Analysing growth curves and other user-defined plates in opm

Lea A.I. Vaas
Leibniz Institute DSMZ

Markus Göker
Leibniz Institute DSMZ

Abstract

This is tutorial about the analysis of growth curves and other user defined kinetics with the **opm** package in the version of December 22, 2013. It is explained how any kinds of growth or respiration measurements can be input into **opm**. Data without a real structuring into plates and wells can nevertheless be studied with **opm** by using a *virtual* arrangement into plates and wells. We also show how Phenotype Microarray (PM) data with user-defined plate types can be analysed. Analysing such data visually and statistically requires in some cases adaptations of function arguments whose defaults are targeting PM data. All these practically relevant issues are explained in detail.

Keywords: Growth Kinetics.

1. Introduction

A detailed description of the OmniLog® Phenotype Microarray (PM) system, its measuring procedure and data characteristics are found in the vignette "**opm**: An R Package for Analysing OmniLog® Phenotype Microarray Data" (called "main tutorial" in the following). How substrate information stored within **opm** can be accessed and used for advanced visual and statistical analyses is explained in the vignette "Working with substrate information in **opm**" (called "substrate tutorial" in the following). The description of the methods below do not presuppose that the user is already familiar with the usage of **opm**. But for details on its approaches to visualisation and statistical analysis we will refer to the main tutorial, the substrate tutorial as well as the entries of the **opm** manual. Especially the concepts behind the different classes of **opm** objects are only explained in the main tutorial, and the methods available for these classes are only explained in the main and substrate tutorial.

In addition to visual inspection or statistical comparative analyses of PM data, as described in the main tutorial and the substrate tutorial, users might be interested in analysing data other than PM data, or analysing PM with user-defined plate types. To work with user-defined PM plates only requires registering these plates, i.e. storing a mapping from well coordinates to substrate names, and optionally also a full, descriptive name for the plate. The analysis of data other than PM data, such as growth curves, additionally requires inputting these data and converting them to OPMX objects. If these data are not really structured into plates and wells, a *virtual* arrangement into plates and wells must be established, as well as a virtual positioning of the plates in a reader, which is used for identifying each plate. This nomenclature may be unusual for data that have not been measured in plate readers, but

presents no problems in practice. Users should be aware, however, which kinds of comparisons can be made within and between plates of the same plate type. Moreover, some defaults of the plotting functions are only suitable for PM data. Hence, the functions should be called slightly distinctly.

Besides these slight restrictions, which are illustrated with examples below, non-PM data can be analysed with **opm** almost like PM data.

2. Preparation

As usual, **opm** must be loaded before any analysis can begin:

```
R> if ("package:opm" %in% search())
     detach("package:opm", unload = TRUE)
R> library("opm")
```

3. Growth-curve data input

3.1. User-entered data frames

In the following we will use the growth-measurements data set from Vaas, Marheine, Sikorski, Göker, and Schumacher (2013) as exemplar. These data have been entered by hand and then input into R with one of the functions for reading Comma-Separated Values (CSV), yielding a data frame that comes with **opm**:

```
R> data("potato")
R> head(potato)
```

```
  Genotype  Treatment Replicate Time  FM DM
1  07-08-1 0.16M NaCl         1    2 597 44
2  07-08-1 0.16M NaCl         2    2 550 40
3  07-08-1 0.16M NaCl         3    2 633 48
4  07-08-1 0.16M NaCl         4    2 490 31
5  07-08-1 0.16M NaCl         5    2 617 47
6  07-08-1 0.16M NaCl         1    4 585 55
```

For details on this data set, enter `?potato` at the R prompt. The measurements are in "long" format and must be reshaped using the eponymous function into "wide" format. The "long" format was deliberately chosen for demonstrating the use of the `reshape` function. We reshape separately for the Dry Mass (DM) and Fresh Mass (FM) measurements within the data set:

```
R> potato.fm <- reshape(potato, v.names = "FM", drop = "DM", direction = "wide",
     idvar = c("Genotype", "Treatment", "Replicate"), timevar = "Time")
R> potato.dm <- reshape(potato, v.names = "DM", drop = "FM", direction = "wide",
     idvar = c("Genotype", "Treatment", "Replicate"), timevar = "Time")
```

For `reshape`, "long" format means that each measurement is stored in a separate record with one entry per row (see above). Thus here for each data point the "Genotype", "Treatment" and "Time" entries have to be repeated, resulting in a data frame with 540 rows in 6 columns. A call to `reshape` can rearrange the data set into a form where the columns "Genotype", "Treatment" and "Replicate" are kept and the columns "Time" and either "FM" or "DM", respectively, are merged, resulting in 9 columns representing the measurement times:

```
R> head(potato.fm)
```

```
   Genotype  Treatment Replicate FM.2 FM.4 FM.6 FM.8 FM.10 FM.12 FM.14 FM.16 FM.18
1   07-08-1 0.16M NaCl         1  597  585  882  844  1291  1847  2232  2560  2808
2   07-08-1 0.16M NaCl         2  550  614  908 1103  1240  1798  2184  2832  2501
3   07-08-1 0.16M NaCl         3  633  570  855 1200  1392  1827  2360  2522  3113
4   07-08-1 0.16M NaCl         4  490  681 1087  994  1478  1921  2315  2317  2761
5   07-08-1 0.16M NaCl         5  617  707  962  849  1446  1853  2335  2564  2426
46  07-08-1 0.32M NaCl         1  395  551  392  342   322   322   368   336   274
```

Thus the dimensions of the data dwindled to 60 rows in 12 columns. Now the data are in the right arrangement for the next step, the conversion into OPMX or MOPMX objects. (When entering data manually, users who directly choose the "wide" format can, of course, skip the conversion with `reshape`.)

The main function for converting user-defined data frames to OPMX or MOPMX objects is `opmx`, which can directly be applied to the objects created in the last step. This works because the "horizontal" input format of `opmx` corresponds to the "wide" format of `reshape`.

```
R> potato.fm <- opmx(potato.fm, position = c("Genotype", "Replicate"),
     well = "Treatment", prefix = "FM.",
     full.name = c(fm = "Growth experiment, fresh mass"))
R> potato.dm <- opmx(potato.dm, position = c("Genotype", "Replicate"),
     well = "Treatment", prefix = "DM.",
     full.name = c(dm = "Growth experiment, dry mass"))
```

The data frames passed to `opmx` contain all substrate information in their "Treatment" column. Its content will be interpreted as substrate names for wells, which are virtual in our case. Hence, **opm** registers the mapping from well coordinates to substrate names on the fly. The substrate names are taken directly from the data frame in "horizontal" format and registered after sorting. The plate type must be provided, however. As it is not within the data frame, the short name of the plate type is taken from the `full.name` argument, whose main purpose is to enter the full, descriptive name of the plate type. That is, a virtual plate with virtual wells, yielding a user-defined plate type, will be registered. "Genotype" and "Replicate" go to the metadata of the resulting object and together identify each plate. In the case of PM data, this is done using the position of the plate within the OmniLog® reader. Thus the relevant argument here is `position`, which must be supplied unless there is a column of that name. So the "plate position" is also virtual, but just acts as an identifier of the plate. The `prefix` argument helps identifying the columns with measurements over time.

The registered plate type can be queried as follows:

```
R> plate_type(TRUE) # shows all existing user-defined plates
```

```
[1] "CUSTOM:DM" "CUSTOM:FM"
```

```
R> listing(wells(plate = c("CUSTOM:FM", "CUSTOM:DM")))
```

```
CUSTOM:FM:
- Growth experiment, fresh mass
- A01: 0.16M NaCl
  A02: 0.32M NaCl
  A03: 0.5M Sorbitol
  A04: Control
CUSTOM:DM:
- Growth experiment, dry mass
- A01: 0.16M NaCl
  A02: 0.32M NaCl
  A03: 0.5M Sorbitol
  A04: Control
```

Note the prefix "CUSTOM:", which is used to distinguish user-defined plate types from those that come with **opm**. The object resulting from `listing` can be output with `to_yaml` or `saveRDS` for externally storing plate types in files. Indeed, please keep in mind that the definition of plate types is only available in the current R session. The definitions will be lost once the session is terminated.

With the resulting `potato.dm` and `potato.fm` objects the user can now follow the **opm** work flow for processing PM data. Please continue in Section 4 and the following sections for plotting and statistical analysis of the estimated curve parameters.

It is possible to first register the plate, as shown in Section 3.2, and then convert the data *via* `opmx`. This makes most sense if another ordering of wells should be enforced. Otherwise `opmx` takes the substrate names directly from the data frame in "horizontal" format and registers them after sorting.

3.2. Direct registration of plate types

An example input file comes with **opm** containing growth-curve data derived from an experiment with two *Escherichia coli* strains (Deutsche Sammlung von Mikroorganismen (DSM) 18039 = K12 and the type strain DSM 30083^{T}) on increasing Glucose concentrations. Here we are dealing with a real plate with real wells, but the registering procedure would be the same for virtual plates with virtual wells. Thus, it will here be shown how to prepare a plate map and register it as a new plate type. Section 3.2 then shows how to import the data and subsequently convert them to an OPMX or MOPMX object. Each combination of strain and Glucose concentration was repeated twice on the plate. It will thus be shown how to define a numbering of these repetitions suitable for later on using the `split` function to split the object into one object per repetition.

The **opm** package offers several ways to set up a user-defined plate layout. The function `register_plate` is useful for both customised PM plates and measurements from quite different experiments such as growth curves and other kinds of kinetics.

For small data sets it might be feasible to type the substrate allocation manually into a character vector, as done in the following. The short name of the plate type will be "growth",

as simply given by the named function argument. Here two argument of the same name are passed to the function for registering the full name and the well mapping in a single call:

```
R> register_plate(
     growth = c(
       A01 = "Negative Control #1", A02 = "10mM Glucose #1",
       A03 = "20mM Glucose #1", A04 = "50mM Glucose #1",
       A05 = "100mM Glucose #1", A06 = "200mM Glucose #1",
       B01 = "Negative Control #2", B02 = "10mM Glucose #2",
       B03 = "20mM Glucose #2", B04 = "50mM Glucose #2",
       B05 = "100mM Glucose #2", B06 = "200mM Glucose #2",
       C01 = "Negative Control #3", C02 = "10mM Glucose #3",
       C03 = "20mM Glucose #3", C04 = "50mM Glucose #3",
       C05 = "100mM Glucose #3", C06 = "200mM Glucose #3",
       D01 = "Negative Control #4", D02 = "10mM Glucose #4",
       D03 = "20mM Glucose #4", D04 = "50mM Glucose #4",
       D05 = "100mM Glucose #4", D06 = "200mM Glucose #4"
     ),
     growth = "Growth on Glucose"
   )
R> listing(wells(plate = "custom:growth"))
```

However, manually entering the well mapping is error prone and not efficient when dealing with data sets containing more than a few wells. Alternatively, a user-designed plate can also be registered with a plate map given as matrix. The matrix then directly represents the allocation of the used substrates on the plate. Because of the repetitions in the substrate names (note the numbering, which is necessary here to generate unique substrate names, and later on important to split the plate), the texts can also be generated with fewer code:

```
R> # create constant part of the substrate names
R> growth <- c("Negative Control", "10mM Glucose", "20mM Glucose",
     "50mM Glucose", "100mM Glucose", "200mM Glucose")
R> # create repetitions and assign according numbers
R> growth <- paste(rep(growth, each = 4), rep(1:4, 4), sep = " #")
R> # create matrix that mirrors the plate layout
R> growth <- matrix(growth, nrow = 4, ncol = 6,
     dimnames = list(LETTERS[1:4], 1:6))
R> # register this plate type and show the result
R> register_plate(growth = growth, growth = "Growth on Glucose")
R> listing(wells(plate = "custom:growth"))
```

Plates with other layouts can be put together in the same way but using other more or fewer row and/or column names. Plates with other repetition structure, or no repetitions of substrates at all, can be put together in the same way, too, but modifying or omitting the way a substrate numbering is introduced. Instead of a matrix, a data frame could be used as well. We will try this here after showing how to delete a plate type again by providing a `NULL` argument:

```
R> register_plate(growth = NULL)
R> growth <- as.data.frame(growth)
R> register_plate(growth = growth, growth = "Growth on Glucose")
R> listing(wells(plate = "CUSTOM:GROWTH"))
```

3.3. Input of TECAN data

The data for which we have registered a full plate name and a mapping from well coordinates to substrate names in Section 3.2 are contained in an exemplar input file that comes with **opm**. It can be found, and input into R, as follows:

```
R> tecan.file <- opm_files("growth")
R> tecan.file <- grep("tecan", tecan.file, ignore.case = TRUE, value = TRUE)
R> tecan <- read.table(tecan.file)
R> head(tecan)
```

```
  V1    V2    V3    V4    V5    V6    V7
1 <> 1.000 2.000 3.000 4.000 5.000 6.000
2  A 0.087 0.088 0.087 0.088 0.085 0.084
3  B 0.087 0.088 0.087 0.086 0.087 0.085
4  C 0.083 0.082 0.081 0.083 0.079 0.077
5  D 0.083 0.083 0.081 0.082 0.080 0.079
6 <> 1.000 2.000 3.000 4.000 5.000 6.000
```

This file was output by a [Lea, please enter the name] instrument as distributed by the TECAN corporation. After recording the data, the [Lea, please enter the name] software provided by TECAN generated such as file *via* [Lea, please enter the procedure].
The resulting format is not particularly useful within R but can be converted using the "rectangular" mode of `opmx`:

```
R> tecan <- opmx(tecan, "rectangular", plate.type = "growth", position = 1,
     interval = 1)
R> tecan
```

```
Class                       OPM
From file
Hours measured              71
Number of wells             24
Plate type                  CUSTOM:GROWTH
Position                    1
Setup time                  Sun Dec 22 18:09:43 2013
Metadata                    0
Aggregated                  FALSE
Discretized                 FALSE
```

Note that we have to refer to the previously registered plate type, "growth". The `position` argument is important for identifying each plate if several plates of this plate type are dealt with. The optional `interval` argument provides the time interval between two consecutive measurements. Ideally, it is provided in hours. Time series with irregular intervals can be entered with the same argument (by directly providing each time point).
[Lea, please check whether this is the correct interval argument for these data.]

The generated OPM object can now be split according to the repetition structure of the wells,

and metadata can be added that describe each resulting plate:
[Lea, please enter the correct strain designations in the code snippet for assigning metadata.]

```
R> tecan <- split(tecan)
R> metadata(tecan) <- data.frame(Replicate = c(1, 2, 1, 2),
     Strain = rep(c("Strain 1", "Strain 2"), each = 2),
     stringsAsFactors = FALSE)
R> dim(tecan)

[1]  4 72  6
```

4. Visualisation of growth curves

Visualising raw measurements of growth curves with the methods intended for PM data is straightforward, but some adaptations are necessary due to the deviations between the distinct kinds of data. For instance, the expected maximum for PM data can seldom be used for delimiting the y axes, and the data are not measured in OmniLog® units, and a negative control might not be present:

```
R> xy_plot(potato.fm, theor.max = FALSE, rcr = 1, include = "Genotype",
     main = list(in.parens = FALSE), ylab = "Mass [g]", neg.ctrl = FALSE)
```

The result is shown in Figure 1. The TECAN data (which contain a negative control) can be visualised in the same way as the potato data, yielding Figure 2. Note the `rm.num` argument, which causes the removal of the numbering from the end of the full well names (which is not needed any more after applying the `split` function as described in Section 3.3):
[Lea, please enter an appropriate y-axis description for the E. coli plot.]

```
R> xy_plot(tecan, theor.max = FALSE, include = "Strain",
     main = list(in.parens = FALSE), ylab = "Value", rm.num = TRUE)
```

5. Estimating parameters from growth curves

TODO: Lea, please make recommendations regarding the spline method to be used with these data. For issues not specific to growth data refer to the main tutorial.

6. Statistical analysis of growth curves

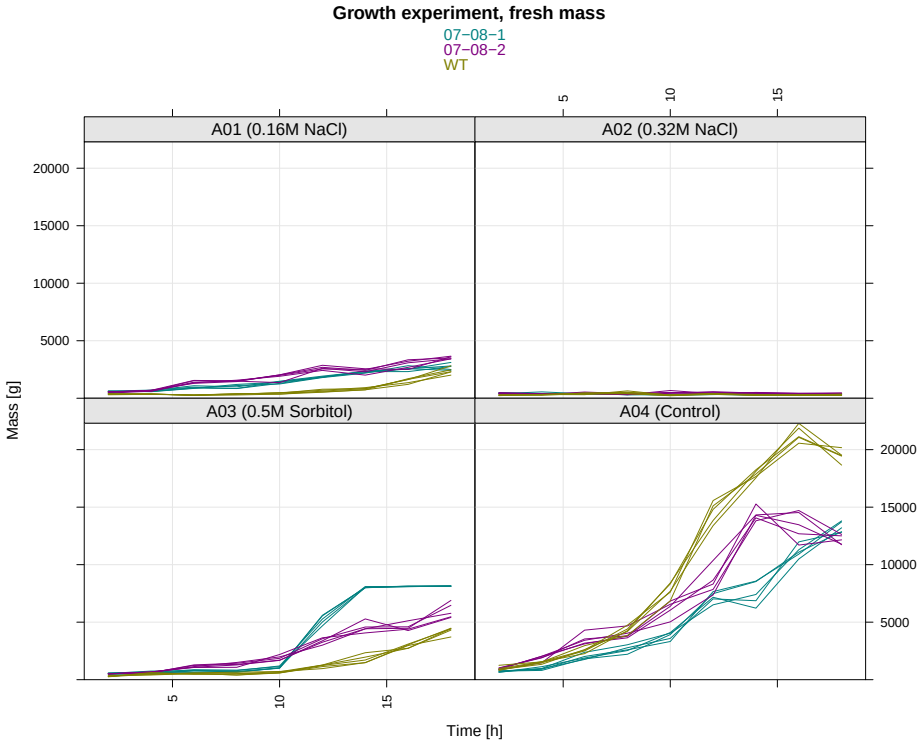

Figure 1: Potato cell line growth measurements, recorded as fresh mass, visualised using the `xy_plot` method. See the main tutorial for details on this kind of plotting. The plot indicates that the wild type grows better than the genetically modified cell lines under non-stress (control) conditions. It also indicates that the stresses impair growth but that the genetically modified cells grow better then the wild type under moderate stress conditions.

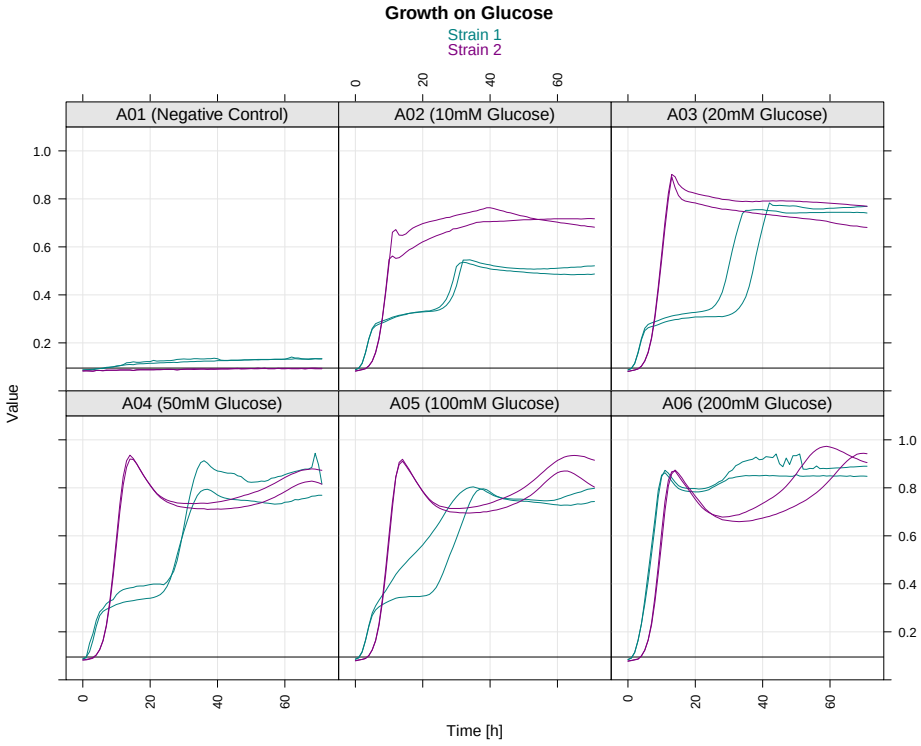

Figure 2: Growth of two *E. coli* strains on Glucose, visualised using the `xy_plot` method. See the main tutorial for details on this kind of plotting. The plot indicates that one of the strains outgrows the other unless high concentrations of Glucose are applied.

TODO: Lea, please conduct selected analyses that make sense for these data. For instance, the interpretations given in the figure captions should be assessed using `opm_mcp`. For issues not specific to growth data refer to the main tutorial.

7. Acknowledgements

We are grateful to Victoria Michael (Deutsche Sammlung von Mikroorganismen und Zellkulturen (DSMZ)) for providing growth curves measured with a TECAN instrument.

References

Vaas LAI, Marheine M, Sikorski J, Göker M, Schumacher HM (2013). "Impacts of pr-10a Overexpression at the Molecular and the Phenotypic Level." *International Journal of Molecular Sciences*, **14**, 15141–15166. doi:10.3390/ijms140715141. URL http://www.mdpi.com/1422-0067/14/7/15141.

Affiliation:

Markus Göker
Leibniz Institute DSMZ – German Collection of Microorganisms and Cell Cultures
Braunschweig

Telephone: +49/531-2616-272
Fax: +49/531-2616-237
E-mail: markus.goeker@dsmz.de
URL: www.dsmz.de