growthTools vignette

Colin T. Kremer

2019-05-21

1 Overview

The growthTools package is designed to help automate the estimation of exponential growth rates from large numbers of population time series (multiple measures of abundance or fluorescence over time), while adjusting for the possible presence of intervals of time where populations are not growing exponentially (e.g., lags in growth or saturating abundances due to density dependence). Additionally, the growthTools package includes functions for fitting two different parametric functions to data describing how exponential growth rate changes with temperature (aka thermal reaction norms or thermal performance curves).

The growthTools package relies heavily on data manipulation tools from the dplyr package to facilitate the automation of these analysis for large numbers of populations/strains/species.

2 Calculate exponential population growth rates

2.1 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()
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
# library(growthTools)

library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.5.2
```

```
library(dplyr)
library(tidyr)
```

```
## Warning: package 'tidyr' was built under R version 3.5.2
```

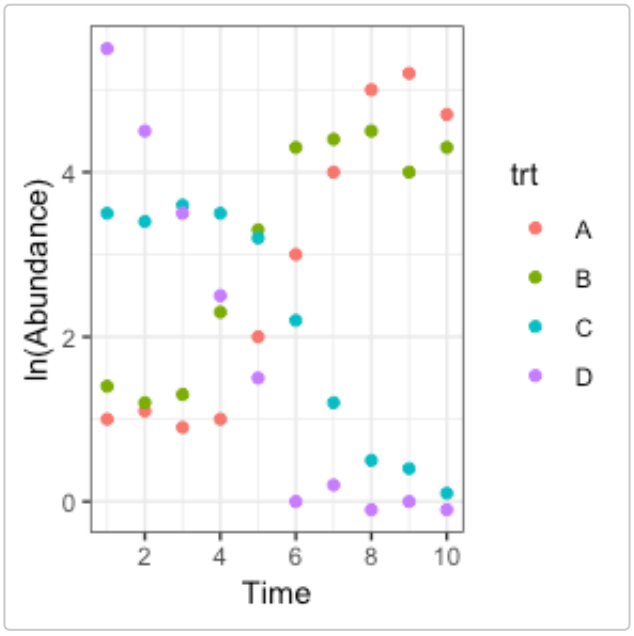
```
library(mleTools)
library(reshape2)
```

```
# 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), ln.fluor = c(c(1,
    1.1, 0.9, 1, 2, 3, 4, 5, 5.2, 4.7), c(1.1, 0.9, 1, 2, 3,
    4, 4.1, 4.2, 3.7, 4) + 0.3, c(3.5, 3.4, 3.6, 3.5, 3.2, 2.2,
    1.2, 0.5, 0.4, 0.1), c(5.5, 4.5, 3.5, 2.5, 1.5, 0, 0.2, -0.1,
    0, -0.1)))
```

2.2 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(name = "Time", breaks = seq(0,
    10, 2)) + scale_y_continuous(name = "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,

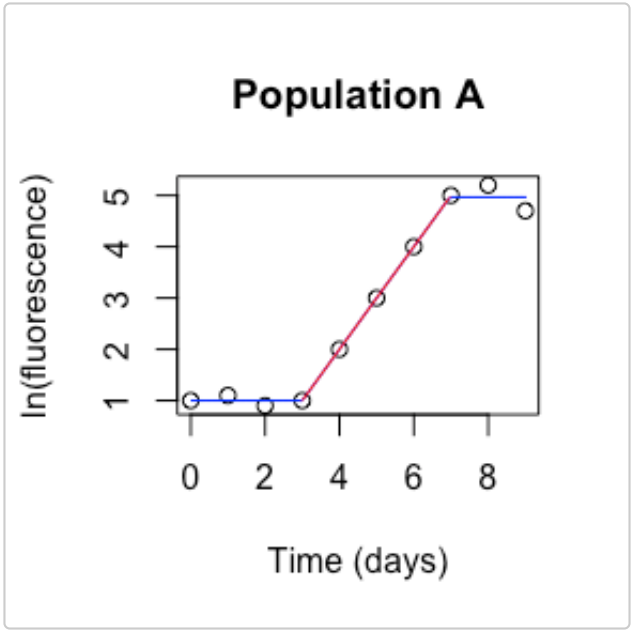
characteristic of a detection threshold, and (5) a model where growth experiences both an initial lag and achieves saturation.

Note that by default: * before fitting occurs, the x-axis (time) is shifted to start at time=0, if necessary. This can improve the stability of the regression fitting. To avoid this, you can specify `zero.time=FALSE` when calling `get.growth.rate()`) * any regression where the inferred phase of exponential growth (increasing or decreasing linear portion) contains fewer than three observations is automatically excluded from consideration, and will not appear in the output. This can be adjusted using the `min.exp.obs` option, but values less than 3 are not recommended. * the remaining models are then competed against each other based on Akaike Information Criteria adjusted for small sample sizes (AICc). Users may instead select AIC or BIC.

The best model identified during model comparison is used to provide the reported estimate of exponential growth.

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. If you are interested, you can extract the IC table used to make this decision (defaults to AICc comparison):

```
res$ictab
```

```
##           dAICc df
## gr.lagsat  0.0  5
## gr        16.4  3
## gr.lag    17.1  4
## gr.sat    20.8  4
```

Given the lagsat model, 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] 0.1003313
```

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   2.3282     0.7505   3.102 0.017269 *
## 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
```

```
##        gr    gr.lag    gr.sat gr.lagsat
## 0.5606061 0.6964286 0.5961677 1.0000030
```

```
res$ses
```

```
##         gr     gr.lag     gr.sat  gr.lagsat
## 0.07157291 0.10093405 0.09890251 0.10033133
```

```
res$slope.rs
```

```
##        gr    gr.lag    gr.sat gr.lagsat
## 0.8846440 0.8729339 0.8880262 0.9998889
```

```
res$slope.ns
```

```
##        gr    gr.lag    gr.sat gr.lagsat
##        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] 0.1003313
```

```
res$ses
```

```
##         gr     gr.lag     gr.sat  gr.lagsat
## 0.07157291 0.10093405 0.09890251 0.10033133
```

2.3 Applying this approach to many time series

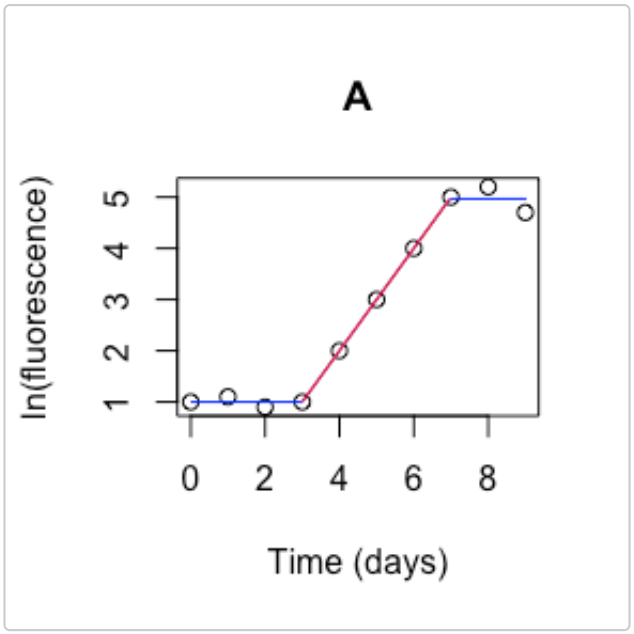
Often we want to extract growth rates from a whole bunch of individual populations/time series. The `get.growth.rate()` function is set up to make automating this process fairly easy, as we will see in this section. This approach takes advantage of `dplyr`.

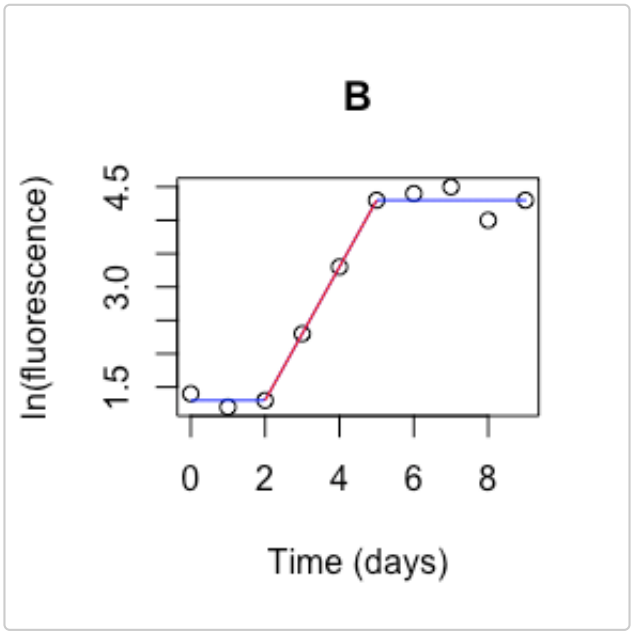
First, we specify the data frame (`sdat`) containing all of our time series, then we pipe this to the `group_by()` command using the `%>%` syntax. The `group_by` command from `dplyr` is used to identify the column(s) that together identify unique individual time series. For the current example, this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

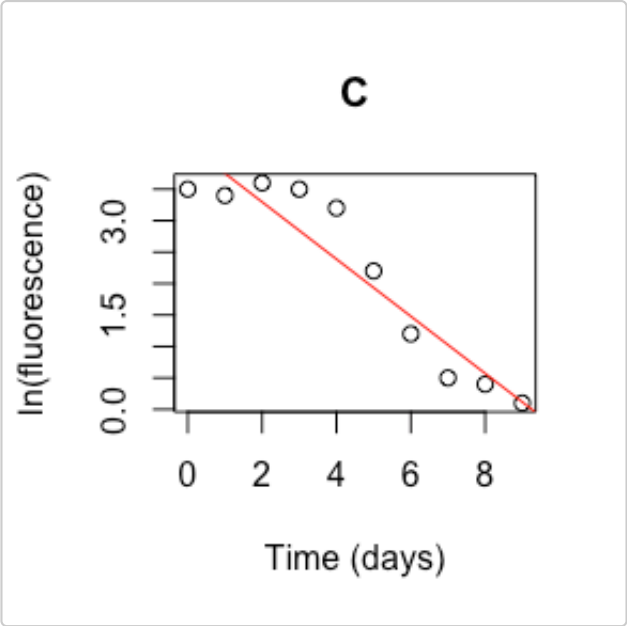
The result is a grouped data frame, which we directly pass to the `do()` command, again using `%>%`. This command applies the `get.growth.rate()` function to each individual time series identified by the grouping variables from the previous step. We pass the time column (`$dtime`) and the ln(abundance)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

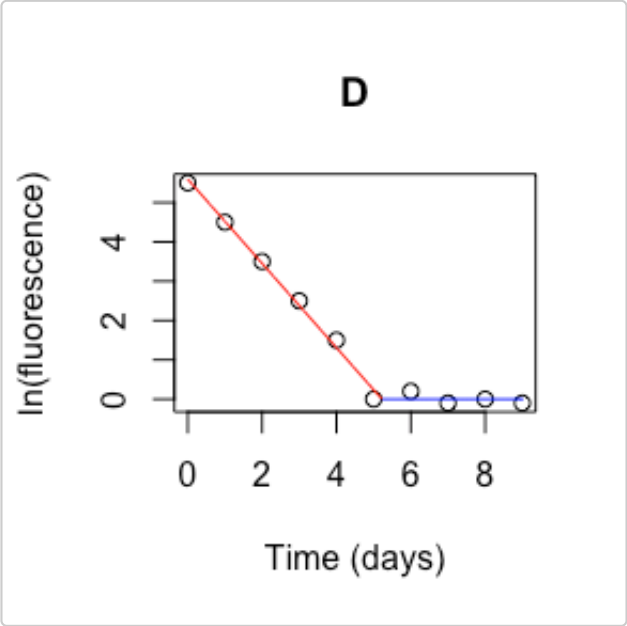
In this instance, we request that the best fitting models are plotted, but leave the file path (`fpath`) option as `NA`, so the plots are displayed rather than saved to a directory. When working with a larger data set, the number of plots can be unwieldly, so it is recommended that you save rather than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


A
ln(fluorescence)
1
2
3
4
5
0
2
4
6
8
Time (days)


B
ln(fluorescence)
1.5
3.0
4.5
0
2
4
6
8
Time (days)





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
##   <fct>  <dbl> <chr>        <dbl>
## 1 A      1.00  gr.lagsat   0.100
## 2 B      1.00  gr.lagsat   0.196
## 3 C     -0.455 gr          0.0585
## 4 D     -1.07  gr.flr      0.0382
```

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$best.se, best.R2 = grs$best.model.rsqr, nobs.exp =
grs$best.model.slope.n)
```

```
## # A tibble: 4 x 6
##   trt       mu best.model best.se best.R2 nobs.exp
##   <fct>  <dbl> <chr>        <dbl>   <dbl>    <int>
## 1 A      1.00  gr.lagsat   0.100    0.995        5
## 2 B      1.00  gr.lagsat   0.196    0.991        4
## 3 C     -0.455 gr          0.0585   0.883       10
## 4 D     -1.07  gr.flr      0.0382   0.996        6
```

2.4 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
##   <fct>  <dbl> <chr>
## 1 A      0.561 gr
## 2 B      0.418 gr
## 3 C     -0.455 gr
## 4 D     -0.656 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 = .$trt, plot.best.Q = F, methods = c("lag",
        "sat", "flr"))) %>% summarise(trt, mu = grs$best.slope,
    best.model = grs$best.model)
gdat
```

```
## # A tibble: 4 x 3
##   trt       mu best.model
##   <fct>  <dbl> <chr>
## 1 A      0.696 gr.lag
## 2 B      0.623 gr.sat
## 3 C     -0.455 gr.flr
## 4 D     -1.07  gr.flr
```

2.5 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. Generally, the basic linear model is favored more often using AICc than AIC, such that small lag or saturated phases may be ignored. Here's an example invoking AIC instead of AICc:

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

```
## # A tibble: 4 x 3
##   trt       mu best.model
##   <fct>  <dbl> <chr>
## 1 A      1.00  gr.lagsat
## 2 B      1.00  gr.lagsat
## 3 C     -0.455 gr
## 4 D     -1.07  gr.flr
```

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

3 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
```

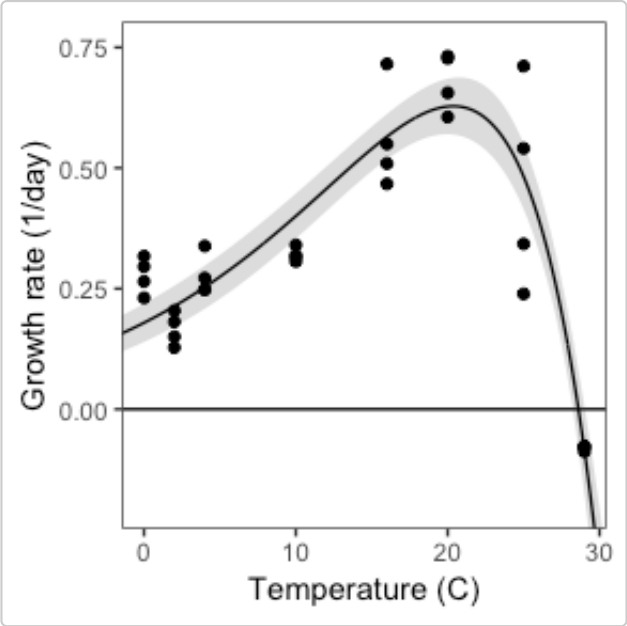
```
## 2  201802_RI03_CH_0 CH30_4_RI_03           0        1        R2 0.3171575
## 3  201802_RI03_CH_0 CH30_4_RI_03           0        1        R3 0.2308663
## 4  201802_RI03_CH_0 CH30_4_RI_03           0        1        R4 0.2649214
## 5 201802_RI03_CH_10 CH30_4_RI_03          10        1        R1 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.

3.1 Norberg model:

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)
```




```
nbcurve.traits
```

```
## $topt
## [1] 20.37253
##
## $w
## [1] 103.776
##
## $a
## [1] -1.494426
##
## $b
## [1] 0.1108879
##
## $s
## [1] -2.36696
##
## $rsqr
## [1] 0.8405531
##
## $tmin
## [1] -75.16319
```

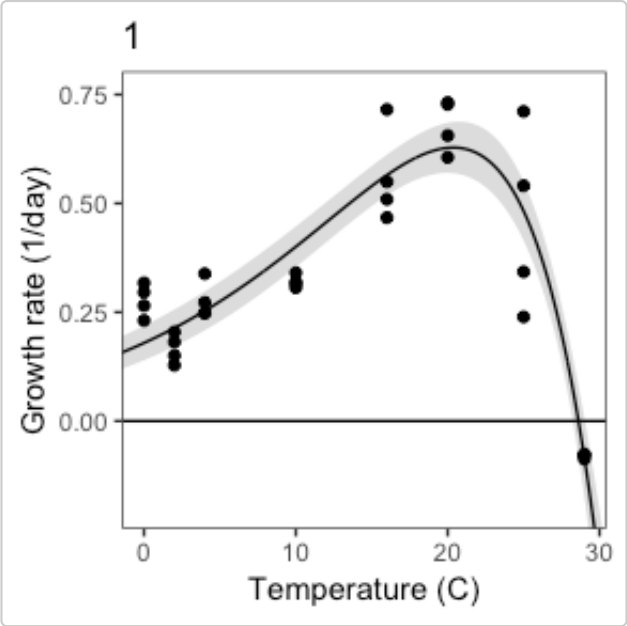
```
##
## $tmax
## [1] 28.61281
##
## $umax
## [1] 0.6281357
##
## $umax.ci
## [1] 0.05881265
##
## $ciF
##             2.5 %      97.5 %
## topt  19.51247148  21.2325974
## w    103.77533420 103.7766640
## a     -1.71497025  -1.2738809
## b      0.09843273   0.1233431
## s     -2.61198734  -2.1219334
##
## $cf
## $cf$topt
## [1] 20.37253
##
## $cf$w
## [1] 103.776
##
## $cf$a
## [1] -1.494426
##
## $cf$b
## [1] 0.1108879
##
## $cf$s
## [1] -2.36696
##
##
## $vcov
##               topt             w             a             b             s
## topt  1.925521e-01  5.375553e-05 -4.402960e-02  2.503967e-03 -2.343328e-04
## w     5.375553e-05  1.150787e-07 -2.839904e-05  1.202294e-06  6.075210e-07
## a    -4.402960e-02 -2.839904e-05  1.266138e-02 -6.537049e-04  6.287984e-05
```

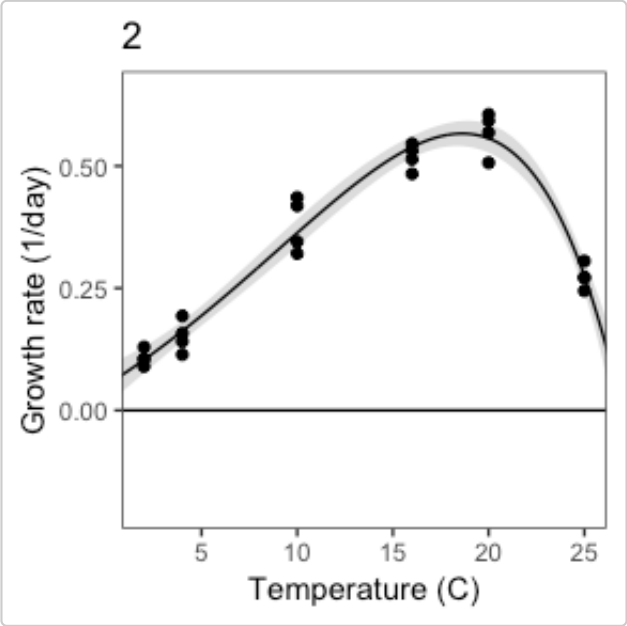
```
## b     2.503967e-03  1.202294e-06 -6.537049e-04  4.038202e-05 -3.768674e-06
## s    -2.343328e-04  6.075210e-07  6.287984e-05 -3.768674e-06  1.562844e-02
##
## $nobs
## [1] 32
##
## $ntemps
## [1] 8
##
## $logLik
## 'log Lik.' 30.33986 (df=5)
##
## $aic
## [1] -50.67972
##
## $data
##    temperature          mu
## 1            0  0.29570190
## 2            0  0.31715749
## 3            0  0.23086634
## 4            0  0.26492142
## 5           10  0.34044182
## 6           10  0.31923753
## 7           10  0.30635082
## 8           10  0.31520954
## 9           16  0.54979405
## 10          16  0.50926419
## 11          16  0.46722819
## 12          16  0.71566211
## 13           2  0.12797674
## 14           2  0.18134072
## 15           2  0.15040029
## 16           2  0.20437558
## 17          20  0.73161025
## 18          20  0.72734410
## 19          20  0.60578314
## 20          20  0.65573113
## 21          25  0.54062615
## 22          25  0.34272669
## 23          25  0.71122612
```

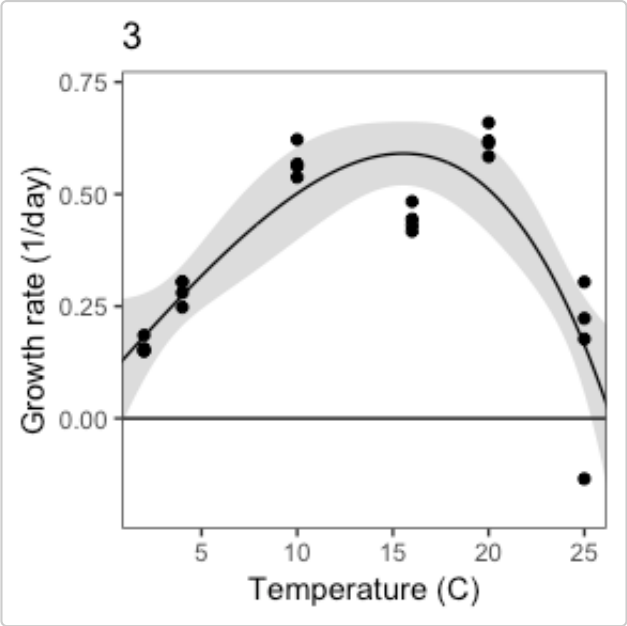
```
## 24          25  0.23908063
## 25          29 -0.07640984
## 26          29 -0.07830364
## 27          29 -0.08734117
## 28          29 -0.07812054
## 29           4  0.24811323
## 30           4  0.33837116
## 31           4  0.24791146
## 32           4  0.27200425
```

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,
    id = .$dilution))
```


1
0.75
0.50
0.25
0.00
Growth rate (1/day)
0
10
20
30
Temperature (C)


2
0.50
0.25
0.00
Growth rate (1/day)
5
10
15
20
25
Temperature (C)

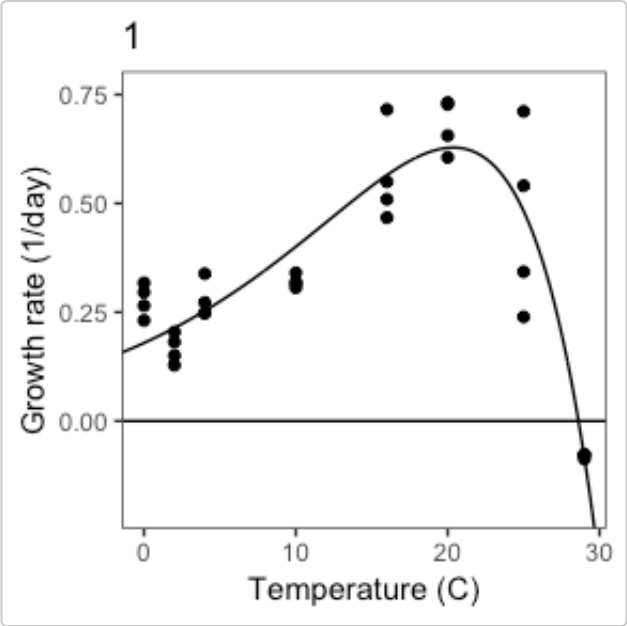


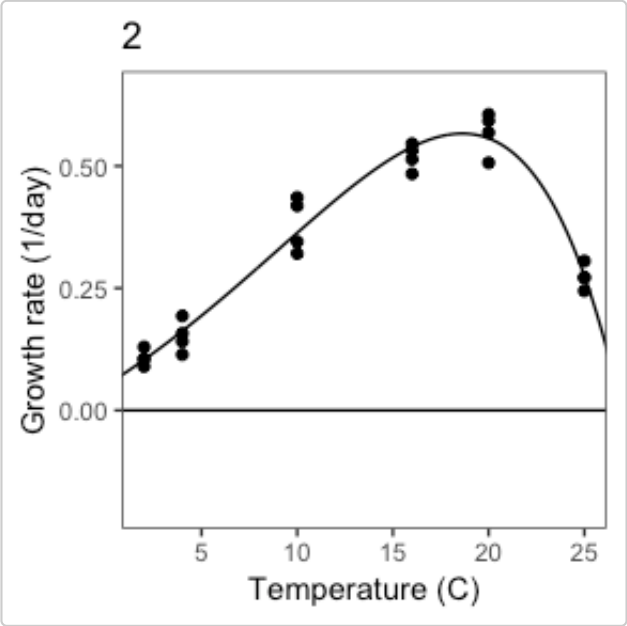
This approach also allows a fair amount of customization. For example, we can save the plots to a file folder, rather than displaying them directly:

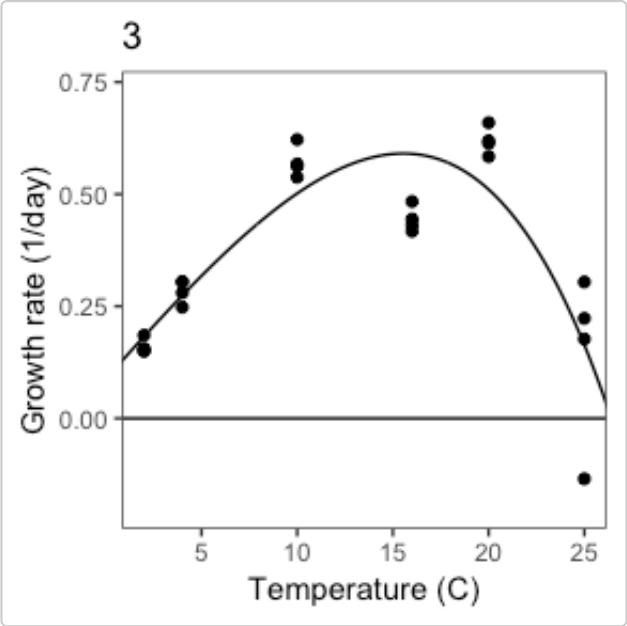
```
# or saving resulting plots:
# fpath<-'/Users/colin/Research/Software/growthTools/user/'

# 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.bana
```

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,
    id = .$dilution))
```


1
0.75
0.50
0.25
0.00
Growth rate (1/day)
0
10
20
30
Temperature (C)


2
0.50
0.25
0.00
Growth rate (1/day)
5
10
15
20
25
Temperature (C)




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

```
## Source: local data frame [3 x 3]
## Groups: <by row>
##
## # A tibble: 3 x 3
##   isolate.id   dilution tpcs
## * <fct>           <int> <list>
## 1 CH30_4_RI_03        1 <list [18]>
## 2 CH30_4_RI_03        2 <list [18]>
## 3 CH30_4_RI_03        3 <list [18]>
```

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$topt,
    tmin = tpcs$tmin, tmax = tpcs$tmax, umax = tpcs$umax, rsqr = tpcs$rsqr,
    a = tpcs$a, b = tpcs$b, w = tpcs$w)
```

```
nb.res2
```

```
## # A tibble: 3 x 10
##   isolate.id   dilution  topt   tmin  tmax  umax  rsqr     a      b     w
##   <fct>           <int> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl> <dbl>
## 1 CH30_4_RI_03        1  20.4 -75.2   28.6 0.628 0.841 -1.49 0.111  104.
## 2 CH30_4_RI_03        2  18.6  -2.19  27.0 0.566 0.964 -1.71 0.0719  29.2
## 3 CH30_4_RI_03        3  15.5  -1.80  26.4 0.591 0.712 -1.01 0.0344  28.2
```

A variety of other diagnostics are available, including the 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, ntemps = tpcs$ntemps,
    topt = tpcs$topt, topt.lwr = tpcs$ciF[1, 1], topt.upr = tpcs$ciF[1,
        2])
```

```
## # A tibble: 3 x 6
##   isolate.id   dilution ntemps  topt topt.lwr topt.upr
##   <fct>           <int>  <int> <dbl>    <dbl>    <dbl>
## 1 CH30_4_RI_03        1      8  20.4     19.5     21.2
## 2 CH30_4_RI_03        2      6  18.6     17.9     19.3
## 3 CH30_4_RI_03        3      6  15.5     12.4     18.6
```

Note that the output also provides an estimate of umax, the maximum growth rate (achieved at a temperature of Topt). Using the same method employed to generate confidence bands, we can also obtain an estimated 95% confidence interval around umax:

```
nb.res %>% summarise(isolate.id, dilution, ntemps = tpcs$ntemps,
    topt = tpcs$topt, topt.lwr = tpcs$ciF[1, 1], topt.upr = tpcs$ciF[1,
        2], umax = tpcs$umax, umax.lwr = tpcs$umax - tpcs$umax.ci,
    umax.upr = tpcs$umax + tpcs$umax.ci)
```

```
## # A tibble: 3 x 9
```

```
##   isolate.id dilution ntemps  topt topt.lwr topt.upr  umax umax.lwr
##   <fct>         <int>  <int> <dbl>    <dbl>    <dbl> <dbl>    <dbl>
## 1 CH30_4_RI…        1      8  20.4     19.5     21.2 0.628    0.569
## 2 CH30_4_RI…        2      6  18.6     17.9     19.3 0.566    0.541
## 3 CH30_4_RI…        3      6  15.5     12.4     18.6 0.591    0.520
## # … with 1 more variable: umax.upr <dbl>
```

3.2 Double exponential model:

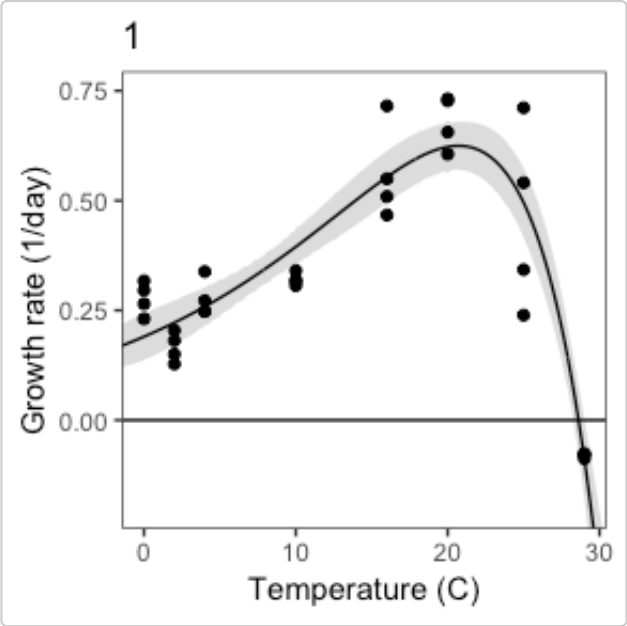
NOTE: many new features introduced into `get.nbcurve.tpc()` have not yet been added to `get.decurve.tpc`, but will be soon (5/21/19).

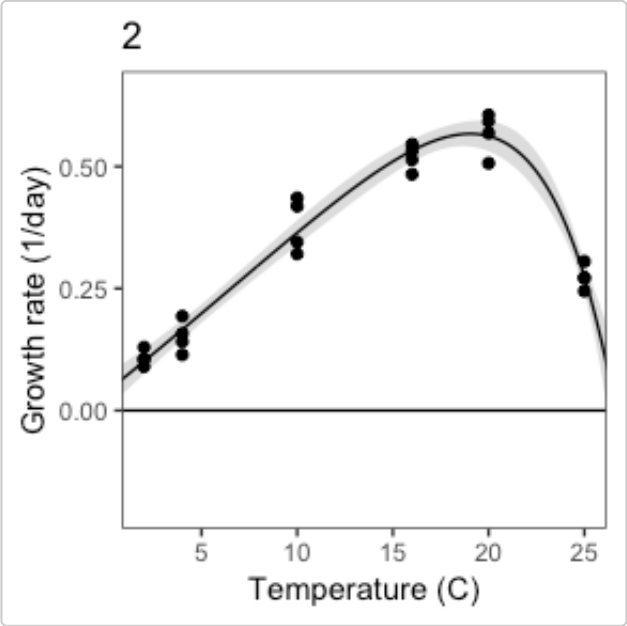
The package also allows fitting an alternative parametric equation, known currently as the double exponential model, to thermal performance curves. Details on this model can be found in Thomas et al. 2017, Global Change Biology. In brief, it models net population growth rate as the difference between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

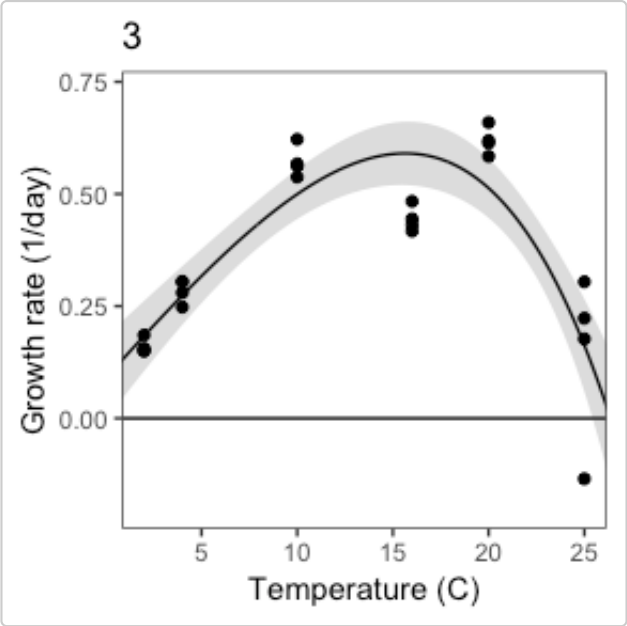
Right now, model fitting using this approach is pretty slow; this model is a 5 parameter model (Norberg uses 4 parameters), and several of the parameter estimates tend to covary strongly, so convergence takes many iterations of the optimization algorithm. There are ways to speed things up (by providing 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,
    id = .$dilution))
```


1
0.75
0.50
0.25
0.00
Growth rate (1/day)
0
10
20
30
Temperature (C)


2
0.50
0.25
0.00
Growth rate (1/day)
5
10
15
20
25
Temperature (C)




```
de.res2 <- de.res %>% summarise(isolate.id, dilution, topt = tpcs$topt,
    tmin = tpcs$tmin, tmax = tpcs$tmax, rsqr = tpcs$rsqr, b1 = tpcs$b1,
    b2 = tpcs$b2, d0 = tpcs$d0, d2 = tpcs$d2)
de.res2
```

```
## # A tibble: 3 x 10
##   isolate.id dilution  topt    tmin  tmax  rsqr    b1     b2      d0
d2
##   <fct>         <int> <dbl>   <dbl> <dbl> <dbl> <dbl>  <dbl>   <dbl>
<dbl>
## 1 CH30_4_RI…        1  20.7 -159.    28.6 0.842 0.204 0.0849 2.73e-7 0.180
## 2 CH30_4_RI…        2  19.0   -1.17  26.6 0.965 2.01  0.0160 1.97e+0 0.235
## 3 CH30_4_RI…        3  15.6   -1.85  26.4 0.713 7.08  0.0480 1.93e+0
0.0577
```

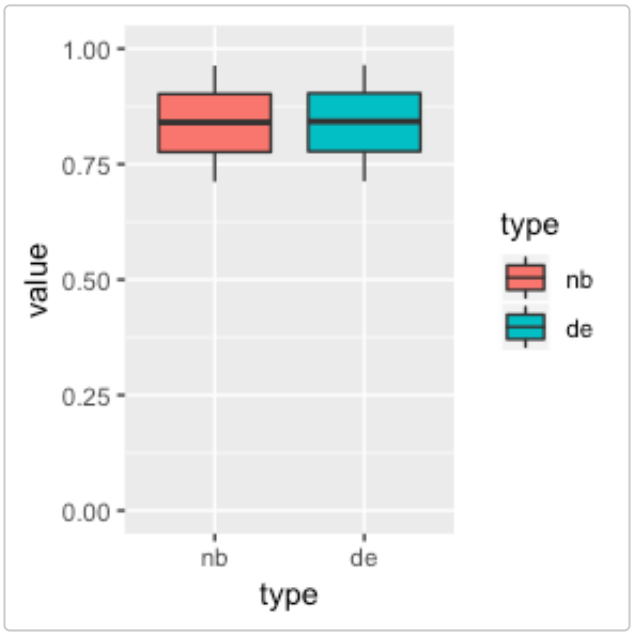
It should also be straightforward to fit both types of model to your data, then try to decide which model performs best, using criteria such as R2 or AIC. For now though, here's a basic visual comparison:

```
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))
res2[res2$variable == "rsqr", ]
```

```
##    type   isolate.id dilution variable     value
## 13   nb CH30_4_RI_03        1     rsqr 0.8405531
## 14   nb CH30_4_RI_03        2     rsqr 0.9635391
## 15   nb CH30_4_RI_03        3     rsqr 0.7117982
## 34   de CH30_4_RI_03        1     rsqr 0.8423461
## 35   de CH30_4_RI_03        2     rsqr 0.9648146
## 36   de CH30_4_RI_03        3     rsqr 0.7127087
```

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



3.3 Allow for multiple grouping variables:

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

```r
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
```

```r
# 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, id = .$id))

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

```
##     isolate.id dilution     topt       tmin     tmax      rsqr         a
## 1 CH30_4_RI_03        1 20.37253 -75.163191 28.61281 0.8405531 0.2243775
## 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 17.36326 -72.395537 25.73787 0.8422184 0.3297192
## 6 TH2_15_RI_03        3 15.24952  -2.818088 75.95296 0.3671215 1.5292428
##             b         w
## 1  0.11088791 103.77600
## 2  0.07190043  29.15413
## 3  0.03440944  28.21000
## 4  0.07847143  53.29196
## 5  0.10826745  98.13341
## 6 -0.03887415  78.77105
```

4 Visualize Thermal Performance Curves

There are two functions within the package to aid in plotting thermal performance curves (TPCs), that allow us to predict and plot values from a given TPC fit.

4.1 Predict values from a TPC

The first of these is `predict.nbcurve()`, which works quite similarly as other `predict()` functions in R. Basically, we can provide the function with the output from a single `get.nbcurve.tpc` fit and it will return a set of temperatures and predicted growth rates. For example:

```
# pull out the fit resulting from a single get.nbcurve.tpc:
fit.info <- nb.res[1, ]$tpcs

# generate predictions for a single curve:
tmp <- predict.nbcurve(fit.info)
head(tmp)
```

```
##   temperature        mu
## 1        -2.0 0.1495289
## 2        -1.9 0.1509083
## 3        -1.8 0.1522985
## 4        -1.7 0.1536996
## 5        -1.6 0.1551116
## 6        -1.5 0.1565345
```

By default, the output is for a sequence of temperatures from -2 to 40 C, in steps of 0.1. However, users can specify a custom set of values:

```
predict.nbcurve(fit.info, newdata = data.frame(temperature = c(20,
    25, 30)))
```

```
##   temperature         mu
## 1          20  0.6275080
## 2          25  0.4823347
## 3          30 -0.3385213
```

The function can also generate standard errors around the predicted growth rate values, using the delta method (see Bolker book, pg. 255).

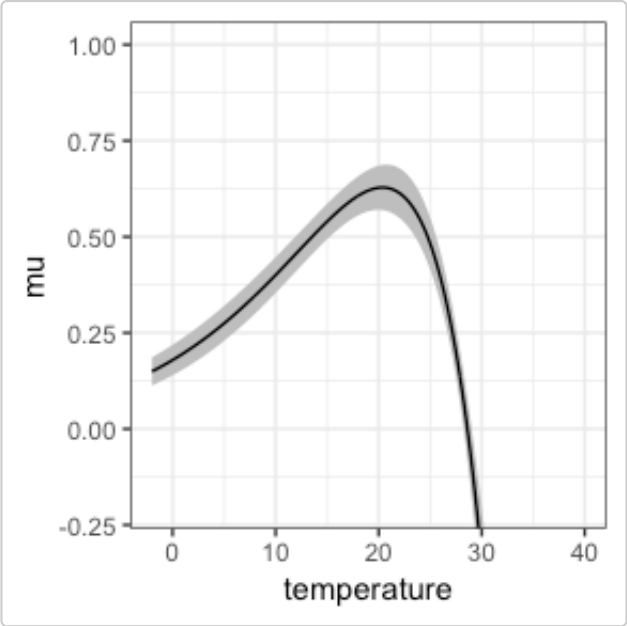
```
predict.nbcurve(fit.info, newdata = data.frame(temperature = c(20,
    25, 30)), se.fit = TRUE)
```

```
##   temperature         mu     se.fit
## 1          20  0.6275080 0.02930842
## 2          25  0.4823347 0.03432288
## 3          30 -0.3385213 0.06753617
```

4.2 Plot values from a TPC

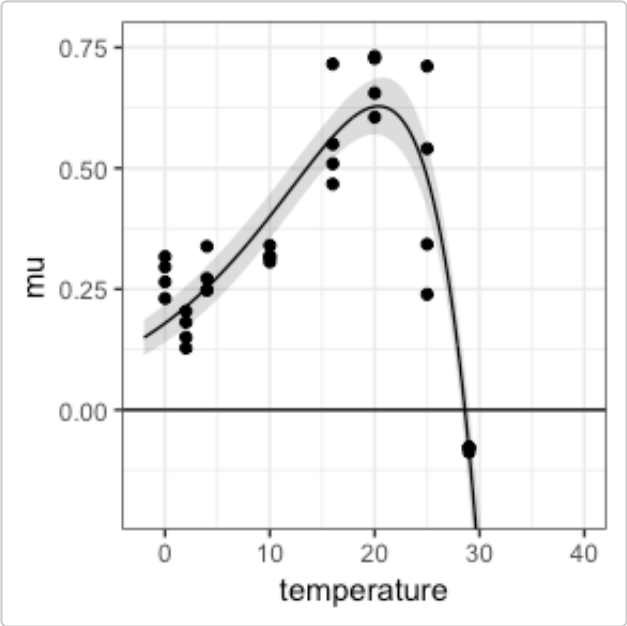
These predicted values can be used as the foundation for plotting fitted Norberg curves:

```
# generate predictions for a single curve, along with
# standard errors:
tmp <- predict.nbcurve(fit.info, se.fit = T)

# plot the predictions
ggplot(tmp, aes(x = temperature, y = mu)) + geom_ribbon(aes(ymin = mu -
    1.96 * se.fit, ymax = mu + 1.96 * se.fit), fill = "gray") +
    geom_line() + coord_cartesian(ylim = c(-0.2, 1)) + theme_bw()
```



For ease, this capacity has been formalized using the `plot.nbcurve()` function, which invokes `predict.nbcurve()` internally:

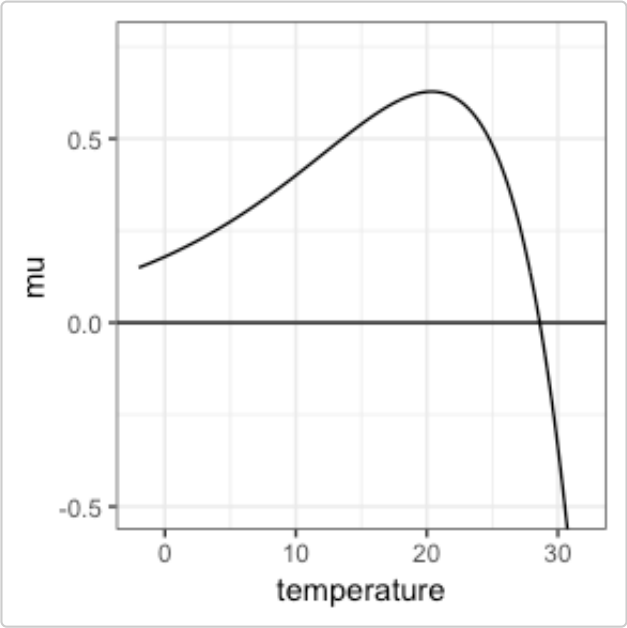
```
# use function to generate a single curve plot:
plot.nbcurve(fit.info)
```



We can also control whether the confidence band and raw data are displayed, or

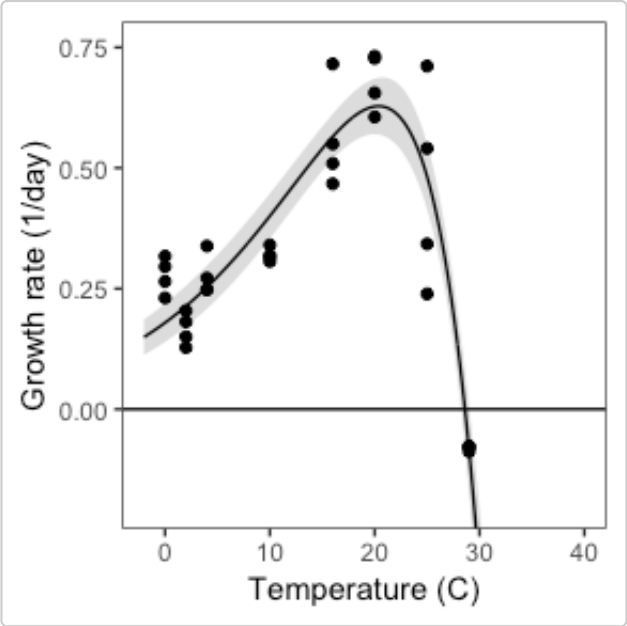
alter the range of the plot:

```
# use function to generate a single curve plot:
plot.nbcurve(fit.info, plot.ci = F, plot.obs = F, xlim = c(-2,
    32), ylim = c(-0.5, 1))
```



Finally, users can also save the ggplot2 object created by the `plot.nbcurve()` and then subsequently modify the layout of the plot, for example:

```
c1 <- plot.nbcurve(fit.info)

# demonstrate changes in layout
c1 + scale_x_continuous("Temperature (C)") + scale_y_continuous("Growth rate
(1/day)") +
    theme(panel.grid = element_blank())
```



4.3 Plot multiple TPCs at once:

The `plot.nbcurve()` function is suited to displaying the results of a single Norberg TPC, but cannot easily be used to combine multiple TPCs within a single plot. To accomplish this, it's better to take advantage of the `predict.nbcurve()` function, combined with the same `dplyr` capabilities that allow us to fit multiple TPCs in the first place.

Recall that when we used `get.nbcurve.tpc` to fit multiple curves, we defined a set of grouping variables that uniquely identified the growth rate data for each individual TPC. The resulting output is a complex data frame that retains columns for each of these grouping variables, as well as a new column (which we called `tpcs`), which holds the individual Norberg curve fits.

We can take this output and process it using `predict.nbcurve()` as follows:

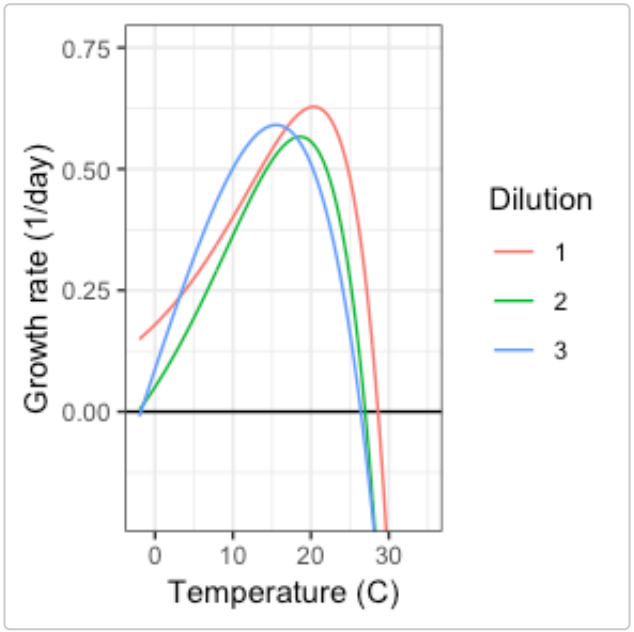
```
# generate predictions for multiple curves:
tmp2 <- nb.res %>% group_by(isolate.id, dilution) %>%
do(predict.nbcurve(.$tpcs,
    se.fit = T))
head(tmp2)
```

```
## # A tibble: 6 x 5
```

```
## # Groups:   isolate.id, dilution [1]
##   isolate.id   dilution temperature    mu se.fit
##   <fct>           <int>       <dbl> <dbl>  <dbl>
## 1 CH30_4_RI_03        1        -2   0.150 0.0186
## 2 CH30_4_RI_03        1        -1.9 0.151 0.0187
## 3 CH30_4_RI_03        1        -1.8 0.152 0.0188
## 4 CH30_4_RI_03        1        -1.7 0.154 0.0188
## 5 CH30_4_RI_03        1        -1.6 0.155 0.0189
## 6 CH30_4_RI_03        1        -1.5 0.157 0.0190
```

Subsequently, we can take this resulting data frame and plot it with ggplot2, using the grouping variables to visually distinguish different curves:

```
# plot multiple curves simultaneously
multi.curve <- ggplot(tmp2, aes(x = temperature, y = mu)) +
geom_hline(yintercept = 0) +
    geom_line(aes(colour = factor(dilution))) + coord_cartesian(ylim =
c(-0.2,
    0.75), xlim = c(-2, 35)) + scale_x_continuous("Temperature (C)") +
    scale_y_continuous("Growth rate (1/day)") +
scale_colour_discrete("Dilution") +
    theme_bw()
multi.curve
```

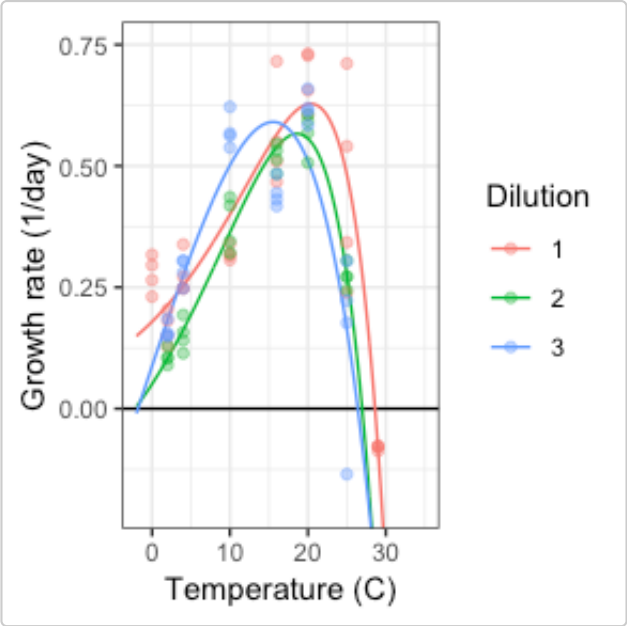


Finally, it is possible to ammend these multi-curve plots with the underlying observations used as the foundation of the regressions. This takes advantage of the fact that the original observations are reported in the `data` field of output from `get.nbcurve.tpc()`.

```
# recapitulate corresponding data frame of observations:
tmp2.obs <- nb.res %>% group_by(isolate.id, dilution) %>%
do(.$tpcs[[1]]$data)
head(tmp2.obs)
```

```
## # A tibble: 6 x 4
## # Groups:   isolate.id, dilution [1]
##   isolate.id   dilution temperature    mu
##   <fct>           <int>       <int> <dbl>
## 1 CH30_4_RI_03        1           0 0.296
## 2 CH30_4_RI_03        1           0 0.317
## 3 CH30_4_RI_03        1           0 0.231
## 4 CH30_4_RI_03        1           0 0.265
## 5 CH30_4_RI_03        1          10 0.340
## 6 CH30_4_RI_03        1          10 0.319
```

```
multi.curve + geom_point(data = tmp2.obs, aes(colour = factor(dilution)),
    alpha = 0.4)
```


0.75
0.50
0.25
0.00
Growth rate (1/day)
0
10
20
30
Temperature (C)
Dilution
1
2
3