User Guide for Real-Time Kinetics (PGO Assay) Data Analysis in R

August 18th, 2020

Written by Alina Arvisais (aarvisai@uwaterloo.ca) for use within the Rose Lab

Table of Contents:

Acknowledgements:

This script utilizes some modified functions from the R OpenSci package ‘plater’ version 1.0.2. This work was originally created by Sean Hughes to ‘Read, Tidy, and Display Data from Microtiter Plates’ and is shared under a GPL-3 license. Since this script uses, in part, a modified form of this work it is also bound by the GPL-3 license. A copy of this license is included in the distribution folder for this tool. Under this license, this script is free code: you can redistribute it and/or modify it under the terms of the GNU General Public License (GPL) as published by the Free Software Foundation, either version 3 of the license, or (at your option) any later version.

Preparing Your R Environment:

To properly install and use R and R-Studio on your devices you will need to first install R, then install RStudio. RStudio is an integrated development environment (IDE) for R which provides a console, syntax-highlighting editor and tools for plotting, debugging, and workspace management. Therefore, while R can be used without R-Studio it is highly recommended to also use R-Studio or another appropriate R IDE.

This code was developed using R version 3.6.1 (2019-07-05).

Installing R:

R can be downloaded from the Comprehensive R Archive Network (CRAN) which is a collection of sites that carry R distributions, extensions, and documentation.

For Windows:

1. Download the setup file for R from the following link.
2. Open the .exe file and install R.

For Mac:

1. Download the appropriate version of the .pkg file from the following link.
2. Open the .pkg file and install R.

For Linux:

1. For complete R installation in Linux, follow the instructions in the following link.

Installing R Studio:

1. Choose the appropriate installer file for your system from the following link to download R-Studio.
2. Once the download is complete, open and run it to install R-Studio.

Required Packages:

A package is a way to organize a work of code and share it with others through repositories where other people can install them from. CRAN is the official repository coordinate by the R foundation. The packages used in this script can all be installed from and found on CRAN.

The packages required by the script are:

1. tidyr
2. data.table
3. purr
4. ggplot2
5. dplyr
6. zoo

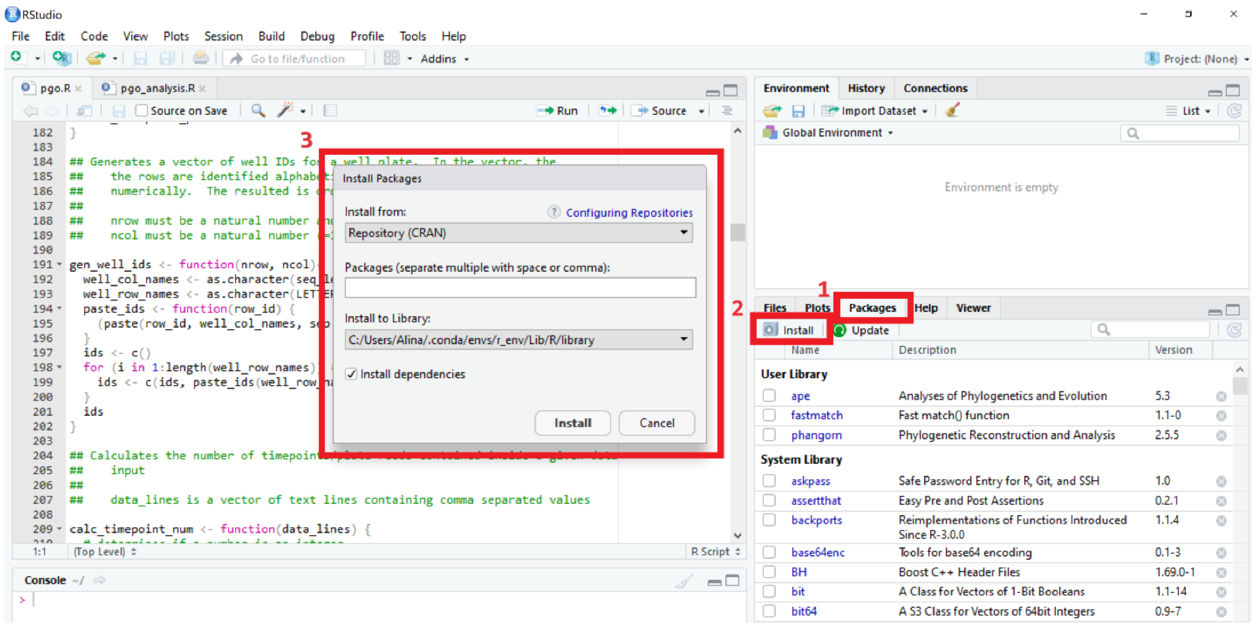
These must be installed before running the script, since the beginning includes the library call. The library call loads these packages in order to use functions contained in these packages throughout the script.

Installing Packages:

To install packages for use in R on your computer, either use:

1. The call `install.packages(“package_name”)` and then press the Enter/Return key.
2. Install them through R-Studio by:
    - Running R-Studio
    - Clicking on the Packages tab in the bottom right window. Clicking install and seeing the dialog box appear.

- In the install packages dialog box, type the name of the desired package and click install. This will install the package or give a list of matching packages based on the name to select.



```
}

## Generates a vector of well IDs fo
##    the rows are identified alphabe
##    numerically.  The resulted is o

##    nrow must be a natural number a
##    ncol must be a natural number (

gen_well_ids <- function(nrow, ncol)
  well_col_names <- as.character(seq_le
  well_row_names <- as.character(LETTER
  paste_ids <- function(row_id) {
    (paste(row_id, well_col_names, se
  }
  ids <- c()
  for (i in 1:length(well_row_names))
    ids <- c(ids, paste_ids(well_row_na
  }
  ids
}

## Calculates the number of timepoint
##    input
##
##    data_lines is a vector of text lines containing comma separated values

calc_timepoint_num <- function(data_lines) {
```

Install Packages

Install from: Repository (CRAN) (Configuring Repositories)

Packages (separate multiple with space or comma):

Install to Library: C:/Users/Alina/.conda/envs/r_env/Lib/R/library

[✓] Install dependencies

Install | Cancel

Name	Description	Version
User Library		
ape	Analyses of Phylogenetics and Evolution	5.3
fastmatch	Fast match() function	1.1-0
phangorn	Phylogenetic Reconstruction and Analysis	2.5.5
System Library		
askpass	Safe Password Entry for R, Git, and SSH	1.0
assertthat	Easy Pre and Post Assertions	0.2.1
backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.4
base64enc	Tools for base64 encoding	0.1-3
BH	Boost C++ Header Files	1.69.0-1
bit	A Class for Vectors of 1-Bit Booleans	1.1-14
bit64	A S3 Class for Vectors of 64bit Integers	0.9-7

How to Run the Analysis (With User Input Examples):

Beyond the required packages, conducting the analysis requires a total of five R files. Only one of these must be open in RStudio while the others must simply be in the working directory. The files which much be present in the working directory or run fully in RStudio include:

- platetidyr.R
- formatr.R
- rtkin.R
- linearrange.R

The file which must be open in RStudio while running the analysis is distributed with the name pgo_analysis and includes all the variable assignments and function calls required to complete the analysis. These can be completed by adding the required user inputs describing a test plate and then run in its entirety to obtain the analyzed results.

All these files may be found hosted on github at: https://github.com/alinaarvisais/96well-real-time-kinetics.

Recommendation: While you could run the whole pgo_analysis script in one go, it is recommended to run section by section, so that if you obtain an error it is easier to locate where it is occurring. It is also recommended to save this file with a descriptive file name so that you can replicate and view the analysis at any later date, much like you would save an Excel file.

Note: To determine your current working directory, you may run the command `getwd()`. If needed, you can set the working directory to a specific path using the command `setwd(“path”)` where “path” is replaced with the path you wish to use. Ensure you are able to write new files to the chosen working directory since the script output consists of .png and .csv files.

0.0 Importing the Required Functions

This section includes source files to each of the aforementioned required R scripts containing the function that complete the data analysis. To indicate to R that you wish to use these functions in this current script, simply run the lines containing the source calls. Without doing so, the required functions and variables will not be loaded into the working environment.

1.0 Importing and Tidying the Plate-Shaped Data

This section does the work of converting plate-shaped data to 'tidy' data. Tidy data in this case refers to a table wherein each row represents a different well and each column represents a different timepoint at which well absorbance as measured.

The first step is to load in the raw data by specifying the path where it can be found:

```
raw_data <- "C:/Users/Alina/Desktop/example_data.txt"
```

Next, you must import the data by calling the function 'read_data' on the raw data.

```
imported_data <- read_data(raw_data)
```

In the process of importing the data, it will create a new file named 'comma_sep.csv'. This simply represents a comma and not space delimited version of the raw data. There is no need to save this file if you ever want to repeat the analysis, if left untouched will simply be re-written next time you run the analysis.

The last step is to convert the imported raw data to tidy data by calling the function 'tidy_plate_data' on the imported data.

```
tidy_data <- tidy_plate_data(imported_data) # must be named tidy_data
```

Note: this must be assigned to a variable named tidy_data to allow proper accessing of data by later functions.

2.0 Organizing the Standard Curve Data

This section is used to assign the standard curve data based on where each standard replicate can be found on the plate. There is also the option to export the table of organized standard data as a .csv file.

For each standard assign a variable to a list containing the well ids for the three replicates. As an example:

```
std1 <- list("A1", "A2", "A3") # 2 mM Glucose

std2 <- list("A4", "A5", "A6") # 1 mM Glucose

std3 <- list("A7", "A8", "A9") # 0.5 mM Glucose

std4 <- list("A10", "A11", "A12") # 0.25 mM Glucose
```

Note: If there are not three replicates for a standard, the unnecessary well id gaps may be replaced with 'NA'. Additionally, if there is a known outlier and sufficient data points remaining, replace that well id with 'NA'

To actually create the standard table, the function 'make_standard_table' must be called on the standards, and the tidy_data. This function also has an optional parameter, 'export' whose default value is TRUE. If export = TRUE, the standard table will be exported as a .csv, and if export = FALSE the standard table still be created and stored in the R environment but not exported.

```
standard_table <- make_standard_table(std1, std2, std3, std4, tidy_data,
                                      export = TRUE)
```

The standards must be input in the same order.

3.0 Organizing the Enzyme Assay Data

The first step is to build test identifiers for each different set of tests conducted on the plate. A test is any different combination of enzyme, substrate, or substrate concentration. Test identifiers are a `list("Enzyme_subs", subs_conc_mM, list(well IDs))`. As many as needed can be created. For example:

```
t1 <- list("NtSI_malt", 1, list("B1", "B2", "B3"))

t2 <- list("NtSI_malt", 0.5, list("B4", "B5", "B6"))

t3 <- list("NtSI_malt", 0.25, list("B7", "B8", "B9"))

t4 <- list("NtSI_malt", 0.125, list("B10", "B11", "B12"))
```

Next the test identifiers must be bound together in a list:

```
test_identifiers <- list(t1, t2, t3, t4)
```

Then, to organize tables of the tidied and extracted test data, run make_test_data_tables on the test identifiers. If you wish to export the test tables with the raw absorbance values for manual analysis set export = TRUE, otherwise set export = FALSE.

```
test_data <- make_test_data_tables(test_identifiers, export = TRUE)
```

4.0 Assign the Blank Wells (Enzyme + Substrate Blanks)

This step is optional if there are no blanks on the plate. But as it is good scientific practice, if blanks are conducted this can be used to assess and apply their effect on the test identifiers.

The plate assay conducted may include enzyme and substrate blanks. If a blank average absorbance is above a given threshold, you may wish to halt the analysis for tests that relate to that blank. Or if above a given threshold, you may alternatively wish to have its value subtracted from the test absorbance value. For the first response, the script will define it as a “substrate” blank, while the second response is that taken to an “enzyme” blank.

To assign which wells belong to which blanks, create as many blank identifiers as are required. Blank identifiers are of the form `list("blank_type", "name", list(well IDs))`. Where “blank_type” must be one of: “substrate” or “enzyme”. The name must also match the same name formatting as used in the test identifiers (i.e. if you abbreviate maltose to malt in the test identifiers, list the maltose blank as malt). Next, bind the blank identifiers together in a list.

```
b1 <- list("substrate", "malt", list("C1", "C2", "C3"))

b2 <- list("enzyme", "NtSI", list("C1", "C2", "C3"))

blanks <- list(b1, b2) # add as many or as few as required
```

Lastly, run ‘apply_blanks’ on the test_identifiers and blanks, with a set threshold. The default for the threshold is 0.1.

```
test_identifiers_blanked <- apply_blanks(test_identifiers, blanks, 0.1)
```

This will return a message printed to the console indicating that no actions have been taken, tests with the substrate blanks above a threshold are removed, or tests with enzyme blanks above a threshold having a blank offset (average absorbance of the blank replicates at that timepoint) subtracted.

5.0 Analyze the Test Data

Before being able to correlate the absorbance values to a glucose concentration, the parameters of the standard curve must be determined. This is done by calling ‘make_concentration_curve’ on the standard table. This function has two option parameters whose default is TRUE. If export or plot are TRUE, a .png containing the plotted standard curve data and a .csv containing the linear regression parameters are saved. If you do not wish to save either or both of these, set their values to FALSE.

```
std_curve <- make_concentration_curve(standard_table, export = TRUE, plot =
                                      TRUE)
```

Next, the test data can be analyzed by converting to glucose concentration, averaging, then normalizing to time t = 0, and performing linear regression on the resulting progress curve. This is done by calling 'calc_test_kinetics' on the test_identifiers, the standard curve. This function also gives the option of exporting both the linear regression parameters for the kinetics data, the analyzed data with averages and standard deviation, or the plot.

```
analyzed_kinetics <- calc_test_kinetics(test_identifiers_blanked, std_curve,
                                        export = TRUE, plot = TRUE)
```

6.0 Modify to Remove Non-Linear Portions and Outliers

The test data may include outliers known from the assay set-up, or visible on the plot due to variation in glucose concentration, and/or based on large standard deviation. If one assay well has been erroneously prepared it is better to set its well id as 'NA' in the test_identifiers. The time range of the assay may also exceed the linear range for the given enzyme and substrate concentration. In this step, non-linear portions and outlying timepoints can be removed.

For each test where a linear range needs to be modified, a linear identifier must be a created where a linear range is list("test_name", list(lin_min, lin_max, outliers)). Where "test_name" is the name for that progress curve either in the exported data file or plot (i. e. "TestIDName_SubsConc"). 'Lin_min' is the minimum timepoint to be included, and 'lin_max' is the maximum timepoint to be included. 'Outliers' is a list of timepoints to be excluded and may be left empty.

These must be bound together into a list of linear identifiers.

```
lin1 <- list("NtSI_malt_1", list(0, 60, list()))
```

```
lin2 <- list("NtSI_malt_0.5", list(10, 60, list()))

lin3 <- list("NtSI_malt_0.25", list(0, 30, list()))

lin4 <- list("NtSI_malt_0.125", list(0, 60, list(2, 4)))

linear <- list(lin1, lin2, lin3, lin4)
```

Lastly, the linear ranges must be applied by calling the function ‘calc_mod_kinetics’ on the analyzed_kinetics, linear identifiers and choosing whether or not to export either, both, or none of the new progress curve plot or modified kinetics data (shows were ‘NA’s have been introduced based on linear identifiers).

```
linear_kinetics <- calc_mod_kinetics(analyzed_kinetics, linear, export =
                                     TRUE, plot = TRUE)
```

Optional: For an estimate of the linear range you may use:

```
test_name <- "NtSI_malt_1" #change as required

suggested_points <- estimate_lin_range(analyzed_kinetics[[1]], test_name)

print(suggested_points)
```

But, in many cases the human eye is better at detecting the linear range.

Exported Files:

1. comma_sep.csv
    - Required
    - Comma separated version of the text delimited spectrophotometer data
2. standards.csv
    - Optional
    - Table containing the obtained absorbance data for the standards at t = 0
3. test_data.csv
    - Optional
    - Table containing the obtained absorbance data for each tests at all timepoints, formatted for continued manual analysis
4. standard_curve_params.csv
    - Optional
    - Table containing the line equation and R^2 output from running a linear regression on the standard data
5. standard_curve.png
    - Optional
    - Plot of the standard curve containing all replicate data and averages
6. test_params.csv
    - Optional
    - Table containing the line equation and R^2 output from running a linear regression on each set of analyzed test data
    - This would be used for future Michaelis Menten kinetics analysis since the parameter m = velocity.
7. analyzed_data.csv
    - Optional
    - Table containing the analyzed data (converted to glucose, normalized, averaged, standard deviation calculated) for each test and timepoints

- Particularly useful for looking at variation within data points

8. kinetics_plot.png
    - Optional
    - Plot of the progress curves for each test containing all replicate data and averages.
9. test_params_linear.csv
    - Optional
    - Table containing the line equation and R^2 output from running a linear regression on each set of analyzed test data
    - This would be used for future Michaelis Menten kinetics analysis since the parameter m = velocity
    - Calculated after linear range of the tests have been taken into account
10. analyzed_test_linear.csv
    - Optional
    - Table containing the analyzed data (converted to glucose, normalized, averaged, standard deviation calculated) for each test and timepoints
    - Particularly useful for looking at variation within data points
    - Calculated after linear range of the tests have been taken into account
11. kinetics_plot_linear.png
    - Optional
    - Plot of the progress curves for each test containing all replicate data and averages
    - Non-linear ranges of the tests have been removed from the line of best fit, but all replicate data is still plotted

Analysis Process Description:

Generating the standard curve:

To generate the concentration curve, the process the script undertakes once all the commands are run can be described as follows:

1. Obtain the absorbance values of the three replicates wells for each standard at t = 0.
2. Average the absorbance values of the three replicate wells.
3. Perform a linear regression using the R function ‘lm’ comparing y = average absorbance to x = glucose concentration (mM), to return m, b, and R^2.
4. Plot the line of best fit and all well replicate data.

Generating the progress curve(s):

To generate the progress curves, the script undertakes the following process for each test set:

1. Obtain the absorbance values of the three replicate wells for the test at all timepoints.
2. If an enzyme blank has an absorbance difference greater than or equal to the given threshold, the average absorbance value of all three blank replicates at a given timepoint is subtracted from each replicate test well at the same timepoint. This is done for all timepoints. If a substrate blank has an absorbance difference greater than the given threshold, the test data is not analyzed. If the blank absorbance differences are smaller than the given threshold it is not considered in the analysis. A message is printed indicating the effects applied or that not effect has been applied.
3. Using the parameters m and b from the concentration curve, convert the absorbance of each well at every time-point to glucose concentration.
4. Normalize each replicate well to the value measured at t=0 by subtracting the glucose concentration of that well at t = 0 from all following well timepoint glucose concentrations.

5. Average the normalized glucose concentrations of all three wells at each individual timepoint. Also calculate the standard deviation of the normalized glucose concentrations of all three wells at each individual timepoint.
6. Perform a linear regression using the R function ‘lm’ comparing y = average absorbance to x = time (min) to return m (velocity), b, and R^2.
7. Plot the line of best fit and all well replicate data.
8. If there are outlying timepoints or timepoints in the non-linear range, after having the user identify these, they are converted to ‘NA’ in the data and omitted from calculation of the average glucose concentration, standard deviation, and linear regression.
9. Re-plot the line of best fit over the linear range, but still include all well replicate data.

Modifying the Script:

Modifying the Analysis:

Here is a list of all defined constants or function locations where changing the value can be useful to apply the same analysis to varying kinetic assays.

To accept input from a slightly different formatted .txt file:

1. *Line 40, platetidyr.R:* change skip to whichever of lines the file header is contained in.
2. *Line 46, platetidyr.R:* change last_col to the last column number (indexed from 1) that contains absorbance data.
3. *Line 48, platetidyr.R:* change first_col to the first column number (indexed from 1) that contains absorbance data.

To tidy data from a plate larger or smaller than 96-wells:

1. *Line 76, platetidyr.R:* change num_rows to the appropriate number of rows in the plate size you are using.
2. *Line 80, platetidyr.R:* change num_col to the appropriate number of columns in the plate size you are using.
3. *Line 328, formatr.R:* change the input parameters to the gen_well_ids to the appropriate number of rows and columns for the plate being run.

To adjust the time difference between timepoints:

1. *Line 170, platetidyr.R:* change timepoint_dif to the time difference between different spectrophotometer reads in the data. Just remember the units for your final velocity ouput!

To change the number of standards concentrations used:

1. *Line 32-67, formatr.R:* provide the desired numbers of standards as parameters to make_standard_table, add or remove the required number of 'else if' statements to

validate the standards, and also pass any additional or fewer parameters to the create_std_table call on line 52.

2. *Line 77-94, formatr.R:* provide the desired numbers of standards as parameters to create_standard_table. Add or remove get_standard_data calls as necessary in the function body, add or remove the desired number of amounts of standard data from the std_table creation in line 87. Lastly, modify the column names as required.
3. *Line 131-150, rtkin.R:* add or remove the required calls for averages for the different standard concentrations. Modify the vector concentration as required, and also the vector_mean to include more or less standard means.

To change the number of standard replicates:

1. *Line 77-94, formatr.R:* Modify the row names as required by adding or reducing replicate numbers.
2. *Line 55, rtkin.R:* modify the colnames to include the correct number of replicates.

To change the timepoint at which the standard absorbance is collected:

1. *Line 120, formatr.R:* change the number “0” to the preferred number.

To change the concentrations of standards:

1. *Line 89, formatr.R:* change the number provided in the column names vectors.
2. *Line 133 + 135 + 137 + 139 + 143, rtkin.R:* change the numbers in the mean call to reflect the appropriate standard concentrations (as set in line 89 or formatr.R), then also apply these new numbers to the values in vector_conc (line 143).

To use a standard that isn’t glucose:

There is no functional requirement to doing so, but the output labels will match the assay.

1. *Line 55 + 59 + 79 + 82 + 106 + 147:* change instances of “glucose_conc” to a character which better reflects the actual identity of the standard.

To change the number of test replicates:

Keep in mind that while there is no requirement on the number of replicates required for each test, there should be of the same number of replicate wells in each test identifier used. This means, if you wish to have less for a specific test include additional 'NA's instead of a well ID in the list of well ID's.

1. *Line 247, formatr.R:* change the value of replicate_num to whichever integer suits the assay you are running.
2. *Line 254, rtkin.R***:** change the column indexes to match what the new test tables will look like with the different number of replicates.
3. *Line 261, rtkin.R:* change the column indexes to match what the new test tables will look like with the different number of replicates.
4. *Line 321, rtkin.R:* change the vector indexes to match what the new test tables will look like with the different number of replicates.
5. *Line 341, rtkin.R:* change the table indexes to match what the new test tables will look like with the different number of replicates.
6. *Line 395, rtkin.R:* change the table indexes to match what the new test tables will look like with the different number of replicates.
7. *Line 315 + 316 + 318 + 319:* change the indexes to match what the new test tables will look like with the different number of replicates.
8. *Line 354, linearrange.R:* replace both '5''s with the column number in which the averaged analyzed data appears in the table and the number of columns in each individual test table.

To export files to different filenames:

1. *Line 57+60, platetidyr.R:* to export the comma-separated raw data to a different file name, change the character name.
2. *Line 63, formatr.R:* to export the raw absorbance data for the standard table to a different file name, change the character value.
3. *Line 163, formatr.R:* to export the raw absorbance data for the test wells to a different file names, change the character value.

4. *Line 71, rtkin.R:* to export the standard curve linear regression parameters to a different filename, change the character value.
5. *Line 177, rtkin.R:* to export the analyzed test data to a different filename, change the character value.
6. *Line 181, rtkin.R:* to export the test linear regression parameters to a different filename, change the character value.
7. *Line 190, rtkin.R:* to export the progress curve plot to a different file, change the character value.
8. *Line 249, linearrange.R:* to export the modified linear test data to a different file, change the character value.
9. *Line 253, linearrange.R*: to export the modified linear test regression parameters to a different file, change the character value.
10. *Line 262, linearrange.R:* to export the modified linear progress curve plot to a different file, change the character value.

Modifying the Plots:

Standard Curve Plot:

- *Lines 80- 87 of rtkin.R*
- To modify the points, change geom_point
- To modify the line of best fit, change geom_smooth
- To modify the titles, changes labs
- Additional information may be found in documentation for ggplot.R

Progress Curve Plot (Pre-Linearization):

- *Lines 200-215 (plot kinetics) of rtkin.R*
- To modify the points change geom_point
- To modify the line of best fit, change geom_smoot

- To add error bars, uncomment line 210
- To change the titles, edit labs
- Additional information may be found in the documentation for ggplot.R

Progress Curve Plot (With Linear Ranges Described):

- *Lines 277 – 297 of linearrange.R*
- To modify the plotted individual well replicates, change geom_point
- To modify the line of best fit, change geom_smooth
- To modify the titles, change labs
- Additional information may be found in the documentation for ggplot.R

Appendix:

Input Type Definitions:

Blank identifier:	Blank identifiers are of the form: list("blank_type", "name", list(well IDs)). Where blank_type must be one of: “substrate” or “enzyme”. The name must also match the same name formatting as used in the test identifiers (i.e. if you abbreviate maltose to malt in the test identifiers, list the maltose blank as malt).
Imported Data:	The imported data refers to the read in plate-shaped form.
Plate-Shaped Data:	Plate shaped data refers to data arranged in a table based on the layout of the plate. Imagine the plate map with well id replaced with the absorbance measurements.
Linear identifier:	Linear ranges are of the form: list(“test_name”, list(lin_min, lin_max, outliers)). Where “test_name” is the name for that progress curve either in the exported data file or plot (i. e. “TestIDName_SubsConc”). ‘Lin_min’ is the minimum timepoint to be included, and ‘lin_max’ is the maximum timepoint to be included. Outliers is a list of timepoints to be excluded and may be left empty.
Raw Data File:	Refers to the untouched text data file saved from the plate reader/spectrophotometer.
Test Identifier:	Test identifiers are of the form: list("Enzyme_subs", subs_conc_mM, list(well IDs)).
Tidy Data:	The imported data organized by well id and time.

Well Id:	Refers the combination of letter-number representing the well position in a 96-well plate. Ex. ‘A1’, ‘B12’.

96-well Plate Map:

This describes how the wells within a 96-well plate are indexed within the code. To orientate yourself this can be directly superimposed to a plate read contained within the raw text file given by the plate reader. To refer to a specific well where a standard or test sample can be found by the proper well ID, refer to the map below.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
A	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12
B	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12
C	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
D	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
E	E1	E2	E3	E4	E5	E6	E7	E8	E9	E10	E11	E12
F	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12
G	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	G11	G12
H	H1	H2	H3	H4	H5	H6	H7	H8	H9	H10	H11	H12