# Vignette EnzymeAssay

*Alisandra Kaye Denton*

*2015-04-07*

The goal of the EnzymeAssay package is to facilitate enzyme assay data analysis, curation and quality control. The idea being that blindly taking the initial slope is a less-than-optimal solution for determining enzyme activity, while manually plotting and range selecting in a spredsheet program can be excedingly tedious. Specifically, the package was designed for the colorimetric assays commonly used in the Institute of Plant Biochemistry, Heinrich-Heine University, Duesseldorf (http://www.plant-biochemistry.hhu.de/). However, it should work for any assay where either the substrate or the product can be measured over time, and the region of maximum slope is most desirable for estimating activity. Source code available at (https://github.com/alisandra/enzyme_assay).

```
# to install and load package
install.packages('path/to/EnzymeAssay', repos = NULL, type = "source")
```

```
library(EnzymeAssay)
```

## Enzyme_Assay class

The Enzyme_Assay class holds the assay data, and facilitates logical plotting and curation there of. The input required to make an Enzyme_Assay object is a matrix of assay measurements, with columns denoting different samples (wells) and rows corresponding to different measurement timepoints. Additionally, a vector of the relevant time points is required.

```
mini.data <- enzyme_assay.data[,1:24]  # subset of example assay data
mini.data[1:6,1:6]
```

```
##      A1    A2    A3    A4    A5    A6
## 3 0.131 0.125 0.127 0.130 0.125 0.127
## 4 0.135 0.129 0.130 0.136 0.129 0.132
## 5 0.139 0.133 0.134 0.142 0.134 0.138
## 6 0.143 0.136 0.138 0.148 0.138 0.144
## 7 0.148 0.140 0.142 0.153 0.142 0.149
## 8 0.151 0.145 0.145 0.158 0.148 0.156
```

```
head(enzyme_assay.time)  # example assay timepoints
```

```
## [1] "0:00:00" "0:00:44" "0:01:28" "0:02:12" "0:02:56" "0:03:40"
```

```
ea <- enzyme_assay(mini.data, enzyme_assay.time)  # constructor function
```

There are several input options. The package defaults to the time format exported by the plate reader ("H:MM:SS"). However, it converts this to and uses seconds internally; and also accepts seconds directly when the parameter `seconds = TRUE` is set.

```
enzyme_assay.seconds <- plate_reader_seconds(enzyme_assay.time)
head(enzyme_assay.seconds)
```

```
## 0:00:00 0:00:44 0:01:28 0:02:12 0:02:56 0:03:40
##       0      44      88     132     176     220
```

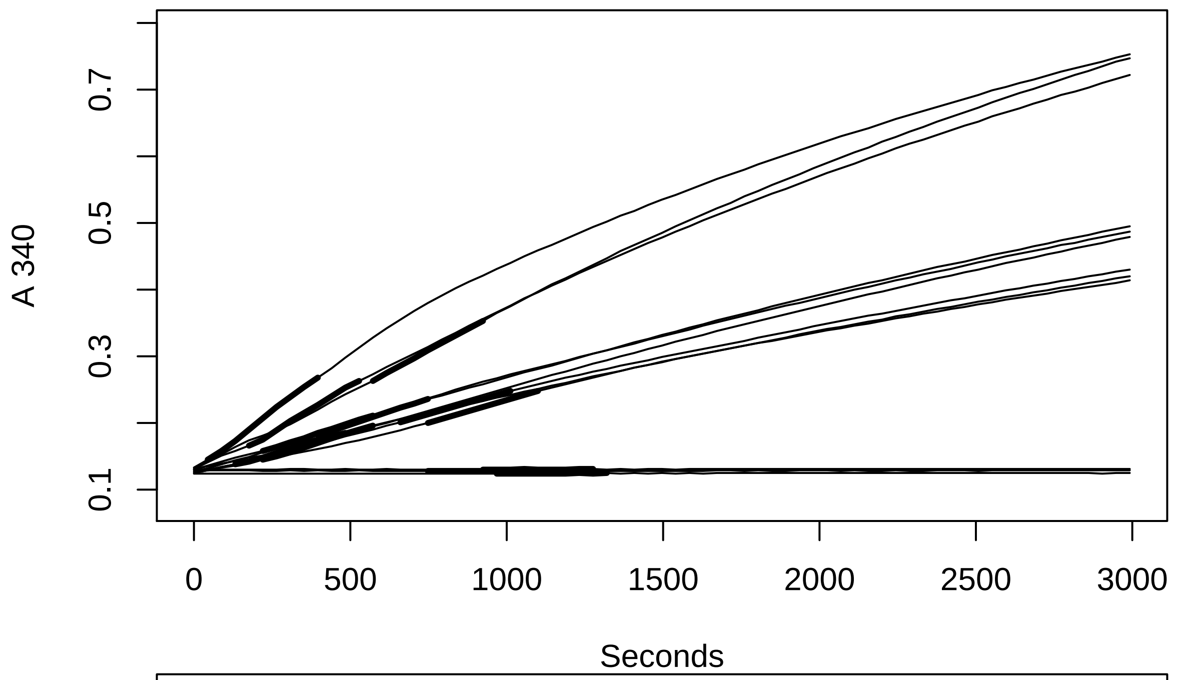
```
ea <- enzyme_assay(mini.data, enzyme_assay.seconds, seconds=TRUE)
```

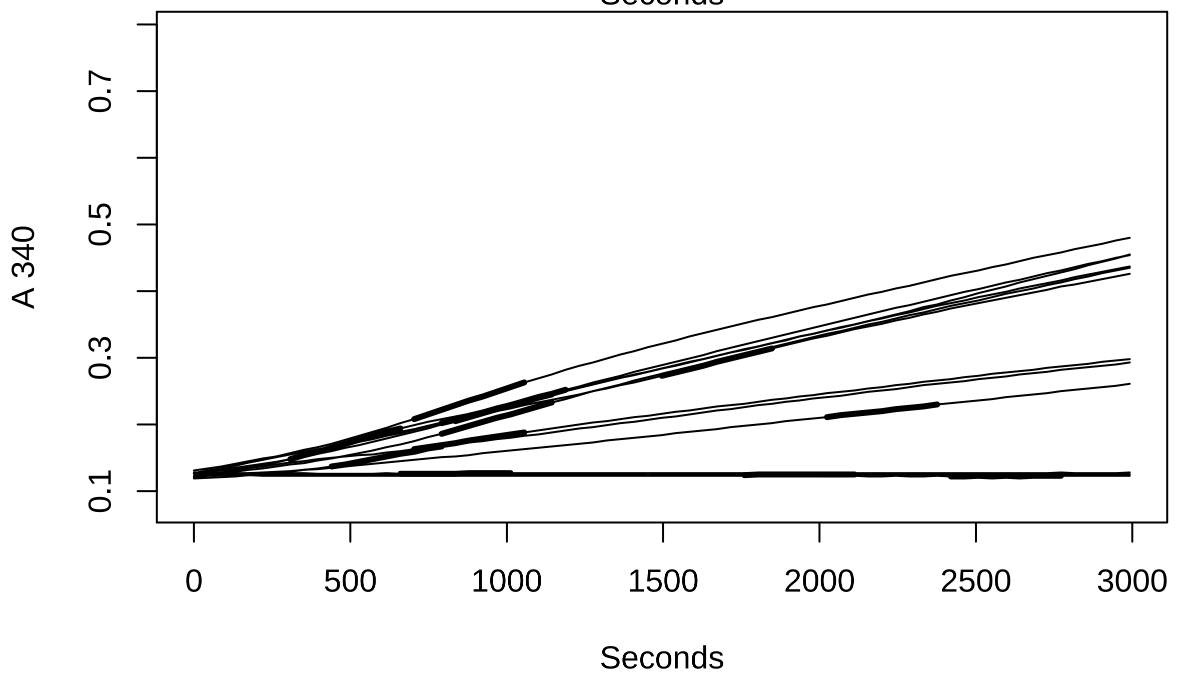
Three more parameters affect how the constructor function finds ranges of maximum slope. The parameter `windowsize` tells the function the width of ranges to consider (default = 8). The parameter `decreasing = TRUE` should be set if it is a decreasing assay; that is, if positive enzyme activity is measured by the rate of decrease in absorbance (or other abundance value). Finally, two methods are implemented: `method = "individual"` (default) calculates the range of maximum slope for each column independently, while `method = "together"` calculates the range of maximum average slope accross all assays.

## Plotting

The Enzyme_Assay class has a specific method of the generic plot function, which produces a simple line plot with the ranges of maximum slope shown in bold.

```
plot(ea)
```





The plot method accepts many standard plotting parameters, and one additional parameter `at_once` which indicates the number of lines per plot. This defaults to 12, as this is the width of a 96 well plate.

## Curating

An Enzyme_Assay object has two associated curation methods. In `enzyme_assay.curate` the lines are plotted as in the plot function, except that all but one is greyed out. The user is then prompted to enter `k` to keep curent value or enter a new value in seconds for both the start and stop of the range of the foreground line. This is repeated for every line, and the curated ranges are saved to the slot `@curated_range`. Entering `c` for cancel saves the automatically generated ranges with the changes included so far into the slot `@curated_range`. Unfortunately, no elegant resume method has been implemented, however, the user can restart the curation based on their previous curation using the parameter `status = "curated"`.

```r
eaCurated <- enzyme_assay.curate(ea)
# to resume after canceling with 'c'
eaRecurated <- enzyme_assay.curate(eaCurated, status = "curated")
# this will start from the changes saved so far in eaCurated
# the user may wish to keep pressing 'k' until reaching their previous spot
```

Sometimes a brief glance at the plot is sufficient to determine the changes that need to be made, or the user may prefer to record changes with a pen and paper or spreadsheet program. For all such cases the `enzyme_assay.curate_batch` function may be helpful. The user should take care that the ranges are stored as measurement index and not as seconds.

```r
# retrieve the automatically determined ranges into a matrix
ranges_under_curation <- ea@auto_range
# modify, in this case, set the negative controls (in columns 10-12)
# to match the range of column 1
ranges_under_curation[,10:12] <- ranges_under_curation[,1]
# add back to Enzyme_Assay object
eaCurated <- enzyme_assay.curate_batch(ea, new_range = ranges_under_curation)
```

## Retrieving data

No special methods have been implemented for the retrieval of data from the Enzyme_Assay class. Slots should be accessed directly by the `@` operator.

```r
eaCurated@auto_slope  # retrieves automatically calculated slopes
eaCurated@auto_range  # retrieves associated automatically calculated ranges
eaCurated@curated_slope  # retrieves curated slopes
eaCurated@curated_range  # retrieves associated curated ranges
```

## Import and Export

As this package is intended to be usable by begining R-users, here is a quick note on getting data in and out of R. There are functions in R to both read from, and write to the standard csv format.

```r
my_data <- read.csv('my_file.csv')  # basic import to data.frame (table)
my_data <- read.csv('my_file.csv', row.names = 1)  # automatically names rows
my_data <- read.csv2('my_file.csv')  # imports 'german' csv (';' separated, ',' for decimal)
# read.csv2 can be quite helpful importing data from the old plate reader laptop

write.csv(eaCurated@curated_slope, file="my_curated_slopes.csv")
write.csv(eaCurated@curated_range, file="my_curated_ranges.csv")
```