Package Overview

Nathan Eastwood

15 March 2016

Overview

Hi, below I will give you a brief demo of how the package works. If you have any questions, then please feel free to email me.

Installation

If you haven't done so already, you will be able to install the package with the following command

```r
install.packages("<path to file>.tgz", type = "source", repos = NULL)
```

This is telling **R** to install from a binary file (`type = "source"`) and not from a repository such as CRAN (`repos = NULL`).

Accessing this vignette

You can access this vignette with the command `vignette("timedist")` once you have installed and loaded the package.

How the model works

You have seen the model working already, but essentially the main function you will need is `timedist()`, see `?timedist` for details. I am now going to show you 2 examples taken from some data you sent to me; both censored and time series data will be covered.

Time Series data

In order for you to construct the data yourself, you can run the following code:

```r
structure(list(Location = c("Plymouth", "Plymouth", "Plymouth",
                            "Plymouth", "Plymouth", "Plymouth", "Plymouth", "Plymouth", "Plymouth",
                            "Plymouth", "Plymouth", "Plymouth", "Plymouth", "Plymouth", "Plymouth",
                            "Plymouth", "Plymouth", "Plymouth", "Plymouth", "Plymouth", "Plymouth"
), Site = c("A", "A", "A", "A", "A", "A", "A", "A", "A", "A",
            "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A"), Time.in.seconds..x. = c(0L,
                                                                                            8L, 17L, 38
                                                                                            146L, 148L,
Proportion.of.ymax = c(0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3,
                       0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85,
                       0.9, 0.95, 1), Eye.estimate = c(0L, 0L, 0L, 0L, 0L, 0L, 0L,
                                                       0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L,
SPSS.fitted.CDF = c(0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L,
                    0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L), SPSS.fitted.PDF = c(0L,
                                                                 0L, 0L, 0L, 0L, 0L
                                                                 0L, 0L, 0L, 0L, 0L
```

But essentially it looks like this (once tidied):

```
##    Location Site   x    y
## 1  Plymouth    A   0 0.00
## 2  Plymouth    A   8 0.05
## 3  Plymouth    A  17 0.10
## 4  Plymouth    A  38 0.15
## 5  Plymouth    A  42 0.20
## 6  Plymouth    A 107 0.25
## 7  Plymouth    A 117 0.30
## 8  Plymouth    A 125 0.35
## 9  Plymouth    A 128 0.40
## 10 Plymouth    A 131 0.45
## 11 Plymouth    A 138 0.50
## 12 Plymouth    A 139 0.55
## 13 Plymouth    A 146 0.60
## 14 Plymouth    A 148 0.65
## 15 Plymouth    A 151 0.70
## 16 Plymouth    A 173 0.75
## 17 Plymouth    A 188 0.80
## 18 Plymouth    A 214 0.85
## 19 Plymouth    A 230 0.90
## 20 Plymouth    A 241 0.95
## 21 Plymouth    A 250 1.00
```

So the model works by giving it x and the y proportion values. This seemed to be the easiest way to keep things consistent between the time series and censored data. Of course one of the initial design specs was to get the function to calculate the proportions for you and to remove any rows containing 0s or repeat data. However the data that I have suggests that the time series data is cumulative whereas the censored data is no. of events for each time point? If I am wrong then correct me but it is easier to pre-sort your data in **R** or Excel than it is to work out what type of data you are using in a function and then sort it. Let me know if my assumption is incorrect, however and I can change this. Also let me know if you'd like a hand writing little functions to sort your data, these aren't really functions I would include in the package, however.

```
library(timedist)
lm1 <- timedist(TSdata, x = "x", y = "y", r = 0.01, c = 0.05, t = 140)
lm1
```

```
## Nonlinear regression model
##   model: y ~ 1 - (1 - (r/(1 + exp(-c * (x - t)))))^x
##    data: data
##         r         c         t
##   0.02610   0.01372 239.56283
##  residual sum-of-squares: 0.08052
##
## Number of iterations to convergence: 23
## Achieved convergence tolerance: 1.49e-08
```

If we take a look at the structure of the `lm1` object, you will see the data it contains

```r
str(lm1)
```

```
## List of 5
##  $ m       :List of 19
##   ..$ resid     :function ()
##   ..$ fitted    :function ()
##   ..$ formula   :function ()
##   ..$ deviance  :function ()
##   ..$ lhs       :function ()
##   ..$ gradient  :function ()
##   ..$ conv      :function ()
##   ..$ incr      :function ()
##   ..$ setVarying:function (vary = rep(TRUE, length(useParams)))
##   ..$ setPars   :function (newPars)
##   ..$ getPars   :function ()
##   ..$ getAllPars:function ()
##   ..$ getEnv    :function ()
##   ..$ trace     :function ()
##   ..$ Rmat      :function ()
##   ..$ predict   :function (newdata = list(), qr = FALSE)
##   ..$ moments   :'data.frame':   1 obs. of  6 variables:
##   .. ..$ mean    : num 136
##   .. ..$ variance: num 3659
##   .. ..$ sd      : num 60.5
##   .. ..$ skew    : num 0.117
##   .. ..$ kurtosis: num -0.213
##   .. ..$ entropy : num 7.94
##   ..$ ymax      : num 1
##   ..$ rss       : num 0.958
##   ..- attr(*, "class")= chr "nlsModel"
##  $ convInfo:List of 5
##   ..$ isConv     : logi TRUE
##   ..$ finIter    : int 23
##   ..$ finTol     : num 1.49e-08
##   ..$ stopCode   : int 1
##   ..$ stopMessage: chr "Relative error in the sum of squares is at most `ftol'."
##  $ data    : symbol data
##  $ call    : language minpack.lm::nlsLM(formula = as.formula(tdFormula), data = data, start = start,
##  $ control :List of 9
##   ..$ ftol   : num 1.49e-08
##   ..$ ptol   : num 1.49e-08
##   ..$ gtol   : num 0
##   ..$ diag   : list()
##   ..$ epsfcn : num 0
##   ..$ factor : num 100
##   ..$ maxfev : int(0)
##   ..$ maxiter: num 50
##   ..$ nprint : num 0
##  - attr(*, "class")= chr [1:2] "timedist" "nls"
```

As you can see, the main information you will be using is accessed via `lm1$m$...()` where `...` is a function name such as `resid`, e.g.

```
lm1$m$deviance()
```

```
## [1] 0.08052405
```

This is the standard output from the `minpack.lm::nlsLM` function, only I have added an additional class, `timedist` and the `moments`, `ymax` and `rss` (note that the rss is the corrected rss - the same as what SPSS provides). These are accessed in the following way

```
lm1$m$moments
```

```
##      mean variance       sd      skew  kurtosis  entropy
## 1 135.776 3659.476 60.49361 0.1174507 -0.212781 7.941479
```

```
lm1$m$ymax
```

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

```
lm1$m$rss
```

```
## [1] 0.9581693
```

You can calculate moments yourself, however, with the `tdMoments()` function of the individual functions, `tdMean`, `tdVariance`, `tdSkew`, `tdKurtosis` and `tdEntropy`.

```
tdMoments(r = 0.01, c = 0.05, t = 140)
```

```
##       mean variance       sd     skew kurtosis entropy
## 1 156.4883 5370.975 73.28694 2.940985  14.0924  7.6902
```

Censored data

```
censored <- structure(list(x = c(91, 92, 95, 97, 98, 99, 100, 101, 102, 103,
                                 105, 106, 108), y = c(0, 0.02, 0.07, 0.2, 0.26, 0.33, 0.55, 0.7,
                                                       0.82, 0.85, 0.93, 0.98, 1)), .Names = c("x", "y"

censored
```

```
##      x    y
## 1   91 0.00
## 2   92 0.02
## 3   95 0.07
## 4   97 0.20
## 5   98 0.26
## 6   99 0.33
## 7  100 0.55
## 8  101 0.70
## 9  102 0.82
## 10 103 0.85
## 11 105 0.93
## 12 106 0.98
## 13 108 1.00
```

We can fit the model as normal

```
lm2 <- timedist(censored, x = "x", y = "y", r = 0.05, c = 0.5, t = 102)
lm2
```

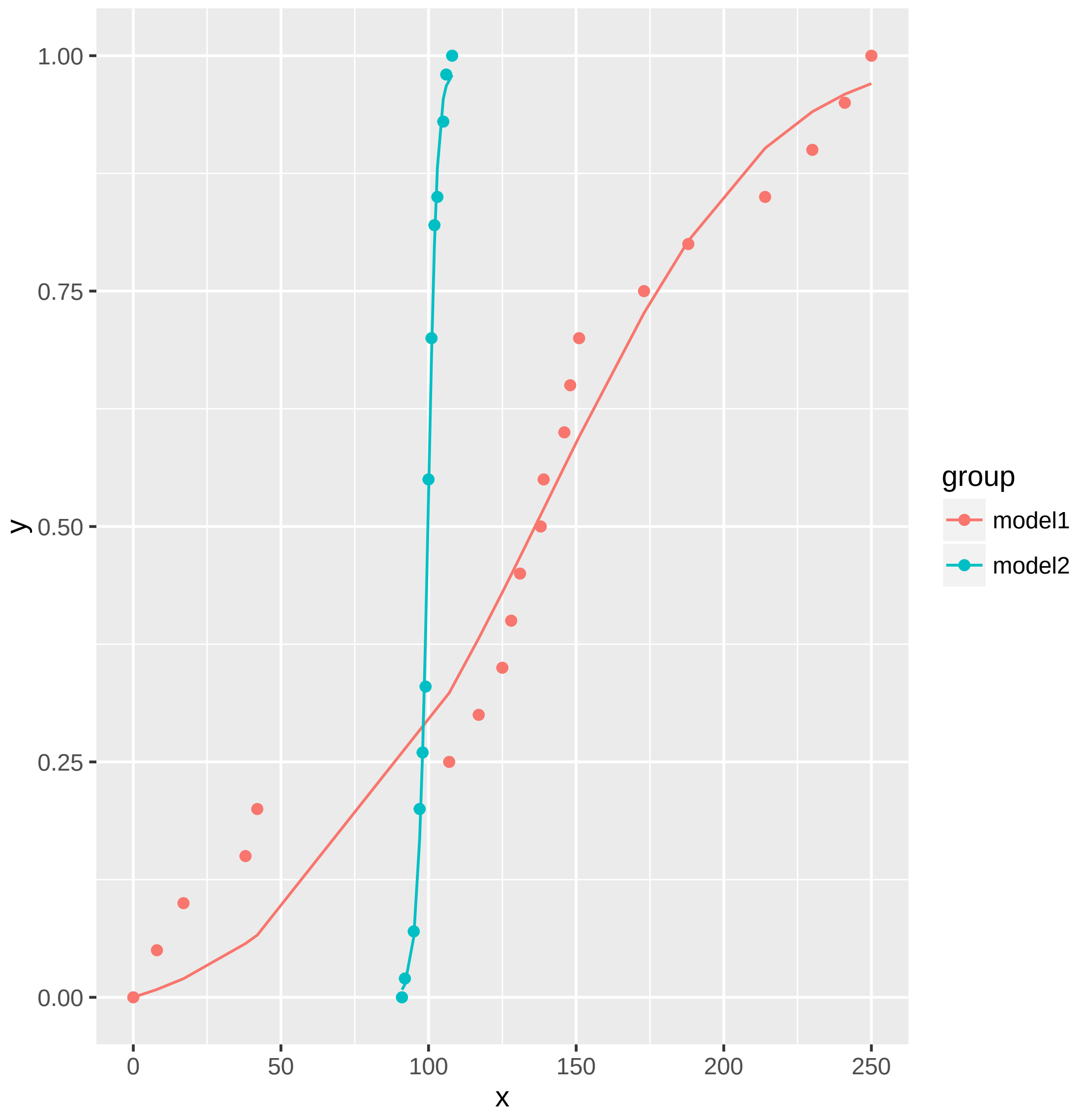
```
## Nonlinear regression model
##   model: y ~ 1 - (1 - (r/(1 + exp(-c * (x - t)))))^x
##    data: data
##         r         c         t
##   0.03754   0.51844 102.65082
##  residual sum-of-squares: 0.007535
##
## Number of iterations to convergence: 7
## Achieved convergence tolerance: 1.49e-08
```

Again, you should find that this fit matches the Excel sheet you sent me.

Plotting

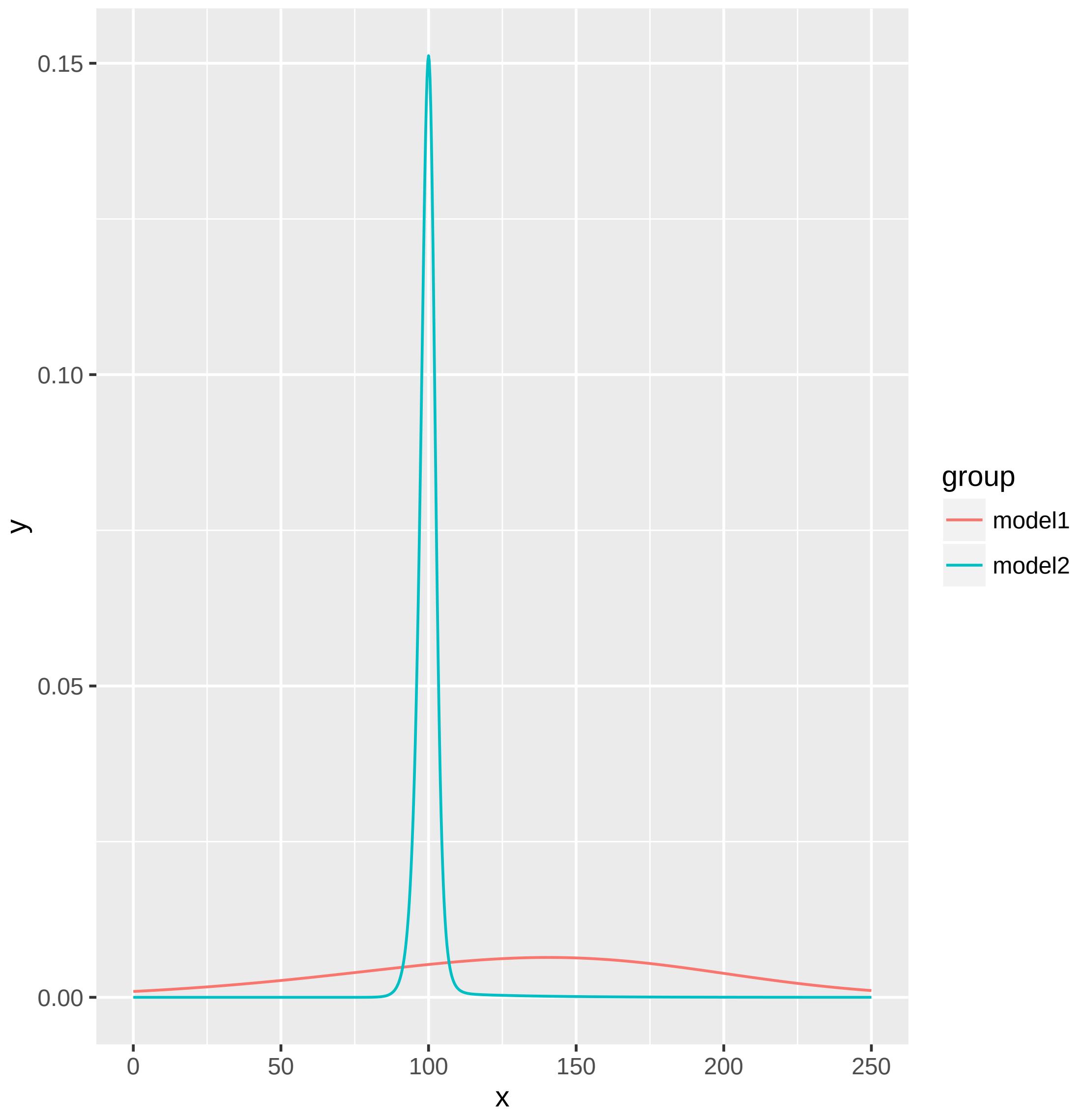
The PDF and CDF plots are generated for a model, or list of models, using the `ggtimedistPDF()` and `ggtimedistPDF()` functions.

```
ggtimedistCDF(lm1, lm2)
```



And the pdfs...

```
ggtimedistPDF(lm1, lm2)
```



These plots have been written in `ggplot2` and are therefore highly customisable.

Possible further work

I looked into the possibility of doing multiple models in one go again and asked around for some advice. Basically, it is possible - as I showed the other day - but it isn't nice/easy. The only other solution we could come up between us is to use a data.frame of starting values instead of nested lists for a cleaner looking UI. You would then join this data.frame to your data by conditioning on what to join by. This would look like this:

```
starts = data.frame(Run = unique(DNase$Run),
                    xmid_start = rnorm(11, sd = 0.1),
                    scale_start = 1 + rnorm(11, sd = 0.1))
starts
```

```
##    Run   xmid_start scale_start
## 1    1 -0.007980969   0.8875778
## 2    2 -0.009923305   0.9084117
## 3    3  0.052049068   0.8803289
## 4    4 -0.052162568   1.0800279
## 5    5  0.062156617   0.8691389
## 6    6 -0.107091002   0.9954350
## 7    7 -0.183568647   1.0548366
## 8    8 -0.055782316   1.0514411
## 9    9 -0.009712246   0.9301112
## 10  10 -0.114279886   1.1337328
## 11  11  0.044827924   0.9216297
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

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

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

```
exampleData %>% glimpse
```

```
## Observations: 176
## Variables: 3
## $ Run     (fctr) 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,...
## $ conc    (dbl) 0.04882812, 0.04882812, 0.19531250, 0.19531250, 0.3906...
## $ density (dbl) 0.017, 0.018, 0.121, 0.124, 0.206, 0.215, 0.377, 0.374...
```

Joining them together gives…

```
exampleData <- exampleData %>%
    left_join(starts) %>%
    group_by(Run)
```

```
## Joining by: "Run"
```

```
exampleData
```

```
## Source: local data frame [176 x 5]
## Groups: Run [11]
##
##       Run       conc density   xmid_start scale_start
##    (fctr)      (dbl)   (dbl)        (dbl)       (dbl)
## 1       1 0.04882812   0.017 -0.007980969   0.8875778
## 2       1 0.04882812   0.018 -0.007980969   0.8875778
## 3       1 0.19531250   0.121 -0.007980969   0.8875778
## 4       1 0.19531250   0.124 -0.007980969   0.8875778
## 5       1 0.39062500   0.206 -0.007980969   0.8875778
## 6       1 0.39062500   0.215 -0.007980969   0.8875778
## 7       1 0.78125000   0.377 -0.007980969   0.8875778
## 8       1 0.78125000   0.374 -0.007980969   0.8875778
## 9       1 1.56250000   0.614 -0.007980969   0.8875778
## 10      1 1.56250000   0.609 -0.007980969   0.8875778
## ..    ...        ...     ...          ...         ...
```

You can then fit multiple models using

```
models <- exampleData %>%
  do(model = nls(density ~ 1/(1 + exp((xmid - log(conc))/scal)),
             data = .,
             start = list(xmid = first(.$xmid_start),
                          scal = first(.$scale_start)),
             algorithm = "plinear"))
models
```

```
## Source: local data frame [11 x 2]
## Groups: <by row>
##
##        Run    model
##     (fctr)    (chr)
## 1       10 <S3:nls>
## 2       11 <S3:nls>
## 3        9 <S3:nls>
## 4        1 <S3:nls>
## 5        4 <S3:nls>
## 6        8 <S3:nls>
## 7        5 <S3:nls>
## 8        7 <S3:nls>
## 9        6 <S3:nls>
## 10       2 <S3:nls>
## 11       3 <S3:nls>
```

Obviously the output isn't nice, but you can access it easily enough

```
models$model[[1]]
```

```
## Nonlinear regression model
##   model: density ~ 1/(1 + exp((xmid - log(conc))/scal))
##    data: .
##  xmid  scal  .lin
## 1.417 1.149 2.351
##  residual sum-of-squares: 0.007244
##
## Number of iterations to convergence: 6
## Achieved convergence tolerance: 6.622e-07
```

The good thing about this method is that setting up `r`, `c` and `t` is much neater than my first solution. If this is something that interests you then this would have to go into the next version of the package as it would potentially require a rewrite of the plots too.

What I need from you

- Test, test, test. I do not expect the package will work first time. If things do break, then to help me out as much as possible, please save your current working environment with `save.image()`, take screen shots, save your history with `savehistory()`, let me know what the error files say and write a short description of what it is you were doing when it went wrong. Basically I need you to provide me with as much information as possible in order for me to recreate the error and debug the code.
- Let me know what I have missed. I am fairly confident the package does most of what you would like it to do. If, however, something doesn't work the way you want it to then let me know.
- I need to know what the `S` parameter is used for in the PDF function and whether or not it should be in any other functions, such as the moments. I also need to know whether it should even be included in the PDF function? (see `?tdPDF`). Currently the pdf plot just uses `S = 1` by default.
- I need you to write a short description of the package, 2-3 sentences long. Of course, I could do this no problem.
- Send me a reference to include in the `?timedist` help file for the original model's paper.
- Re-write this vignette. Obviously these are instructions that are very specific to you. But really this vignette should describe how to use the package in a more general, but technical, way. It should describe how your model works and how to implement it with this package. It would be useful for you to look at other vignettes from packages for specific models. If you know `rmarkdown` then great, please write it in that, otherwise just write it in word but include the code in a separate .R script.