growthTools vignette

Colin T. Kremer

2018-10-17

Calculate exponential population growth rates using growthTools

Getting Started

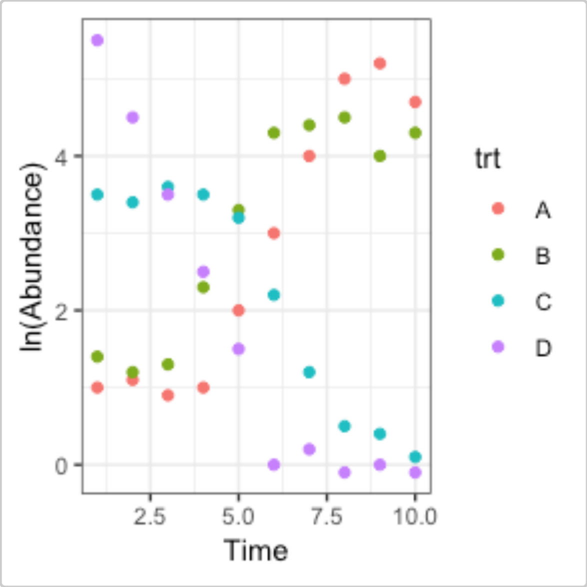
Load essential packages, including `growthTools`.

```
# To direct vignette code to use the growthTools package during development, calling
devtools::load_all() is suggested. When ready for release, use the library command.
# http://stackoverflow.com/questions/35727645/devtools-build-vignette-cant-find-functions

devtools::load_all()
#library(growthTools)

library(ggplot2)
library(dplyr)
library(tidyr)
library(mleTools)


# Construct example data set:
sdat<-data.frame(trt=c(rep('A',10),rep('B',10),rep('C',10),rep('D',10)),dtime=rep(seq(1,10),4
0.1,0,-0.1)))
```

Basic example: working with a single time series

We can take a quick look at the abundance time series created in this mock data set:

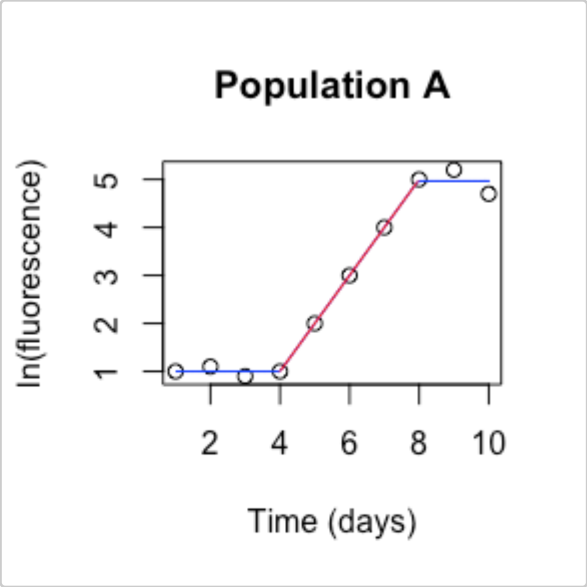
```
ggplot(sdat,aes(x=dtime,y=ln.fluor))+
  geom_point(aes(colour=trt))+theme_bw()+
  scale_x_continuous('Time')+
  scale_y_continuous('ln(Abundance)')
```



First, let's focus on applying this technique to a single time series, to get a feel for the methodology. In the process, a series of 5 different possible models are fit to the supplied time series data: (1) a linear model, (2) a lagged growth model, (3) a saturating growth model, (4) a model of exponential decline that hits a floor, characteristic of a detection threshold, and (5) a model where growth is both lagged and achieves saturation. Any model fit where the inferred phase of exponential growth (increasing or decreasing linear portion) contains fewer than three observations is automatically excluded from consideration. The remaining models are then competed against each other based on information criteria (IC), and the estimate of exponential growth corresponding to the best model is returned. Users can select one of a few different IC options, including AIC, AICc, BIC; more on this later.

Here is an example from a single time series:

```
# subset the data, focusing on population A:
sdat2<-sdat[sdat$trt=='A',]

# calculate growth rate using all available methods:
res<-get.growth.rate(sdat2$dtime,sdat2$ln.fluor,plot.best.Q = T,id = 'Population A')
```



The result (shown in the above plot) indicates that both lagged and saturating portions are present in the time series, as the lagsat model outperforms all of the simpler models. The inferred exponential phase - highlighted in red - has a slope that corresponds to the estimated growth rate, obtained from the best model:

```
res$best.model
```

```
## [1] "gr.lagsat"
```

```
res$best.slope
```

```
## [1] 1.000003
```

A variety of other diagnostics are available, including the R2 for the best model:

```
res$best.model.rsqr
```

```
## [1] 0.9949958
```

The standard error associated with the slope estimate:

```
res$best.se
```

```
## [1] 0.1003267
```

And the number of observations falling within the exponential phase identified in the best model, as well as the R2 for just the exponential portion of the model:

```
res$best.model.slope.n
```

```
## [1] 5
```

```
res$best.model.slope.r2
```

```
## [1] 0.9998889
```

These diverse diagnostics can be used in downstream analyses to assess the quality of the model fits, and the reliability of the exponential growth rate estimates.

It’s also possible to access information from the entire suite of models that were fit, not just the one selected as the best model. For example, here’s a summary of the lagged growth model (saturated, floored, and lag/sat models can be similarly accessed):

```
summary(res$models$gr.lag)
```

```
## 
## Formula: y ~ lag(x, a, b, B1, s = 1e-10)
## 
## Parameters:
##    Estimate Std. Error t value Pr(>|t|)    
## B1   3.3282     0.7505   4.435 0.003027 ** 
## a    1.0000     0.3084   3.243 0.014194 *  
## b    0.6964     0.1009   6.900 0.000231 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 0.5341 on 7 degrees of freedom
## 
## Number of iterations to convergence: 3 
## Achieved convergence tolerance: 1.49e-08
```

We can also get summary values for all of the models, including the slopes (growth rates), and the standard error, R2, and number of observations associated with the slope estimates:

```
res$slopes
```

```
## [1] 0.5606061 0.6964286 0.5964945 1.0000030
```

```
res$ses
```

```
## [1] 0.07157291 0.10093405 0.09890661 0.10032667
```

```
res$slope.rs
```

```
## [1] 0.8846440 0.8729339 0.8880417 0.9998889
```

```
res$slope.ns
```

```
## [1] 10  7  9  5
```

Looking at these, we can see that the growth rate estimates from the linear, lagged, and saturating models are all significantly lower than the estimate from the lagged and saturated model, which captures the correct slope.

This is also reflected in the standard error values associated with each slope estimate, obtained via:

```
res$best.se
```

```
## [1] 0.1003267
```

```
res$ses
```

```
## [1] 0.07157291 0.10093405 0.09890661 0.10032667
```

Applying this approach to many time series

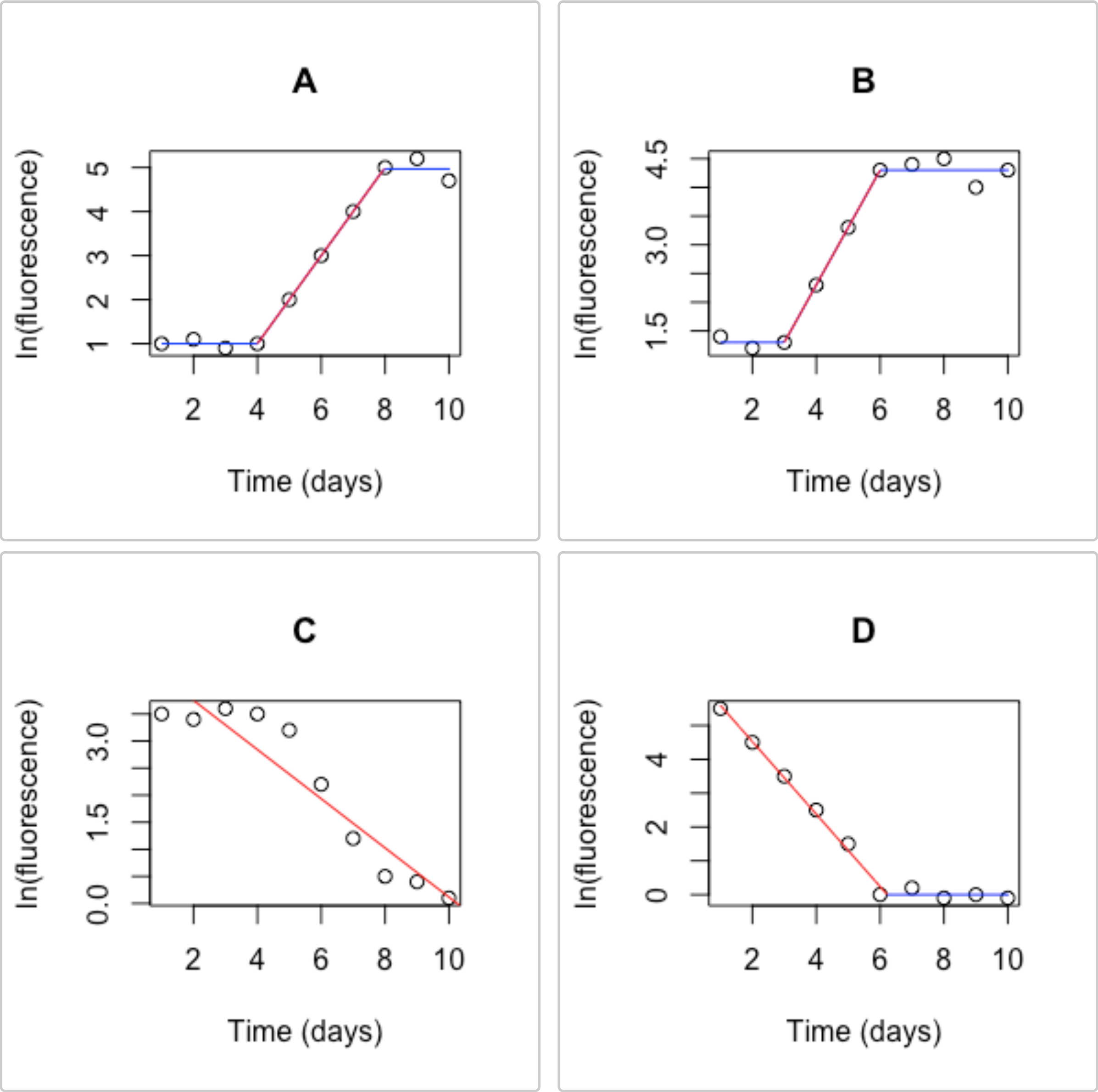
Often we want to extract growth rates from a whole bunch of individual populations/time series. The `get.growth.rate()` function is set up to make automating this process fairly easy, as we will see in this section. This approach takes advantage of `dplyr`.

First, we specify the data frame (`sdat`) containing all of our time series, then we pipe this to the `group_by()` command using the `%>%` syntax. The `group_by` command from `dplyr` is used to identify the column(s) that together identify unique individual time series. For the current example, this is just the treatment column `trt`. In a more complicated data set, this might include several columns that identify species, culture conditions, replicates, etc.

The result is a grouped data frame, which we directly pass to the `do()` command, again using `%>%`. This command applies the `get.growth.rate()` function to each individual time series identified by the grouping variables from the previous step. We pass the time column (`$dtime`) and the ln(abundance) column (`$ln.fluor`) to `get.growth.rate()`, and can also specify additional options, including the id of each population/time series, and plotting options.

In this instance, we request that the best fitting models are plotted, but leave the file path (`fpath`) option as `NA`, so the plots are displayed rather than saved to a directory. When working with a larger data set, the number of plots can be unwieldly, so it is recommended that you save rather than immediately display them.

```
gdat <- sdat %>% group_by(trt) %>%
do(grs=get.growth.rate(x=.$dtime,y=.$ln.fluor,id=.$trt,plot.best.Q=T,fpath=NA))
```



The resulting data frame is a complex structure - it contains the columns used for grouping (in this case, just `trt`), as well as a new column called `grs`. The entries in `grs` consist of the result of applying `get.growth.rate()` to each time series, including the identity of the best fit model, slope estimates, model contents, etc.

We can process this complex data structure further to obtain the different output that we're interested in. Say for example that we just want the best slope (growth rate) estimates, the identity of the best model, and the standard error associated with the growth rate estimate:

```
gdat %>% summarise(trt,mu=grs$best.slope,best.model=grs$best.model,best.se=grs$best.se)
```

```
## # A tibble: 4 x 4
##      trt         mu best.model    best.se
##   <fctr>      <dbl>      <chr>      <dbl>
## 1      A  1.0000030  gr.lagsat 0.10032667
## 2      B  1.0000090  gr.lagsat 0.19634081
## 3      C -0.4545455         gr 0.05849819
## 4      D -1.0714286     gr.flr 0.03823105
```

The result lists each population/time series, the best growth rate estimate (`mu`), and the identity of the model that produced this estimate.

However, you can select any diagnostics you want by including them in the `summarise()` command. For example, we could also request the R2 of the best model, and the number of observations falling within the exponential phase:

```
gdat %>%
summarise(trt,mu=grs$best.slope,best.model=grs$best.model,best.se=grs$best.se,best.R2=grs$bes
```

```
## # A tibble: 4 x 6
##      trt         mu best.model    best.se   best.R2 nobs.exp
##   <fctr>      <dbl>      <chr>      <dbl>     <dbl>    <int>
## 1      A  1.0000030  gr.lagsat 0.10032667 0.9949958        5
## 2      B  1.0000090  gr.lagsat 0.19634081 0.9909910        4
## 3      C -0.4545455         gr 0.05849819 0.8830012       10
## 4      D -1.0714286     gr.flr 0.03823105 0.9955992        6
```

Method specification

In some situations it may be desirable to only invoke particular models/methods when estimating growth rate. For example, appropriately detecting lags or saturation requires longer time series, otherwise there's a greater chance of overfitting the data - so two or three observations won't be sufficient. Indeed, situations with only 2 timepoints will throw an error message, although the algorithm still proceeds with the linear model.

In these cases, we might prefer to tell the algorithm to just calculate growth rate using a simple linear regression, and not bother with the more complicated approaches. Here's an example (note that I've also turned off the plotting feature):

```
# Only use the linear method:
gdat <- sdat %>% group_by(trt) %>%
do(grs=get.growth.rate(x=.$dtime,y=.$ln.fluor,id=.$trt,plot.best.Q=F,methods=c('linear')))
%>% summarise(trt,mu=grs$best.slope,best.model=grs$best.model)
gdat
```

```
## # A tibble: 4 x 3
##      trt         mu best.model
##   <fctr>      <dbl>      <chr>
## 1      A  0.5606061         gr
## 2      B  0.4181818         gr
## 3      C -0.4545455         gr
## 4      D -0.6563636         gr
```

Or maybe we want to use the lagged, saturated, or floored models, but not linear or lagsat:

```
# Only use the linear method:
gdat <- sdat %>% group_by(trt) %>%
do(grs=get.growth.rate(x=.$dtime,y=.$ln.fluor,id=.$trt,plot.best.Q=F,methods=c('lag','sat','f
%>% summarise(trt,mu=grs$best.slope,best.model=grs$best.model)
gdat
```

```
## # A tibble: 4 x 3
##      trt         mu best.model
##   <fctr>      <dbl>      <chr>
## 1      A  0.6964286     gr.lag
## 2      B  0.6228571     gr.sat
## 3      C -0.4545455     gr.flr
## 4      D -1.0714286     gr.flr
```

Model selection

We can also indicate which of three information criteria to use when selecting the ‘best model’ - AIC, AICc, or BIC. The default if no option is specified is to use AICc, which adjusts for the effects of small sample sizes, but converges on AIC in the limit of large sample sizes. Here’s an example invoking BIC instead:

```
gdat <- sdat %>% group_by(trt) %>%
do(grs=get.growth.rate(x=.$dtime,y=.$ln.fluor,id=.$trt,plot.best.Q=F,model.selection=c('BIC')
%>% summarise(trt,mu=grs$best.slope,best.model=grs$best.model)
gdat
```

```
## # A tibble: 4 x 3
##      trt         mu best.model
##   <fctr>      <dbl>      <chr>
## 1      A  1.0000030  gr.lagsat
## 2      B  1.0000090  gr.lagsat
## 3      C -0.4545455         gr
## 4      D -1.0714286     gr.flr
```

Note that if one model strongly outperforms the rest, the choice of IC to use likely won't change the outcome.

Fit Thermal Performance Curve to growth rate data

Load data set:

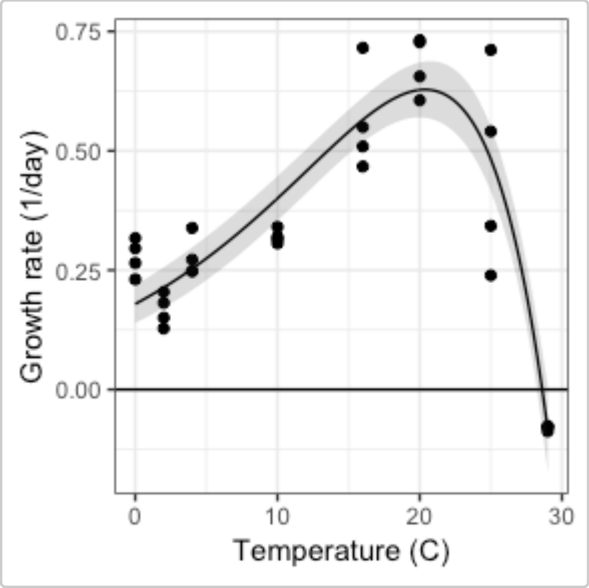
```
head(example_TPC_data)
```

```
##       experiment.ID   isolate.id temperature dilution replicate        mu
## 1  201802_RI03_CH_0 CH30_4_RI_03           0        1        R1 0.2957019
## 2  201802_RI03_CH_0 CH30_4_RI_03           0        1        R2 0.3171575
## 3  201802_RI03_CH_0 CH30_4_RI_03           0        1        R3 0.2308663
## 4  201802_RI03_CH_0 CH30_4_RI_03           0        1        R4 0.2649214
## 5 201802_RI03_CH_10 CH30_4_RI_03          10        1        R1 0.3404418
## 6 201802_RI03_CH_10 CH30_4_RI_03          10        1        R2 0.3192375
##   best.model
## 1     gr.sat
## 2     gr.sat
## 3         gr
## 4     gr.sat
## 5     gr.lag
## 6         gr
```

This data set contains information on the growth rate (mu) of one focal strain of a diatom (isolate.id) collected from Narragannsett Bay, RI in 2017. This isolate was grown at a range of differen temperatures, over three separate dilution periods, and replicated three times at each level. The final column, 'best.model' reflects the model used to provide the growth rate estimates, using functions from the `growthTools` package as described in the preceding section.

Now, we can use the function `get.nbcurve.tpc()` to fit thermal performance curves (using the re-parameterized Norberg function - see `?nbcurve2()`) to the growth rate data. This is easily done for a single dilution period:

```
# Single data set:
sp1 <- example_TPC_data %>% filter(isolate.id=='CH30_4_RI_03' & dilution==1)

# obtain Norberg curve parameters, using a grid search algorithm to consider a range of
possible initial parameter values, and plot the results
nbcurve.traits<-get.nbcurve.tpc(sp1$temperature,sp1$mu,method='grid.mle2',plotQ=T,conf.bandQ
= T,fpath=NA)
```

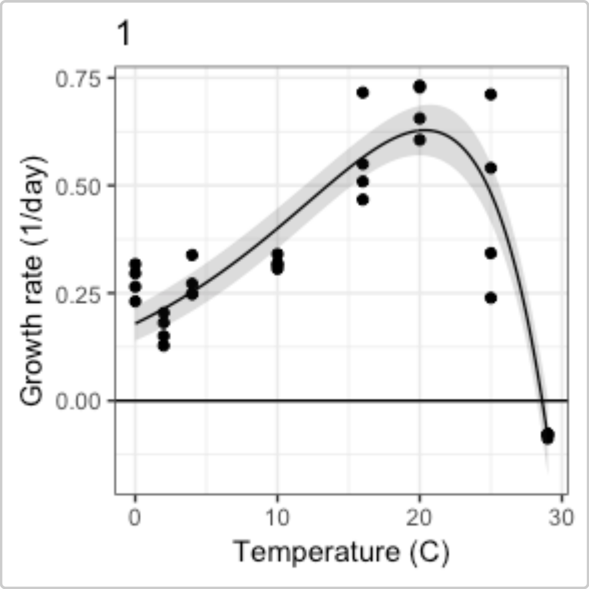


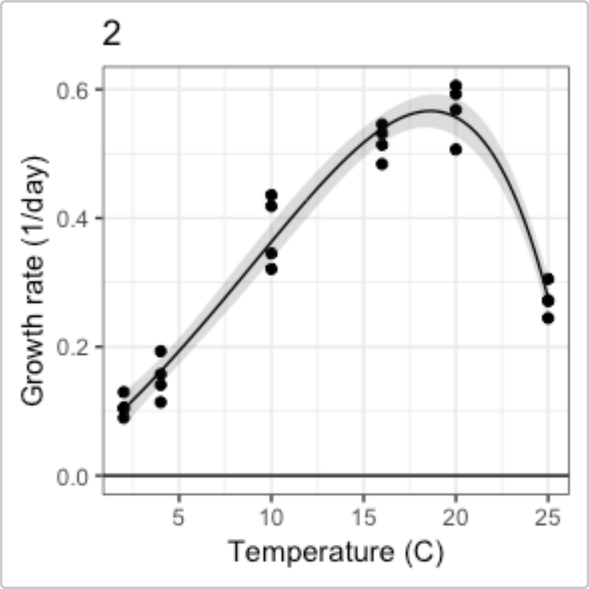
```
data.frame(nbcurve.traits)
```

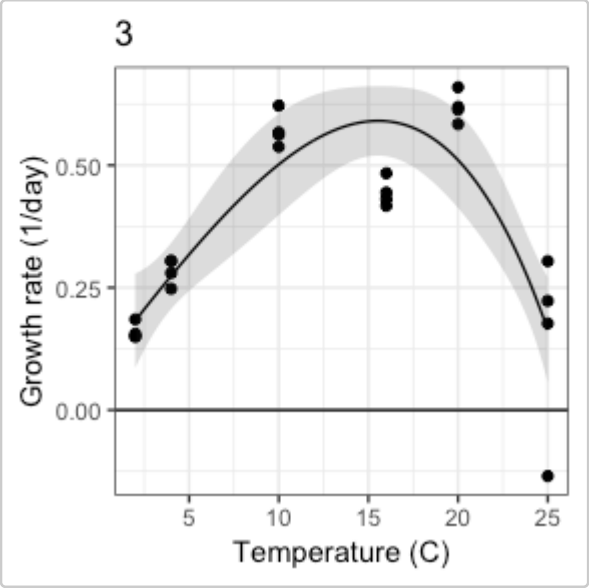
```
##          o        w         a         b        s      rsqr      tmin
## o 20.37253 103.7759 -1.494426 0.1108879 -2.36696 0.8405531 -75.16312
## w 20.37253 103.7759 -1.494426 0.1108879 -2.36696 0.8405531 -75.16312
## a 20.37253 103.7759 -1.494426 0.1108879 -2.36696 0.8405531 -75.16312
## b 20.37253 103.7759 -1.494426 0.1108879 -2.36696 0.8405531 -75.16312
## s 20.37253 103.7759 -1.494426 0.1108879 -2.36696 0.8405531 -75.16312
##       tmax    ciF.2.5..  ciF.97.5..        vcov.o        vcov.w
## o 28.61281  19.51247130  21.2325973  1.925522e-01  5.375506e-05
## w 28.61281 103.77526579 103.7765956  5.375506e-05  1.150784e-07
## a 28.61281  -1.71497067  -1.2738813 -4.402960e-02 -2.839893e-05
## b 28.61281   0.09843272   0.1233431  2.503967e-03  1.202288e-06
## s 28.61281  -2.61198735  -2.1219334 -2.343350e-04  6.075243e-07
##          vcov.a        vcov.b        vcov.s nobs ntemps   logLik       aic
## o -4.402960e-02  2.503967e-03 -2.343350e-04   32      8 30.33986 -50.67972
## w -2.839893e-05  1.202288e-06  6.075243e-07   32      8 30.33986 -50.67972
## a  1.266138e-02 -6.537049e-04  6.288011e-05   32      8 30.33986 -50.67972
## b -6.537049e-04  4.038202e-05 -3.768712e-06   32      8 30.33986 -50.67972
## s  6.288011e-05 -3.768712e-06  1.562844e-02   32      8 30.33986 -50.67972
```

But can also be automatically applied to multiple dilution periods or otherwise independent thermal performance curves (e.g., from different species or populations):

```
# First, let's look at an example with multiple dilutions but the same strain:
sp1b <- example_TPC_data %>% filter(isolate.id=='CH30_4_RI_03')

# apply get.nbcurve to the entire data set, grouping by isolate and dilution
res <- sp1b %>% group_by(isolate.id,dilution) %>%
do(tpcs=get.nbcurve.tpc(.$temperature,.$mu,method='grid.mle2',plotQ=T,conf.bandQ=T,fpath=NA,i
```





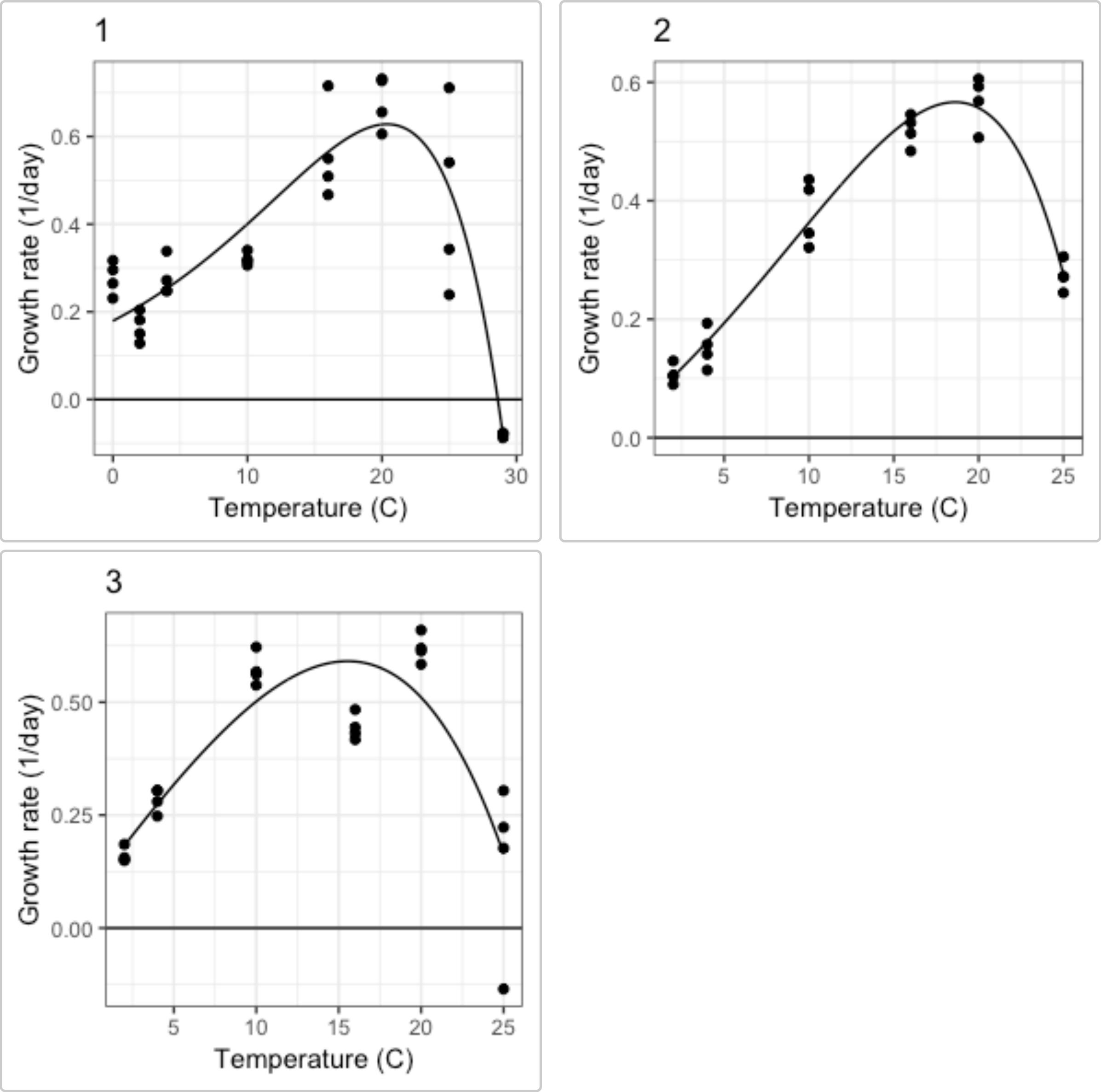


This approach also allows a fair amount of customization. For example, we can save the plots to a file folder, rather than displaying them directly:

```
# or saving resulting plots:
#fpath<-'/Users/colin/Research/Software/growthTools/user/'

# provide an explicit fpath to invoke plot saving; when `id` is also provided, this column
will be used to produce the plot's title, as well as included in the file name.
#res <- sp1b %>% group_by(isolate.id,dilution) %>%
do(tpcs=get.nbcurve.tpc(.$temperature,.$mu,method='grid.mle2',plotQ=T,conf.bandQ=T,fpath=fpat
```

You can also turn off plotting all together, or turn off only the confidence bands:

```
nb.res <- sp1b %>% group_by(isolate.id,dilution) %>%
do(tpcs=get.nbcurve.tpc(.$temperature,.$mu,method='grid.mle2',plotQ=T,conf.bandQ=F,fpath=NA,i
```



```
sp1b %>% group_by(isolate.id,dilution) %>%
do(tpcs=get.nbcurve.tpc(.$temperature,.$mu,method='grid.mle2',plotQ=F,conf.bandQ=F,fpath=NA,i
```

```
## Source: local data frame [3 x 3]
## Groups: <by row>
##
## # A tibble: 3 x 3
##     isolate.id dilution        tpcs
## *       <fctr>    <int>      <list>
## 1 CH30_4_RI_03        1 <list [14]>
## 2 CH30_4_RI_03        2 <list [14]>
## 3 CH30_4_RI_03        3 <list [14]>
```

After executing any of these, you can use `summarise` to recover parameter estimates and other diagnostic information corresponding to each curve fit.

```
# process results
nb.res2<-nb.res %>%
summarise(isolate.id,dilution,topt=tpcs$o,tmin=tpcs$tmin,tmax=tpcs$tmax,rsqr=tpcs$rsqr,a=exp(

nb.res2
```

```
## # A tibble: 3 x 9
##     isolate.id dilution     topt       tmin     tmax      rsqr         a
##         <fctr>    <int>    <dbl>      <dbl>    <dbl>     <dbl>     <dbl>
## 1 CH30_4_RI_03        1 20.37253 -75.163123 28.61281 0.8405531 0.2243774
## 2 CH30_4_RI_03        2 18.62345  -2.193116 26.96102 0.9635391 0.1817672
## 3 CH30_4_RI_03        3 15.54443  -1.802635 26.40737 0.7117982 0.3653026
## # ... with 2 more variables: b <dbl>, w <dbl>
```

A variety of other diagnostics are available, including log likelihood of the model fit, AIC, the total number of observations the regression is based on, how many unique temperature treatments are involved, and approximate confidence intervals for the coefficients of the model (based on Fisher information). These can be accessed by including additional terms in the summarise command, for example:

```
nb.res %>%
summarise(isolate.id,dilution,topt=tpcs$o,topt.lwr=tpcs$ciF[1,1],topt.upr=tpcs$ciF[1,2],ntemp
```

```
## # A tibble: 3 x 6
##     isolate.id dilution     topt topt.lwr topt.upr ntemps
##         <fctr>    <int>    <dbl>    <dbl>    <dbl>  <int>
## 1 CH30_4_RI_03        1 20.37253 19.51247 21.23260      8
## 2 CH30_4_RI_03        2 18.62345 17.94308 19.30382      6
## 3 CH30_4_RI_03        3 15.54443 12.44188 18.64698      6
```

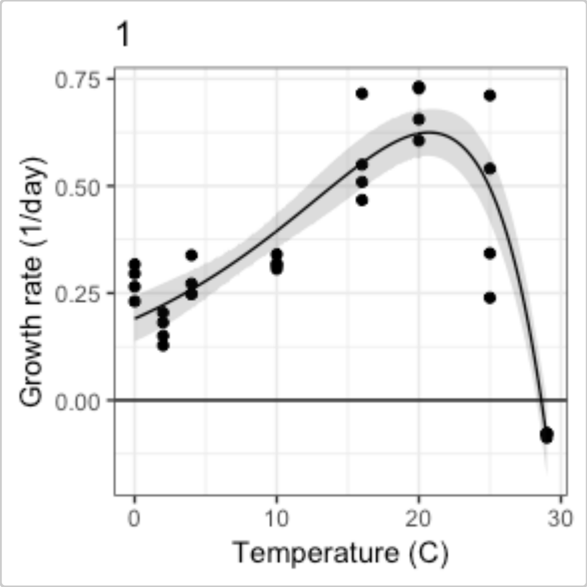
Double exponential model:

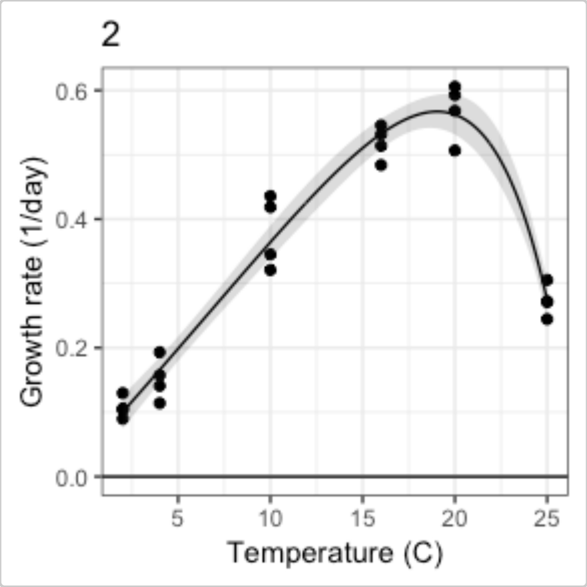
The package also allows fitting an alternative parametric equation, known currently as the double exponential model, to thermal performance curves. Details on this model can be found in Thomas et al. 2017 Global Change Biology. In brief, it models net population growth rate as the difference between two processes that depend exponentially on temperature: birth and death.

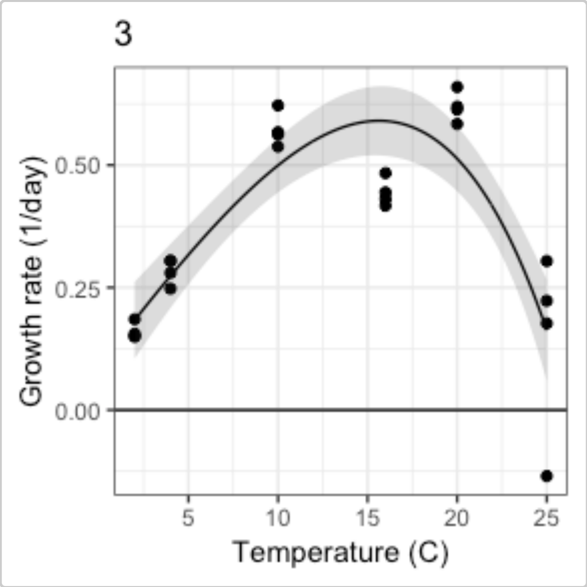
Right now, model fitting using this approach is pretty slow; this model is a 5 parameter model (Norberg uses 4 parameters), and several of the parameter estimates tend to covary strongly, so convergence takes many iterations of the optimization algorithm. There are ways to speed things up (by providing a smaller grid of initial parameter guesses to grid.mle2 behind the scenes), but this comes at an elevated risk of finding a local rather than globally optimal set of parameter estimates.

You can access this functionality the same way as fitting the Norberg curve, but using `get.decurve.tpc()` inside of the `dplyr` command:

```
de.res <- sp1b %>% group_by(isolate.id,dilution) %>%
do(tpcs=get.decurve.tpc(.$temperature,.$mu,method='grid.mle2',plotQ=T,conf.bandQ=T,fpath=NA,i
```







```
de.res2 <- de.res %>%
summarise(isolate.id,dilution,topt=tpcs$topt,tmin=tpcs$tmin,tmax=tpcs$tmax,rsqr=tpcs$rsqr,b1=

de.res2
```

```
## # A tibble: 3 x 10
##     isolate.id dilution     topt        tmin     tmax      rsqr        b1
##         <fctr>    <int>    <dbl>       <dbl>    <dbl>     <dbl>     <dbl>
## 1 CH30_4_RI_03        1 20.71794 -159.243606 28.61903 0.8423461 0.2036075
## 2 CH30_4_RI_03        2 19.04222   -1.172731 26.64611 0.9648146 2.0063929
## 3 CH30_4_RI_03        3 15.63678   -1.851455 26.36538 0.7127087 7.0850162
## # ... with 3 more variables: b2 <dbl>, d0 <dbl>, d2 <dbl>
```

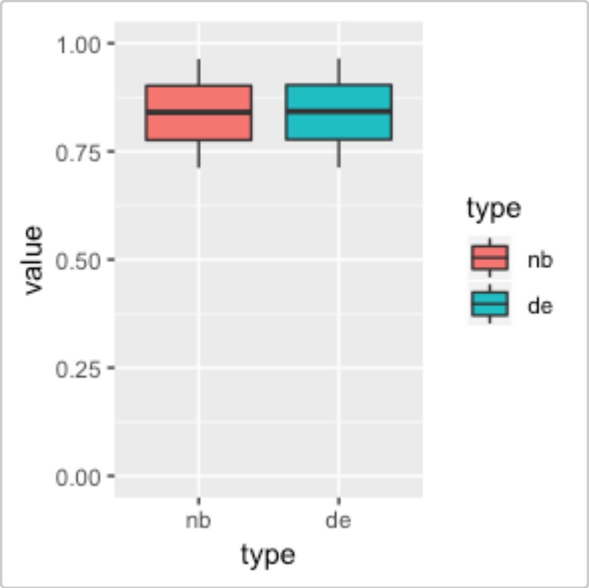
It should also be straightforward to fit both types of model to your data, then try to decide which model performs best, using criteria such as R2 or AIC. For now though, here’s a basic visual comparison:

```
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
nb.res3<-melt(nb.res2,id.vars=c('isolate.id','dilution'))
de.res3<-melt(de.res2,id.vars=c('isolate.id','dilution'))
res2<-rbind(data.frame(type='nb',nb.res3),data.frame(type='de',de.res3))

ggplot(res2[res2$variable=='rsqr',],aes(x=type,y=value))+
  geom_boxplot(aes(fill=type))+
  scale_y_continuous(limits=c(0,1))
```



Allow for multiple grouping variables:

We could also apply this approach to a larger data set, with more species and additional grouping variables:

```
table(example_TPC_data[,c('isolate.id','dilution')])
```

```
##                dilution
## isolate.id      1  2  3
##   CH30_4_RI_03 32 24 24
##   TH2_15_RI_03 32 24 24
```

```
# create informative ID column for each combo of unique strain and dilution period:
example_TPC_data$id<-paste(example_TPC_data$isolate.id,example_TPC_data$dilution)

res2 <- example_TPC_data %>% group_by(isolate.id,dilution) %>%
do(tpcs=get.nbcurve.tpc(.$temperature,.$mu,method='grid.mle2',plotQ=F,conf.bandQ=F,fpath=NA,i

clean.res <- res2 %>%
summarise(isolate.id,dilution,topt=tpcs$o,tmin=tpcs$tmin,tmax=tpcs$tmax,rsqr=tpcs$rsqr,a=exp(
data.frame(clean.res)
```

```
##     isolate.id dilution     topt       tmin     tmax      rsqr         a
## 1 CH30_4_RI_03        1 20.37253 -75.163123 28.61281 0.8405531 0.2243774
## 2 CH30_4_RI_03        2 18.62345  -2.193116 26.96102 0.9635391 0.1817672
## 3 CH30_4_RI_03        3 15.54443  -1.802635 26.40737 0.7117982 0.3653026
## 4 TH2_15_RI_03        1 19.13414 -24.304844 28.98711 0.9173149 0.2977834
## 5 TH2_15_RI_03        2 17.36286 -72.345184 25.73798 0.8422178 0.3296402
## 6 TH2_15_RI_03        3 15.24797  -2.814094 76.09215 0.3671341 1.5327040
##              b         w
## 1  0.11088790 103.77593
## 2  0.07190043  29.15413
## 3  0.03440944  28.21000
## 4  0.07847143  53.29196
## 5  0.10825405  98.08316
## 6 -0.03892924  78.90624
```