed since there is much more control for the user.

Resource for choosing pieceise regression packages: <https://lindeloev.github.io/mcp/articles/packages.html>