opm: An R Package for Analysing OmniLog® Phenotype MicroArray Data

Lea A.I. Vaas
CBS-KNAW Fungal Biodiversity Centre

Johannes Sikorski
Leibniz Institute DSMZ

Benjamin Hofner
Universität Erlangen-Nürnberg

Nora Buddruhs
Leibniz Institute DSMZ

Anne Fiebig
Leibniz Institute DSMZ

Hans-Peter Klenk
Leibniz Institute DSMZ

Markus Göker
Leibniz Institute DSMZ

Abstract

The OmniLog® Phenotype Microarray system is able to monitor simultaneously, on a longitudinal time scale, the phenotypic reaction of single-celled organisms such as bacteria, fungi, and animal cell cultures to up to 2,000 environmental challenges spotted on sets of 96-well microtiter plates. The phenotypic reactions are recorded as respiration kinetics with a shape comparable to growth curves. Tools for storing the curve kinetics, aggregating the curve parameters, recording associated metadata of organisms and experimental settings as well as methods for analysing these highly complex data sets graphically and statistically are increasingly in demand.

The **opm** R package facilitates management, visualization and statistical analysis of Phenotype Microarray data. Raw measurements can be easily input into R, combined with relevant meta-information and accordingly analysed. The kinetics can be aggregated by estimating curve parameters using several methods. Some of them have been specifically adapted for obtaining robust parameter estimates from Phenotype Microarray data. Containers of **opm** data can easily be queried for and subset by using the integrated metadata and other information. The raw kinetic data can be displayed with customized plotting functions. In addition to 95% confidence plots and enhanced heat-map graphics for visual comparisons of the estimated curve parameters, the package includes customized functionality for user-defined simultaneous multiple comparisons of group means. It is also possible to discretize the curve parameters and to export them for reconstructing character evolution or inferring phylogenies with external programs. Tabular and textual summaries suitable for, e.g., taxonomic journals can also be automatically created and customized. Export and import in the YAML markup language facilitates the data exchange among labs. All functionality is exemplified using real-world data sets that are part of the package.

Keywords: Bootstrap, Cell Lines, **grofit**, Growth Curves, **lattice**, Metadata, Microbiology, Respiration Kinetics, Splines, YAML.

1. Introduction

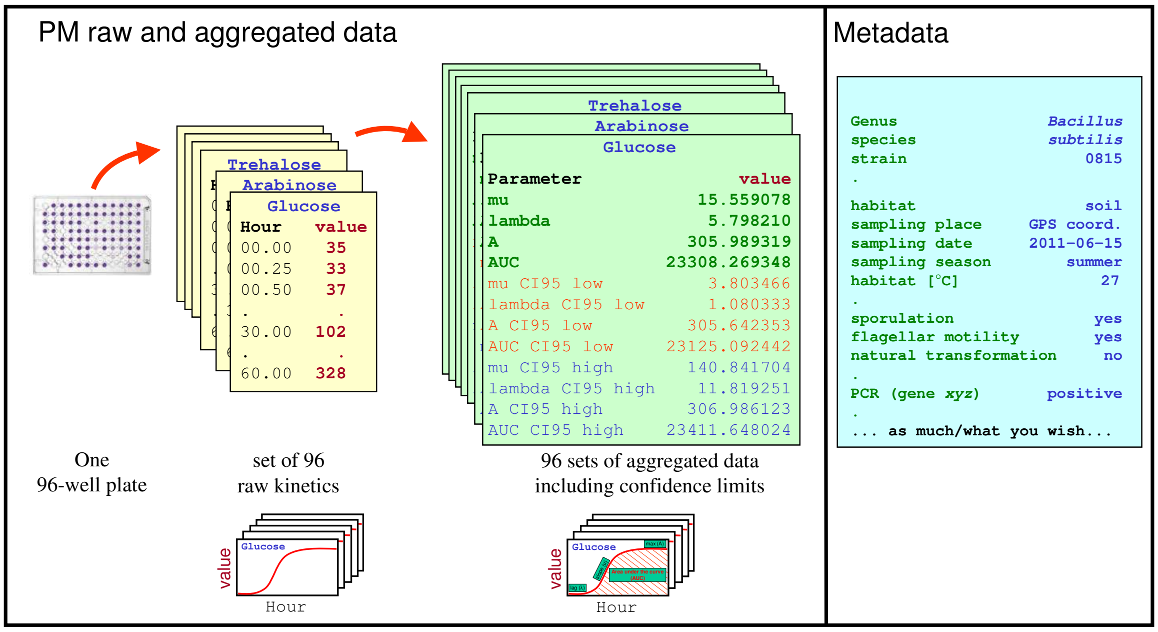
1.1. Preamble for impatient readers

Readers who want to jump right into examples for applying **opm** to their data will find an overview of what the package can do for them in Figure 2. Next, Figure 5 should be looked at, as it lists the names of the functions that can be used in each step of the **opm** work flow. Examples for each step would then be found in the according subsections of of Section 3. Details on the scientific background could well be skipped during a first reading. The user would nevertheless find them in Section 2, including references for important functionality.

1.2. Scientific introduction

The phenotype is regarded as the set of all types of traits of an organism (Mahner and Kary 1997). The phenotype is of high biological relevance, as it is the phenotype which is the object of selection and, hence, is the level at which evolutionary directions are governed by adaptation processes (Mayr 1997). It is also the phenotype which is of direct relevance to humans, for example in exploiting microorganisms for industrial purposes or in the combat of pathogenic organisms (Broadbent, Larsen, Deibel, and Steele 2010; Mithani, Hein, and Preston 2011). In the study of single-cell living beings, such as bacteria, fungi, plant or animal cells, it is an important field of research to study the phenotype by measuring physiological activities as a response to environmental challenges. These can be single carbon sources, which may be utilized as nutrients and hence trigger cellular respiration, or substances such as antibiotics, which may slow down or even inhibit cellular respiration, indicating a successful inhibitory effect on potentially pathogenic organisms. The intensity of cellular respiration correlates with the production of NADH engendering a redox potential and thus a flow of electrons in the electron transport chain. To measure cellular respiration in an experimental assay, this flow of electrons can be utilized to reduce a tetrazolium dye such as tetrazolium violet, thereby producing purple colour (Bochner and Savageau 1977). In principle, the more intense the colour, the larger the physiological activity.

The Phenotype MicroArray (PM) system is capable of measuring a large number of phenotypes in a high-throughput-system utilizing the above described tetrazolium detection system. About 2,000 distinct physiological challenges, such as the metabolism of single carbon sources for energy gain, the metabolism under varying osmolyte concentrations, and the response to varying growth-inhibitory substances are included in the PM microtiter plates (Bochner, Gadzinski, and Panomitros 2001; Bochner 2009). The OmniLog® PM system records the colour formation in an automated setting (every 15 minutes) throughout the duration of the experiment, which may last up to several days. Thus the experimenter ends up with high-dimensional sets of longitudinal data, the PM respiration kinetics. For a detailed introduction into the experimental setup for obtaining OmniLog® PM respiration kinetic data we refer to the OmniLog® website (http://www.biolog.com/) and the associated hardware and software manuals. Briefly, 96-well microtiter plates with substrates, dye, and bacterial cells are loaded into the OmniLog® reader, a hardware device which provides the appropriate incubation conditions and also automatically reads the intensity of colour formation during tetrazolium reduction. The OmniLog® reader is driven by the *Data Collection* software. The stored results files, which are in a proprietary format, are then imported into the *Data Manage-*



Hour	value
00.00	35
00.25	33
00.50	37
.	.
30.00	102
.	.
60.00	328

Parameter	value
mu	15.559078
lambda	5.798210
A	305.989319
AUC	23308.269348
mu CI95 low	3.803466
lambda CI95 low	1.080333
A CI95 low	305.642353
AUC CI95 low	23125.092442
mu CI95 high	140.841704
lambda CI95 high	11.819251
A CI95 high	306.986123
AUC CI95 high	23411.648024

Genus	*Bacillus*
species	*subtilis*
strain	0815
.	
habitat	soil
sampling place	GPS coord.
sampling date	2011-06-15
sampling season	summer
habitat [°C]	27
.	
sporulation	yes
flagellar motility	yes
natural transformation	no
.	
PCR (gene *xyz*)	positive
.	
... as much/what you wish...	

Figure 1: Overview of the data assembly from an PM experiment and the possible additions using **opm**. The raw colour-formation values result in sets of 96 raw kinetics per plate. Using **opm**, they can be augmented by the information coded in the shape characteristics. This yields 96 sets of parameters per plate, each containing four robustly estimated parameters that describe distinct aspects of the respective curve shape. The **opm** package also offers tools for further combining this bundle of raw and aggregated data of each single kinetic with meta-information on the organisms and/or experiments. Based on this meta-information, a variety of visual and statistical comparison tools for either the raw or the aggregated data are available in **opm**.

ment, *File Management/Kinetic Analysis*, and *Parametric Analysis* software packages for data analysis.

In the case of positive reactions, the kinetics are expected to appear as sigmoidal curves in analogy to typical bacterial growth curves (Figure 1). The intrinsic higher level of data complexity contains additional valuable biological information which can be extracted by exploration of the shape characteristics of the recorded curves (Brisbin, Collins, White, and McCallum 1987). These curve features can, in principle, unravel fundamental differences or similarities in the respiration behaviour of distinct organisms, which cannot be identified by the traditional end-point measurements alone. But the meta-information of interest on the organisms and experimental conditions must also be available for an according statistical assessment.

The motivation for the here presented **opm** package originated from (i) the need to overcome the limited graphical and analysis functions of the proprietary OmniLog® PM software and (ii) the desirability of an analysis system for this kind of data in a free statistical software environment such as R (R Development Core Team 2011). At the moment, the visualisation of the kinetics is of limited quality, especially when simultaneously comparing the curves from more than two experiments. The calculation of curve parameters is rather crude (Vaas, Sikorski, Michael, Göker, and Klenk 2012; BiOLOG Inc. 2009). The statistical treatment of raw kinetic data and curve parameters would involve cumbersome manual and hence error-prone manipulations of data in typical spreadsheet applications before they may be imported into appropriate statistical software. Finally, the amount of organismic or experimental metadata that can be added to the raw data is extremely limited.

Based on a previous study (Vaas *et al.* 2012) the here presented **opm** package offers functionalities for a fast and comprehensive evaluation of PM respiration kinetics suitable for a wide range of experimental questions.

Using customized input functions, raw kinetic data can be transferred into R, stored as S4 objects (Chambers 1998) containing single or multiple OmniLog® PM plates and further processed. The package features the statistically robust calculation and attachment of aggregated curve parameters including their (bootstrapped) confidence intervals. Moreover, infrastructure is provided to merge this with any kind of additional metadata. These complex data bundles can then be exported in YAML format (http://www.yaml.org/), which is a human-readable data serialization format that can be read by most common programming languages and facilitates fast and easy data exchange between laboratories.

The framework for data evaluation starts with several functions for the graphical display of the data such as the raw respiration curve kinetics or the confidence intervals of aggregated curve parameters. With sophisticated selection methods the user is able to sort, group and arrange the data according the specific experimental questions in the plotting and analysis framework. Since most addressed experimental questions require to statistically compare not only single curves, but distinct groups of curves, the package provides adapted functionality for performing simultaneous multiple comparisons of group means (Bretz, Hothorn, and Westfall 2010). Because the definition of groups using stored metadata is highly flexible, the user is enabled to compute contrast tests for individually defined sets of mean comparisons (Hsu 1996).

For further specific graphical or statistical analysis according to the needs of the users, the **opm** package organises and maintains the data such that any additional data exploration using other packages in the R environment are easily applicable.

The work flow described below includes (i) the input of raw kinetic data and integration of corresponding metadata, (ii) conversion into suitable storage formats, (iii) the computation of a set of four parameters sufficient for comprehensively describing the curves' shape (aggregated data), (iv) manipulating and querying the constructed objects, (v) visualizing both raw kinetics and aggregated data, (vi) statistical comparison of group means, (vii) discretization of the curve parameters and corresponding export methods and (viii) obtaining additional information the substrates.

2. Methods

2.1. Overview

In the following the work flow (see Figure 2) for generating an R object that contains multiple OmniLog® plates along with the kinetic raw data, the corresponding metadata of interest, and the corresponding aggregated curve parameters, is described. Further it is explained how to analyse either raw data, metadata, aggregated curve parameter data, or combinations of all, as stored in the respective R object, by both graphical or statistical approaches.

The raw data of the reduced tetrazolium colour intensity values can be exported by the proprietary OmniLog® software *File Management/Kinetic Analysis* as CSV (comma-separated values) files and imported into the **opm** package using `read_opm()`. In a first step the valuable biological information coded in the shape characteristics of the recorded curves have to be extracted. Using the function `do_aggr()`, the length of the lag phase λ, the respiration rate μ (corresponding to the steepness of the slope) and the maximum cell respiration A are calculated. As an additional descriptive parameter of cell respiration, the area under the curve (AUC) is estimated, usually *via* numerical integration. All implementations also provide confidence limits calculated *via* bootstrapping, with 95% being the default confidence value (Efron 1979).

To facilitate a comprehensive and straightforward data processing and analysis, the raw and aggregated data of each single kinetic can be concatenated and combined with metadata (see Figure 1) using, e.g., the `include_metadata()` function. It has to be emphasized that metadata can include all kind of describing characteristics of the observed organism(s) such as taxonomic affiliation and geographical and/or ecological origin, or of the performed experimental setting such as culture conditions, genetic modifications, physiological information of any kind and so on.

The work flow of the package was designed for offering a maximum of flexibility with respect to the type of information added to the R object and to the order of steps in which this is achieved. For example, it is possible to add first the metadata and to perform some of the later described analysis and second to aggregate the raw kinetics and go on with analysis of the aggregated values. Since experimental frameworks can be imagined where only very limited meta-information is available, it is also feasible to work without metadata at all.

2.2. Data import

The proprietary OmniLog® PM data analysis software *File Management/Kinetic Analysis*

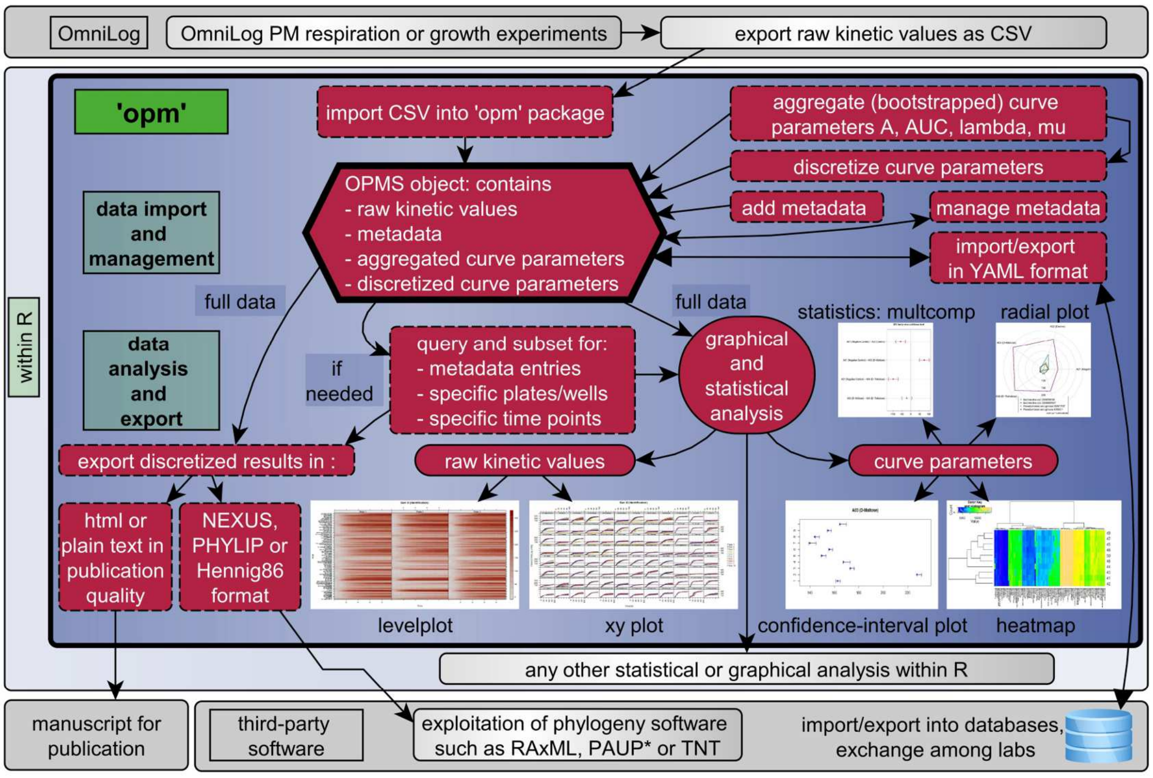

OmniLog
OmniLog PM respiration or growth experiments
export raw kinetic values as CSV
within R
'opm'
import CSV into 'opm' package
aggregate (bootstrapped) curve parameters A, AUC, lambda, mu
discretize curve parameters
add metadata
manage metadata
import/export in YAML format
data import and management
OPMS object: contains
- raw kinetic values
- metadata
- aggregated curve parameters
- discretized curve parameters
full data
full data
data analysis and export
if needed
query and subset for:
- metadata entries
- specific plates/wells
- specific time points
graphical and statistical analysis
statistics: multcomp
radial plot
export discretized results in :
raw kinetic values
curve parameters
html or plain text in publication quality
NEXUS, PHYLIP or Hennig86 format
levelplot
xy plot
confidence-interval plot
heatmap
any other statistical or graphical analysis within R
manuscript for publication
third-party software
exploitation of phylogeny software such as RAxML, PAUP* or TNT
import/export into databases, exchange among labs

Figure 2: A depiction of the possible work flow within **opm** and its potential interplay with R and third-party software. The work flow allows the user full flexibility with respect to the type of information added to the created R objects and to the order of steps in which this is achieved. See Figure 5 for the functions that can be used the respective step.

(BiOLOG Inc. 2009) allows one to export the kinetic raw data from single or multiple plates as CSV files containing a small amount of associated run information that the user can enter at the interface of the OmniLog® PM *Data Collection* software which controls the OmniLog® reader. Currently this conversion involves the creation of files with the extension "d5e" from the original ones with the extension "oka". For use with **opm**, the raw kinetic data should be exported into a single CSV file for each measured plate. The **opm** package currently does not support the input of several plates from PM-mode runs stored in a single CSV file, but it offers functionality for splitting old-style CSV files containing multiple plates. (We refer to the CSV exports from the currently distributed OmniLog® PM *File Management/Kinetic Analysis* software as "old style". Forthcoming versions are expected to export the data in a slightly different CSV format we call "new style". Please contact your local representative of the vendor for the latest software version.) As of version 0.4-0, **opm** also supports the input of MicroStation® CSV files (frequently used in conjunction with EcoPlate® assay for microbial community analysis). These files contain only end-point measurements but potentially several plates, which can nevertheless be input together with their potentially also rich meta-information.

The easiest way to load the raw kinetic data (as CSV files or as YAML) into R in a single step is using the function `read_opm()` (see Figure 2). If raw data from only one single-plate OmniLog® PM are imported, the resulting object belongs to the S4 class OPM. This class for holding single-plate OmniLog® PM data originally includes the information read from the original input CSV files, but an arbitrary amount of metadata can be added later on (see Figure 2). If multiple plates are imported, the resulting object automatically belongs to the S4 class OPMS. In the OPMS class, data may have been obtained from distinct organisms and/or replicates, but must correspond to the same plate type and must contain the same wells (see Figure 2). The function `read_opm()` has an argument "convert" which controls how sets of plates with distinct types are treated; for instance, the function can return a list of OPMS objects, one for each plate type encountered. The entire S4 class hierarchy used by **opm** is shown in Figure 3. Users come in direct contact only with OPM, OPMA, OPMD and OPMS classes. A number of S3 helper classes are also used by several functions.

2.3. Integration of metadata

The interface of the *Data Collection* software of the OmniLog® reader is size-restricted and allows for only a few entrances to enter accompanying information such as the organism under study, the culture conditions, etc. to the plate, and not all of these fields are exported together with the raw measurements. However, for most experimental designs there clearly exists the need to add much more meta-information to the kinetic data. To this end, the **opm** user can integrate the metadata in OPM and OPMS objects using the function `include_metadata()` (among other functions for this task; see Figure 2). Usually, the metadata are kept in a data frame which can be generated from a CSV file. To guarantee an unambiguous match between the raw kinetic data in the OPMS object and the collected metadata, a unique identifier is needed. By default the combination of *Setup Time* and *Position* is used, which should unequivocally identify certain plates. *Setup Time* indicates the date and time at the precision of seconds of starting the batch read in the OmniLog® reader. *Position* indicates the position of the plate in the OmniLog® reader; for instance, *10-A* indicates the plate sliding carriage number 10 in slot A of the reader. Both *Setup Time* and *Position* are automatically recorded by the OmniLog® reader *Data Collection* software and are exported by the OmniLog® PM

File Management/Kinetic Analysis software into CSV files together with the raw kinetic data.

To facilitate the compilation of metadata information, `collect_template()` generates a data frame (and additionally, if requested, a CSV file) in which each line represents a single PM plate. The function `collect_template()` automatically includes the *Setup Time* and *Position* (or any other CSV data of interest) of each plate into the data frame or file. The user can subsequently add further columns describing any metadata of interest of any PM plate of interest. The data frame or CSV file can then be queried for the information specific to each plate, and the resulting data integrated into OPM or OPMS objects using `include_metadata()`. Whereas this function will usually result in non-nested metadata entries, the implementation allows one, in principle, to deal with arbitrarily nested meta-information. The amount of meta-information added and plates analysed is only limited by the available computer memory.

The user can provide additional information to the metadata data frame on the fly (if not provided in CSV) by calling the function `edit()`, which opens the R editor enabling the user to modify and add data. Beside changing the metadata entries by using the R Editor, the function `map_metadata()` offers a secure way to map metadata within OPMS objects. The replacement function `metadata()<-` enables the user to set the entire meta-information, or specific entries, directly.

2.4. Batch conversion of many files

To process and store huge numbers of raw data files, the function `batch_opm_to_yaml()` reads all OmniLog® CSV files (or YAML files previously generated with **opm**) within a given list of files and/or directories and converts them to **opm** YAML format. It is possible to let **opm** automatically include metadata and aggregated values (curve parameters) during this conversion. File selection and unselection using regular expressions or globbing patterns is integrated in the function. The result from each file conversion is reported in detail, and a *demo* mode is available for viewing the attempted file selections and conversions before actually running the (potentially time consuming) conversion process. The package is accompanied by a command-line script `run_opm.R`, enabling the users to run the batch conversion without starting an interactive R session.

2.5. Aggregating data by estimating curve parameters

Descriptive curve parameters from the kinetic raw data can be calculated and included in OPM and OPMS objects using the function `do_aggr()`. Curve parameters can be extracted using a spline-based fitting procedure implemented in **opm**. Three different modelling alternatives for the splines exist: (low-rank) cubic smoothing splines (Reinsch 1967) as implemented in `smooth.spline` from the **base** package, thin plate splines (Wood 2003, a generalization of smoothing splines) and P-splines (Eilers and Marx 1996). The latter two are implemented in the package **mgcv**. It is also possible to access methods from the package **grofit** (Kahm, Hasenbrink, Lichtenberg-Frate, Ludwig, and Kschischo 2010) or to use a native implementation which is faster but only estimates two of the four parameters.

The descriptive curve parameters estimated by **opm** are shown in Figure 4. The parameters λ, μ, and A are derived by default from spline fits, whereas AUC is estimated *via* numerical integration (see Figure 2 in Kahm *et al.* (2010) for details). In addition to the point estimates for the parameters from both model and spline, confidence limits can be calculated (for the

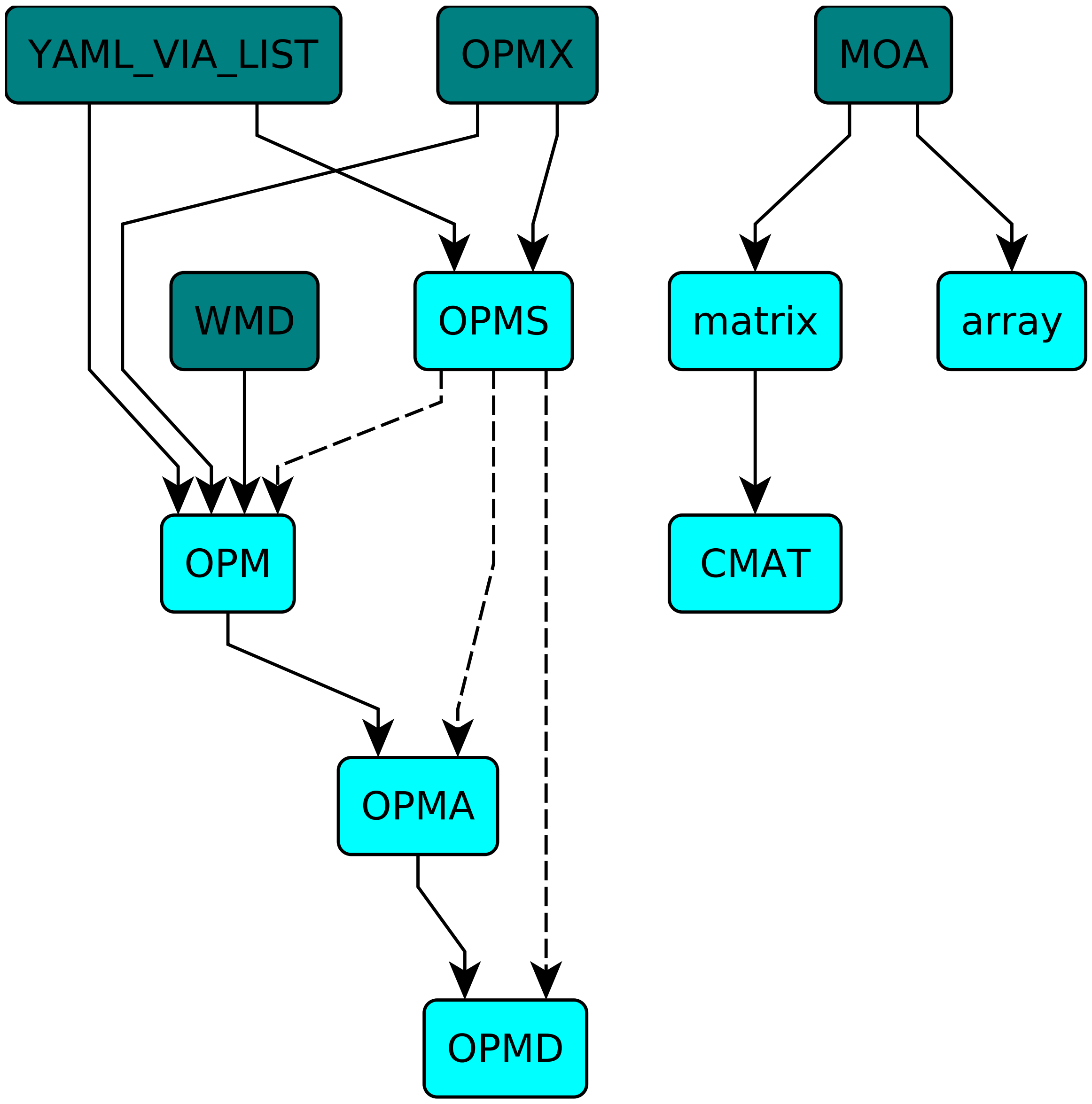


Figure 3: This picture shows the S4 class hierarchy used by **opm**. Class names are shown within the boxes. Boxes with dark background indicate virtual classes, those with light background indicate real classes whose objects can be created and manipulated by some code. Continuous arrows indicate inheritance relationships (pointing from parent to child class), dotted arrows indicate object composition (pointing from the container class to its element classes). Note particularly that OPM, which only contains raw data, csv data and metadata, is the parent class of OPMA, which also contains aggregated data (and has methods for dealing with them). OPMD inherits from OPMA and stores discretized curve parameters in addition to aggregated values. OPMS is a container class that holds OPM, OPMA and/or OPMD objects. These can usually co-occur in a single OPMS object but for some calculations the additional information in OPMA or OPMD objects is strictly required. The query functions `has_aggr()` and `has_disc()` are available for checking from which kinds of objects an OPMS is composed. See the **opm** manual and Section 3 for further details. The non-virtual classes in the right part of the figure are either well-known in R (matrices and arrays) or not directly manipulated by the user (CMAT).

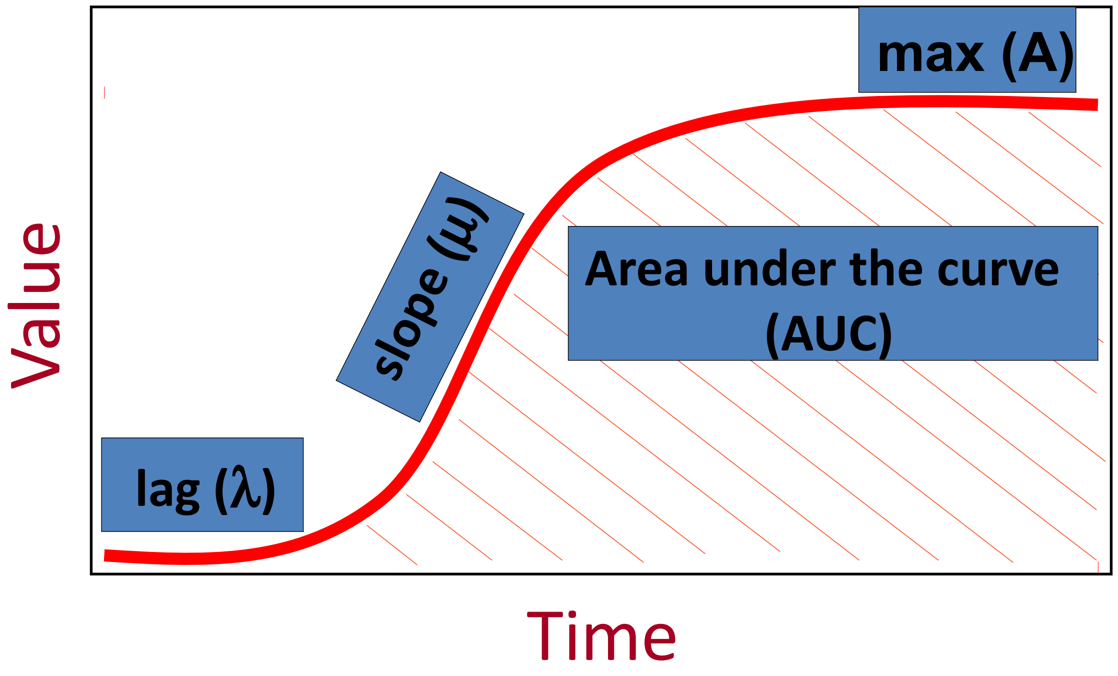


Figure 4: A schematic depiction of a typical growth or respiration curve and the parameters estimated by **opm**. The descriptive curve parameters are the lag phase λ, the respiration rate μ (corresponding to the steepness of the slope), the maximum cell respiration A (corresponding to the maximum value of the curve) and the area under the curve AUC. Note that many respiration curves, even if representing a clearly positive reaction, do not correspond to this idealized scheme. The parameters can nevertheless be robustly estimated from deviating curves, particularly *via* spline fits (Vaas *et al.* 2012).

spline-based approach *via* bootstrapping), with 95% being the default value (Efron 1979). But confidence intervals and according group means can also be calculated from experimental repetitions, as explained in Section 2.7.2. Attaching the aggregated data to an OPM object yields an object of the class OPMA, which can also be stored within an OPMS container object.

2.6. Manipulation of OPM and OPMS data

After integration of additional metadata *via* `include_metadata()` and adding aggregated curve parameters *via* `do_aggr()`, an OPMA or OPMS object comprises basically three pieces of information: (i) the kinetic raw data; (ii) the aggregated data, i.e., the curve parameters λ, μ, A, and AUC and optionally their corresponding 95% confidence limits; and (iii) the metadata.

As usual, the data analysis starts with data exploration for which the user may now wish to subset and query using these pieces of information. As Figure 5 illustrates, the package provides methods for (i) querying and sub-setting OPMS objects, (ii) plotting the data in some customized manner, and (iii) converting the OPM or OPMS to other objects for an independent exploration by the user (discretization and exporting in some useful file formats

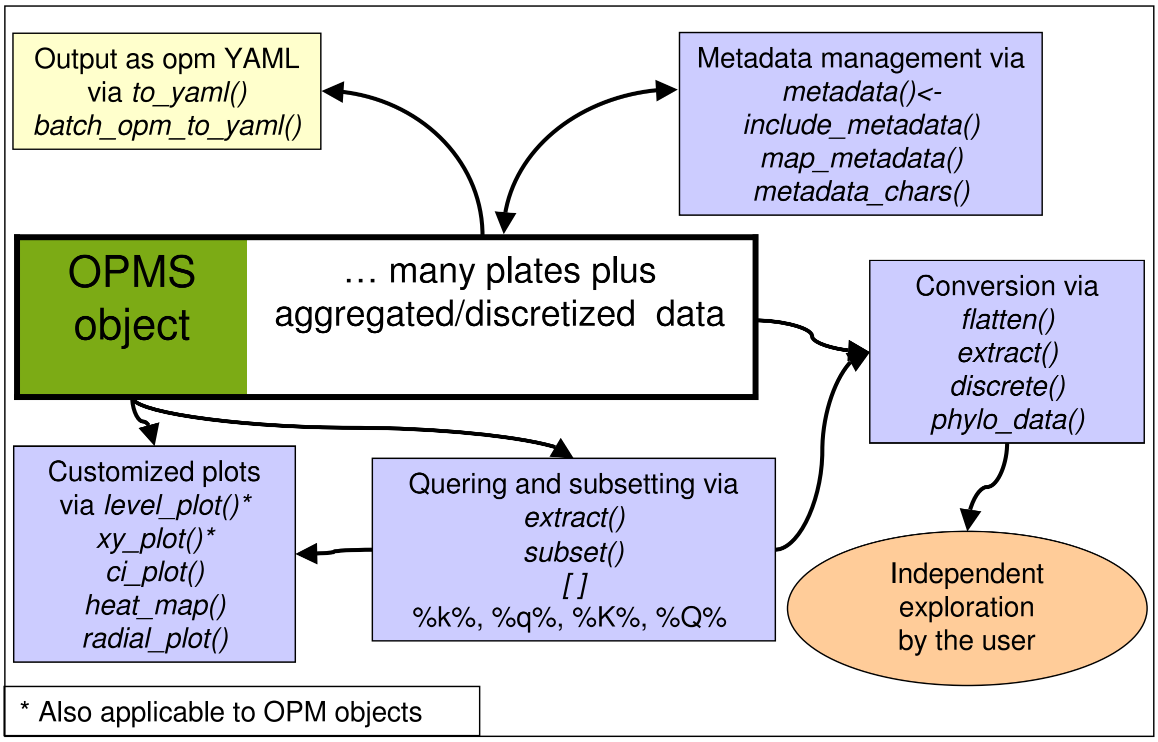


Figure 5: This scheme provides an overview of the possible strategies and appropriate functions for data analysis using the **opm** package. Update needed, should be merged with old Figure 1. Taking an OPMS object containing the kinetic raw data and the aggregated curve parameters (optionally with their corresponding 95% confidence limits) as starting point, methods for metadata management, plotting the data in a customized manner, querying and sub-setting OPMS objects, and data conversion tools including discretization and output in files are provided.

is also possible).

Furthermore, the bundled structure of an OPMS object, and the methods of the class, permit queries for the presence of a specific metadata key or a specific value of a specific metadata key, or a specific combination of values and/or keys, and also enable the user to subset an OPMS object accordingly.

2.7. Plotting functions

Plotting functions for raw data

Lea, a disadvantage of using subsubsections is that they are not numbered and cannot be referred to. If this cannot be fixed (without generating other problems), please convert back to the previously used sections in all parts of this document. Please also use label annotations and check for dead links.

The function `xy_plot()` displays the raw measurements (y-axis) in dependency of the time (x-axis). For each well one sub-panel is drawn, and the user is free to colour the plotted curves by either their affiliation to a specific plate or by a combination of (metadata-)variables of choice. By default the panels are arranged according to the factual microtiter plate dimensions (eight rows labelled A to H $\times$ 12 columns labelled 01-12), but other user-defined arrangements are easily feasible because the plates can be subset by selecting specific wells. Every panel is annotated with the microtiter plate numbering (A01 to H12) and additionally or alternatively with the substrate name (given the plate type, the **opm** package can translate all well coordinates to substrate names). Thus, the function enables the user to compare the curve data in a customized and useful arrangement (Vaas *et al.* 2012).

The function `level_plot()` provides false-colour level plots from the raw respiration measurements over time. Each respiration curve can be displayed as a thin horizontal line, in which the measured respiration value (OmniLog® units) is represented by colour, while the x-axes indicates the measurement times. With increasing respiration measurement values, the displayed colour changes (by default) from light yellow into dark orange and brownish. By default one sub-panel in the level-plot corresponds to one complete plate comprising 96 lines, but as in the case of `xy_plot()` plotting could also be preceded by creating subsets of the plates. The user can obtain an overview in a compacted design (Vaas *et al.* 2012). This plot offers a display format which is especially powerful in uncovering general differences between plates, for example longer lag-phases or smaller AUC values across the majority of wells.

Plotting the aggregated data

The phrasing of this subsection appears to be fine for Johannes now. At moment, Johannes has now own experience on normalizations using `extract()`. If phrasing is also ok, for Lea, she should remove this note.

For the graphical representation of the aggregated data, namely point estimators and corresponding confidence limits for the curve parameters, the function `ci_plot()` provides a framework to plot subsets of different parameters in a convenient and easily applicable manner. This straightforward assembly of different curves' characteristics in a single overview facilitates the interpretation and comparison of user-defined data subsets arranged according to the technical and/or biological repetition structure or other aspects of the experimental design (Vaas *et al.* 2012).

When analysing empirically obtained measurements such as PM data it is important to consider possible systematic variations and control for those. For a PM experiment the purpose of such a normalization is to minimize systematic variations in the aggregated curve parameters so as to more easily recognize biological differences, as well as to allow for the comparison of parameters across plates processed in different experimental runs. The underlying ideas are mainly derived from DNA-microarray experiments for measuring gene-expression levels (Quackenbush 2002).

With `extract()` the user is enabled not only to select certain aggregated or discretized values into common matrices or data frames, but also to apply normalizations before computing point estimates and their respective confidence intervals for individually defined experimental groups. Normalization indicates here the subtraction of the mean of each plate from each of its values (over all wells of that plate), either per row or per column. Alternatively, all values can be divided by the mean of the value. The details can be set in the respective arguments of `extract()`. Although this function is intended mainly as a helper function for `ci_plot()`, it can be quite useful for specific normalization purposes, for example when data were derived before and after servicing the Omnilog® facility, which might result in shifting the measurements by a certain amount. In conjunction with `extract()`, `ci_plot()` allows for visualizing point estimates and confidence intervals of groups of parameter estimates. For visualizing differences between groups and their confidence intervals, see `opm_mcp()` as described in Section 2.8.

Additionally, the package offers the possibility of plotting the aggregated curve parameters as a heat map *via* the function `heat_map()`. Heat maps appear particularly powerful for visualizing the outcomes of PM experiment because dendrograms inferred from both the substrates and the plates can be used to rearrange the plot. Since the user is free to define which metadata or strain information are of interest for the annotation of the plot and the clustering analysis, this tool provides a powerful feature for data exploration in specialized contexts. For instance, the naming scheme of the individual plates can be devised by selecting associated metadata; it is also possible to automatically construct row groups by selecting the same or other meta-information. `heat_map()` is mainly a wrapper for the `heatmap()` functions from either the **stats** or the **gplots** R package, but contains some useful adaptations to the PM data, facilitates the selection of a clustering algorithm and the construction of row and column groups, and provides more appropriate default solutions for row and column descriptions sizes (we suppose that in most situations the pictures produced by `heat_map()` should not need manual adaptation in these respects).

2.8. Statistical comparisons of group means

The phrasing of this subsection is fine for Johannes now. If phrasing is also ok, for Lea, she should remove this note.

Besides comparing single curves, the user may also be interested in statistically comparing the mean values of distinct groups of curves. For example, imagine the comparison of four different bacteria using GEN-III microplates. Assume that for each bacterial strain, ten replicates have been performed. (An according example dataset is actually available in the **opmdata** package.) Do these four bacteria differ in, e.g., the mean value of, e.g., curve parameter A of, e.g, well A1? Here, a statistical comparison of four groups (four organisms) of each 10 values (curve parameter A of 10 replicates of well A1) would need to be performed. Statistically, this results in simultaneous inferences across multiple questions (Hothorn, Bretz, and Westfall 2008). To address this issue the function `opm_mcp()` provides functionality for performing simultaneous multiple comparisons of group means by internally calling `glht()` from the **multcomp** package (Hothorn *et al.* 2008) but providing an easier interface for it, specifically adapted to the typical objects of **opm**. By referring to available metadata and/or the substrate names, the user is able to define groups of interest (e.g., four different bacteria), set up a model of choice and perform multiple comparison of group means on individually specified contrasts (Bretz *et al.* 2010; Hsu 1996). The choice of appropriate models and contrasts will be explained in detail below. As comparisons of the different curve parameters are performed separately, it is possible to ask very specific questions on differences between curve shapes.

At this point, it is necessary to highlight the power and flexibility of simultaneous multiple comparison procedures and encourage the user to apply contrast tests on individually designed sets of mean comparison rather than to employ the probably more popular classical ANOVA-based approaches followed by F-tests. It has to be pointed out that in general F-tests *only* provide global information about main effects and interaction effects. That is, only the significance of a result yields evidence for a difference in the means among any of the considered treatments. For example, in the framework of PM data, a significant F-test on the effect of the substrate would indicate that at least two of the substrates cause distinct respiration. Considering that each PM experiment encounters at least 96 different substrates (overall up to 2,000), this information is obviously nearly useless. Moreover, F-tests neither provide information about effect sizes nor do they ease to address comparisons of particular interest (Schaarschmidt and Vaas 2009).

From our point of view, the very most underlying experimental questions in PM experiments can be best expressed as a set of particular mean comparisons, resulting in a multiple-comparison problem (Hochberg and Tamhane 1987). But if an increasing number of hypotheses is tested, with the number of true hypotheses unknown, the probability of at least one wrong testing decision also increases. That is, if an increasing number of groups is compared to each other, conclusions on significant differences between a pair of groups are increasingly likely to be wrong. Thus the so-called family-wise error-rate, which is essentially the probability of at least one false rejection among all the null hypotheses, needs to be controlled (Tukey 1994). The here internally employed functionality from the package **multcomp** provides solutions for all listed difficulties, since it allows for testing a user-defined set of contrasts based on a broad range of model types while internally controlling the family-wise error-rate.

Lea & Johannes, explain the implementation of how factors are merged to produce a pseudo-one-way-layout (Schaarschmidt and Vaas 2009), and/or remove this note. Needs to be done by Lea (I, Johannes, do not feel competent enough for that)

The function `opm_mcp()` internally reshapes the data into a "flat" data frame containing one column for the chosen parameter value, one column for the well (substrate) name and optionally additional columns for the selected metadata. For performing the testing procedure, a model has to be stated that specifies the factor levels that determine the grouping (Searle 1971; Hothorn *et al.* 2008). The `opm_mcp()` function allows for applying such testing directly to OPMS objects.

2.9. Discretizing the aggregated data and export for phylogenetic analysis

Whereas the main data analysis strategies of the **opm** package are based on quantitative, continuous data (as described in the previous chapters), users may nevertheless be interested in discretizing the estimated curve parameters. For instance, discretizing the data is necessary for analysing the data with external programs that cannot deal with continuous characters. Indeed, phylogeny software such as PAUP* (Swofford 2003) and RAxML (Stamatakis, Ludwig, and Meier 2005) is limited to at most 32 distinct character states (to the best of our knowledge, a maximum-parsimony algorithm applicable directly to continuous data has only been implemented in TNT (Goloboff, Farris, and Nixon 2008)). Phylogenetic studies of PM data are of interest because such phenotypic information is frequently used for taxonomic purposes in microorganisms, and here phylogenetic inference methods might be superior to clustering algorithms (Felsenstein 2004). But tabular or textual descriptions of physiological reactions classified into negative, weak (ambiguous) and positive reactions (see next paragraph for details) are of even greater relevance in current microbial taxonomy (Tindall, Kämpfer, Euzéby, and Oren 2006).

The **opm** package includes data-transformation functionality within the `discrete()` methods for coding continuous characters by assigning them to a given number of equal-width categories within a given range. For example, for the parameter A (the maximum curve height) the theoretically possible range between 0 and 400 OmniLog® units could be used. The data should then be analysed under ordered (Wagner) maximum parsimony in PAUP* (Farris 1970) or with the options for ordered multi-state phenotypic characters in RAxML (Berger and Stamatakis 2010), or corresponding settings in other programs, to minimize the loss of information caused by discretizing the values. For this reason, this kind of unsupervised, equal-width-intervals discretization (Dougherty, Kohavi, and Sahami 1995; Ventura and Martinez 1995), even though simple, appears appropriate for this task. In this context, it also makes not much sense to let a discretization method determine the number of categories because they are not dictated by some property of the data but by the limitations of the subsequently to apply analysis software. The **opm** package offers appropriate functions for data export.

2.10. Determining positive and negative reactions and displaying them as text or table

If users were interested to discretize the parameters into "positive" and "negative" results, this would apparently make most sense for the parameter A because here it is not of interest when and how fast a reaction starts (which would be coded in λ and μ, respectively) or how much overall respiration was achieved (as coded in AUC) but whether or not a reaction takes place at all. Unfortunately, PM data frequently result in a continuum of A values between clearly negative and clearly positive reactions. For instance, the distribution of A in the example datasets distributed with the **opm** and **opmdata** packages is clearly bimodal, but contains a large number of intermediary values. For this reason, the `discrete()` methods and their more user-friendly wrapper `do_disc()` offer a gap-mode discretization by interpreting a given range of values (within the overall range of observations) as "ambiguous". Values below would then be coded as negative, values above the range as positive, and values within the range as either missing information or an intermediary state, "weak". This range could be determined by some discretization approach known from the literature (Dougherty *et al.* 1995; Ventura and Martinez 1995).

The **opm** package offers its automated determination using k-means partitioning as implemented in **Ckmeans.1d.dp** (Wang and Song 2011), using an exact algorithm for one-dimensional data. Alternatively, an algorithm implemented in `best_cutoff()` is available, but it requires measurement replicates (which are highly recommended, if not mandatory, anyway) which need to be specified in the metadata. Both methods are accessible *via* `do_disc()`. Export as richly annotated HTML table is possible using `phylo_data()`. If analysis with phylogenetic programs was of interest, in the case of an intermediary state the data should then be analysed as described above. If intermediary values were coded as missing information they could be analysed under either Wagner or unordered (Fitch) maximum parsimony in PAUP* (Farris 1970; Fitch 1971) or with the options for binary phenotypic characters in RAxML (Berger and Stamatakis 2010), or corresponding settings in other programs.

3. Program application

3.1. Overview

The example dataset distributed with the package (Vaas *et al.* 2012) comprises the results from running 114 GEN-III plates (AES Chemunex BLG 1030) in the PM mode of the OmniLog® reader. The organisms used were two strains of *Escherichia coli* (DSM 18039 = K1 and the type strain DSM 30083^{T}) and two strains of *Pseudomonas aeruginosa* (DSM 1707 and 429SC (Selezska, Kazmierczak, Müsken, Garbe, Schobert, Häussler, Wiehlmann, Rohde, and Sikorski 2012)). The strains with a DSM number could be ordered from the Leibniz Institute DSMZ – German Collection of Microorganisms and Cell Cultures.

Johannes, has the fourth strain also been made available in the meantime?

Each strain was measured in two biological replicates, each comprising ten technical replicates, yielding a total of 80 plates. To additionally investigate the impact of the growth age of cultures on the technical and biological reproducibility of the PM respiration kinetics, strain *E. coli* DSM 18039 was grown on solid LB medium for nine different durations, from 16.75 h (t1) to 40.33 h (t9), respectively. For each growth duration four technical replicates were performed except for t9 (which was repeated only twice), yielding 34 plates for this time-series experiment. All further biological and experimental details of this dataset have been described previously (Vaas *et al.* 2012). The dataset `vaas_et_al` comes with the supporting package **opmdata** and can (if that package is installed, of course) be loaded using:

```
R> library("opm")
R> data("vaas_et_al", package = "opmdata")
```

The metadata included in these objects comprise seven entries. The entry *Experiment* denotes the biological replicate or the affiliation to the time-series experiments. The keys *Species* and *Strain* refer to the organism used for the respective experiment (see above), and *Slot* (either *A* or *B*) indicates whether the plate was placed in the left or the right half of the OmniLog® reader. (Note that for an assessment of the reproducibility of the curves the slot is occasionally of relevance.) Two additional entries contain the index of the time point and the corresponding sample point in minutes for the time series experiment. The key *Plate number* indicates the technical replicate (per biological replicate). The combination of the keys *Strains*, *Species*, *Experiment* and *Plate number* results in a unique label which unequivocally annotates every single plate.

Use `?opmdata::vaas_et_al` to obtain further details. The subsets `vaas_1` and `vaas_4` are described in the **opm** manual. Use `?vaas_1` and `?vaas_4` to view their help pages.

3.2. Data import

The following code describes the import of the OmniLog® CSV file(s) into the **opm** package. The CSV files with the OmniLog® raw data should be stored in one to several user-defined folders. Setting the working directory of R to the parent folder of these using `setwd()` frequently facilitates file selection, but in principle the user can provide any number of paths to input files and/or directories containing such files to the function `read_opm()`, which can load several CSV files (and also YAML files generated by **opm**) at once. A restriction of the input functions is that they can only read CSV files that only contain the measurements from a single plate per file (either a PM plate or a single Gen-III plate measured in either PM- or identification modus). But the package contains a function `split_files()` which can be used to split CSV files with multiple plates into one file per plate For details see the **opm** manual; all functions relevant here are contained in a family of functions called "IO-functions" with according cross-references.

To illustrate the file import step by step, a set of example input CSV files is provided with the package. Before starting, please load the **opm** package by typing:

```
R> library("opm")
```

Then use the built-in function `opm_files()` to find the example files in your R installation and check whether this returns a vector of nine file names:

```
R> (files <- opm_files("testdata"))
R> stopifnot(length(files) == 9) # => error if there were not nine files
```

(It might fail in very unusual R installation situations; in that case, the files must be found manually.) One of these files contains multiple plates and acts as an example for `split_files()`; the other ones can be read directly.

From the given vector of file and/or directory names, files can be easily selected and deselected using globbing or regular-expression patterns. For instance, for reading the three example files in "new style" CSV format (see Section 2.2), use the following code. After performing this step, the OPMS object should contain three plates, as indicated by the customized `summary()` function:

```
R> summary(example.opm <- read_opm(files, include = "*Example_?.csv.xz"))
R> stopifnot(length(example.opm) == 3) # => error if there were not three plates
```

As previously addressed, instead of a single file name the user could also provide several file names to `read_opm()`, or a mixture of file and directory names; if these are contained as subdirectories of the current working directory, `read_opm(".")` or `read_opm(getwd())` would be sufficient to input these files. To filter the files with patterns, the arguments `exclude` and `include` are available. There is also a *demo* mode allowing the user to check the effect of applying these arguments before actually reading files. One can use the `gen.iii` argument to trigger the automated conversion of the plate type to, e.g., "Gen III" or "ECO" plates run in "PM" mode, or convert later on using the `gen_iii()` function itself. Plate-type conversions to one of the "PM" modes are disallowed (and are, to the best of our knowledge, not relevant in practice anyway). The plate type is crucial, as it is disallowed to integrate distinct plate types into a single OPMS objects. The reason is that comparing the same well positions from distinct plate types would be almost always equivalent to comparing apples and oranges.

If more than one plate of the same plate type is read, however, data from all files are automatically integrated in a single OPMS object. To read plates from several types at once, have a look at the documentation of the `convert` argument in the manual. If one uses `read_opm(..., convert = "grp")`, a named list is created with, as each list element, one OPM or OPMS object per plate type, depending on whether only a single plate of that plate type, or several such plates, have been found. The objects for each plate type encountered could then easily be accessed *via* the names of the list.

A single plate could also be imported using, e.g.,:

```
R> example.single <- read_single_opm(files[1])
```

But this might only occasionally be useful, as `read_opm` can cope with single files, too.

3.3. Integration and manipulation of metadata

Several ways for linking metadata to OPM or OPMS objects are possible. The easiest one is probably the batch-inclusion after creating a template with plate identifiers associating it with metadata. In the first step, either a data frame to be manipulated within R or a CSV file to be modified with a suitable editor are created. The **opm** package supports metadata integration by creating a template for such a table from an OPM or OPMS objects that contains plate identifiers in the first columns; by default the keys *Setup Time*, *Position* and *File*. These data must not be changed, ensuring that the package can later on link the metadata to the dedicated plates according to these identifiers.

In the **opm** manual, most functions relevant for metadata manipulation are contained in a family called "metadata-functions" with according cross-references. For the collection of a metadata template in a data frame to be manipulated in R, use this command:

```
R> metadata.example <- collect_template(files, include = "*Example_?.csv.xz")
```

For the generation of a metadata template file, the following command can be used:

```
R> collect_template(files, include = "*Example_?.csv.xz",
     outfile = "example_metadata.csv")
```

This will result in a file "example_metadata.csv" in the current working directory (whose name is accessible using `getwd()`). If other metadata have previously been collected, by default a pre-existing file with the same name will be reused. The pre-defined columns will be respected, novel rows be added, old metadata will be kept and identifiers for novel files will be included and their so far empty metadata columns are set to missing data (NA). You can also provide the location of another previously created metadata file with the `collect_template()` argument `previous`.

The generated CSV file could then be edited using external software; for the purpose of this tutorial, we load it directly and manipulate it in R. To avoid the usual changes in data format and header of the table during the import a customized import function was implemented as a wrapper for `read.delim()`:

```
R> metadata.example <- to_metadata("example_metadata.csv")
```

Per default, this expects CSV columns separated by tabulators, with the fields protected by quotes. To input other formats, consider the `sep` argument for defining an alternative column separator, as well as the `strip.white` argument for turning the removal of whitespace at the beginning and end of the fields on or off (which is relevant if a spreadsheet program exports CSV *without* quotes). Now the user could add information to the data frame by calling `edit()`, which would open the R editor, or by any other way of manipulating data frames in R. New columns could be defined, or the existing metadata modified. But the first columns must remain unchanged because they are needed to identify individual PM plates for linking them to their meta-information. As an example, we here add an (arbitrary) *Colour* column with the values “blue”, “red” and “yellow”:

```
R> metadata.example[, "Colour"] <- c("blue", "red", "yellow")
```

Now the metadata are ready to be included into the previously generated OPMS object:

```
R> example.opm <- include_metadata(example.opm, md = metadata.example)
```

The metadata could then be received as follows:

```
R> metadata(example.opm)
```

This returns the entire metadata entries as a list. By default only the added metadata are included in the object, but not the identifiers used for assigning data frame rows to plates.

One might want to tidy the files up if they are not needed any more:

```
R> unlink("example_metadata.csv")
```

A couple of other functions have been implemented for manipulating metadata included in OPM and OPMS objects. For instance, the entire meta-information, or specific entries, can be set using the replacement function `metadata()<-` (see the **opm** manual for details). In the following we discuss metadata modification using `map_metadata()`.

Making use of the exemplar generated above, the key *Colour* could be changed to *Colony colour* as follows:

```
R> (md.map <- metadata_chars(example.opm, values = FALSE))
```

This yields a character vector including itself as `names` attribute, thus implying an identity mapping. Next the new labels will be defined and will then be exchanged with the old ones using `map_metadata()`.

```
R> md.map["Colour"] <- "Colony colour"
R> example.opm <- map_metadata(example.opm, md.map, values = FALSE)
R> metadata(example.opm)
```

The keys should have been changed to *Colony colour* now but the values should have remained unaffected. In addition to mapping based on character vectors, a mapping function could also have been used. By setting their argument `values` to `TRUE`, the functions `metadata_chars()` and `map_metadata()` could be used as well to modify values instead of key. For instance, assume any entries "red" in the field denoted *Colony colour* should be changed to "green":

```
R> (md.map <- metadata_chars(example.opm, values = TRUE))
R> md.map["red"] <- "green"
R> example.opm <- map_metadata(example.opm, md.map, values = TRUE)
R> metadata(example.opm)
```

This command will transform all entries in the table with the value "red" to "green". Other values, as well as the keys, should be unaffected. It is possible to map other types of entries such as numeric vectors by requesting their coercion to the character type; see the **opm** manual for details.

3.4. Batch conversion of many files

In addition to `read_opm()` and `read_single_opm()`, which need to be called before an interactive exploration of PM data, batch-processing large numbers of files by converting them from CSV (or previously generated YAML) to YAML format, optionally after aggregating the raw data by estimating curve parameters and integrating metadata, is also possible. Again there is a *demo* mode to first investigate the attempted conversions:

```
R> batch_opm_to_yaml(files, include = "*Example_?.csv.xz",
     aggr.args = list(boot = 100, method = "opm-fast"),
     outdir = ".", demo = TRUE)
```

The arguments `aggr.args` and `md.args` control aggregation and metadata incorporation, respectively; details on both processes are given below, and for the exact use of these arguments see the **opm** manual. The following command would thus read three of the seven example input files, estimate two of the four curve parameters using the fast native method including 100 rounds of bootstrapping, and store the resulting YAML files (one per plate) in the current working directory (given by "."):

```
R> batch.result <- batch_opm_to_yaml(files, include = "*Example_?.csv.xz",
     outdir = ".",
     aggr.args = list(boot = 100, method = "opm-fast"))
```

By default, progress messages are printed to the screen. The return value, here assigned to the `batch.result` variable, also contains all information about the success of the individual file conversions. The `run_opm.R` script distributed with the package is an Rscript-dependent command-line tool for non-interactively running such file conversions. Its location in the file system can be obtained using

```
R> opm_files("scripts")
```

3.5. Aggregating data by estimating curve parameters

The package brings along an OPMS object, named `vaas_et_al`, containing multiple full 96-well plates, aggregated data (curve parameters), and metadata. For demonstration purposes a subset of one plate, provided in the object `vaas_1`, will be used:

```
R> data("vaas_1")
```

Data aggregation (curve-parameter estimation) can be performed using `do_aggr()`. In the **opm** manual, this one and the other functions relevant for data aggregation are contained in a family called "aggregation-functions" with according cross-references. `vaas_1` already contains aggregated data but we will re-calculate some for demonstration purposes. For invoking the fast estimation method, use:

```
R> vaas_1.reaggr <- do_aggr(vaas_1, boot = 100, method = "opm-fast")
```

This will only estimate two of the four parameters, namely A and AUC. (Screen messages output by `boot.ci()` might be annoying but can usually be ignored.) Information about the data aggregation settings is available *via*:

```
R> aggr_settings(vaas_1)
R> aggr_settings(vaas_1.reaggr)
```

and the aggregated data can be extracted as a matrix *via*:

```
R> vaas_1.aggr <- aggregated(vaas_1)
R> vaas_1.reaggr.aggr <- aggregated(vaas_1.reaggr)
```

The default function of `do_aggr()` includes 100-fold bootstrapping of the data to obtain confidence intervals. As this is a time-consuming intensive process (if **grofit** is used), it may be split over several cores on a multicore machine if the **multicore** R package is available by setting the cores argument to a value larger than one.

One can also specify different spline fitting methods using `method = "spline"`. Options such as the spline type, the number of knots used for the spline and other options for the splines can be easily set using the function `set_spline_options` (which can only be used in conjunction with `method = "spline"`). To essentially reproduce the results from `method = "grofit"` we use smoothing splines (and for the sake of computing time in this vignette we only use 10 bootstrap replicates to compute confidence intervals for the parameter estimates):

```
R> op <- set_spline_options(type = "smooth.spline")
R> vaas_1.aggr2 <- do_aggr(vaas_1, boot = 10, method = "spline", options = op)
```

Other spline types can be specified *via* the `type` argument in the function `set_spline_options`.

3.6. Manipulation of OPM and OPMS data

In the **opm** manual, the functions relevant for retrieving information contained in OPM or OPMS objects are included in a family called "getter-functions" with according cross-references.

For instance, the user may wish to select specific wells from the input plates, which are present in a 96-well layout, numbered from A01 to H12. The function `dim()` provides the dimensions of an OPMS object as a three-element vector comprising (i) number of contained OPM or OPMA objects, (ii) the number of time points (of the first contained plate; these values need not be uniform within an OPMS object), and (iii) the number of wells (which must be uniform within an OPMS object).

To extract, for example, only the data from wells G11 and H11 together with the negative-control well A01 from the dataset `vaas_et_al` the bracket operator defined for the OPMS class has to be invoked as follows:

```
R> data("vaas_et_al", package = "opmdata")
R> vaas.small <- vaas_et_al[, , c("A01", "G11", "H11")]
R> dim(vaas.small)
```

R users should be familiar with this subsetting style, which was modelled after the style for multidimensional arrays, even though the internal representation is quite different.

After metadata have been added, OPM and OPMS objects can be queried for their content. Specialized infix operators `%k%` and `%q%` (for `%K%` and `%Q%` see the **opm** manual) have been modelled in analogy to R's `%in%` operator. The user may be interested whether an OPM or OPMS object contains a specific value associated with a specific metadata key, or the key associated with any value, or combinations of keys and/or values. `%k%` allows the user to search in the metadata keys. The user can test whether all given keys are present as names of the metadata. `%q%` tests whether all given query keys are present as names of the metadata and refer to the same query elements.

Some examples using `vaas_et_al` are given in the following. This OPMS object contains a metadata key *Experiment* with the three possible values *Time series*, *First replicate*, and *Second replicate*, and a metadata key *Species* with either *Escherichia coli* or *Pseudomonas aeruginosa* as values.

```
R> data("vaas_et_al", package = "opmdata")
```

Which plates within `vaas_et_al` have *Experiment* as metadata key?

```
R> "Experiment" %k% vaas_et_al
```

Which plates within `vaas_et_al` have *Experiment* and *Species* as metadata key?

```
R> c("Experiment", "Species") %k% vaas_et_al
```

Which plates within `vaas_et_al` have *Experiment* and *Species* as metadata key with the respective values *First replicate* and *Escherichia coli*?

```
R> c(Experiment = "First replicate",
     Species = "Escherichia coli") %q% vaas_et_al
```

Which plates within `vaas_et_al` have *Species* as metadata key associated with the value *Escherichia coli* or the value *Bacillus subtilis*?

```
R> list(Species = c("Escherichia coli", "Bacillus subtilis")) %q% vaas_et_al
```

In addition to conducting queries with alternatives, using lists as queries would also allow for nested queries (as the metadata entries could also be nested). The results of these infix operators are reported as logical vector with one value per plate; the usual R functions such as `all()`, `any()` or `which()` could be applied to work on these vectors. They could also be used directly as the first argument of the bracket operator for OPMS objects to create subsets:

```
R> vaas.e.coli.1 <- vaas_et_al[c(Experiment = "First replicate",
     Species = "Escherichia coli") %q% vaas_et_al]
```

Alternatively, the user may wish to subset a certain part of the data set using the function `subset()`, which is based on these kinds of querying for metadata keys and their values. Prior to this, the user could check the keys of the metadata:

```
R> data("vaas_et_al", package = "opmdata")
R> metadata_chars(vaas_et_al, values = FALSE)
```

The values in the metadata could be obtained by using `values = TRUE`. Additionally, the user can check the values of special keys in the metadata:

```
R> metadata(vaas_et_al, "Species")
```

The resulting vectors could then also be used for mapping old metadata keys or values to novel ones (for details see Section 2.3).

The presented plotting results of `xy_plot()` and `level_plot()` (see Section 3.7) show selected subsets of `vaas_et_al`. In our example below, the function `subset()` extracts the plates which contain the value *First replicate* in the metadata key *Experiment* and the value *6* in the key *Plate number*, resulting in one representative technical repetition and thus four plates (because four strains were involved) from the data set `vaas_et_al`:

```
R> vaas.1.6 <- subset(vaas_et_al,
     query = list(Experiment = "First replicate", 'Plate number' = 6))
```

Providing the desired combination of metadata keys and values as a list offers a maximum of flexibility, but other approaches are also implemented, as well as the selection of plates based on the presence of keys only (like `%k%` described above; it makes not much sense for `vaas_et_al` whose plates are uniform regarding the keys), and nested queries (like `%q%` with a list described above; makes of course more sense if the metadata contain nested entries). The `subset()` function also has a "time" argument that allows one to create a subset containing only the time points that were common to all plates. This is useful because deviations regarding the overall measurement hours might exist. See the manual for details.

In addition to plate-wise querying and subsetting of OPMS objects, a number of conversion functions for selected content of all plates have been implemented. The **opm** manual lists them in a family of functions called "conversion-functions" with according cross-references. For instance, the user may wish to explore the aggregated curve parameters (lag phase λ, steepness of the slope μ, maximum curve height A, and area under the curve AUC). These may be exported either as matrix or data frame using `extract()`:

```
R> vaas.mu <- extract(vaas_et_al, dataframe = TRUE,
     as.labels = NULL, subset = "mu")
```

To extract also the full or partial set of metadata, it is sufficient to add a list of desired metadata:

```
R> vaas.mu <- extract(vaas_et_al, dataframe = TRUE,
     as.labels = list("Experiment","Number of sample time point",
       "Plate number", "Slot", "Species", "Strain", "Time point in min"),
     subset = "mu")
```

This only works if this meta-information is present for the plates under study. Once a data frame is exported, these metadata will be contained in additional columns; once a matrix is exported, they will be used to construct the row names.

3.7. Plotting functions

Plotting functions for raw data

In the **opm** manual, the functions relevant for plotting are contained in a family called, well, "plotting-functions" with according cross-references. The function `xy_plot()` displays the respiration curves as such (see Figure 6). In our example the selected OPMS object `vaas.1.6` is the subset of the dataset `vaas_et_al` constructed in Section 3.6:

```
R> xy_plot(vaas.1.6, main = "E. coli vs. P. aeruginosa",
     include = list("Species", "Strain"))
```

Using the argument `main`, the user can include a main title in the plot; if it is omitted, by default the title is automatically constructed from the plate type. Likewise, the well coordinates are automatically converted to substrate names (details can be set using additional arguments). The content of the legend (mainly a description of the assignment of the colours

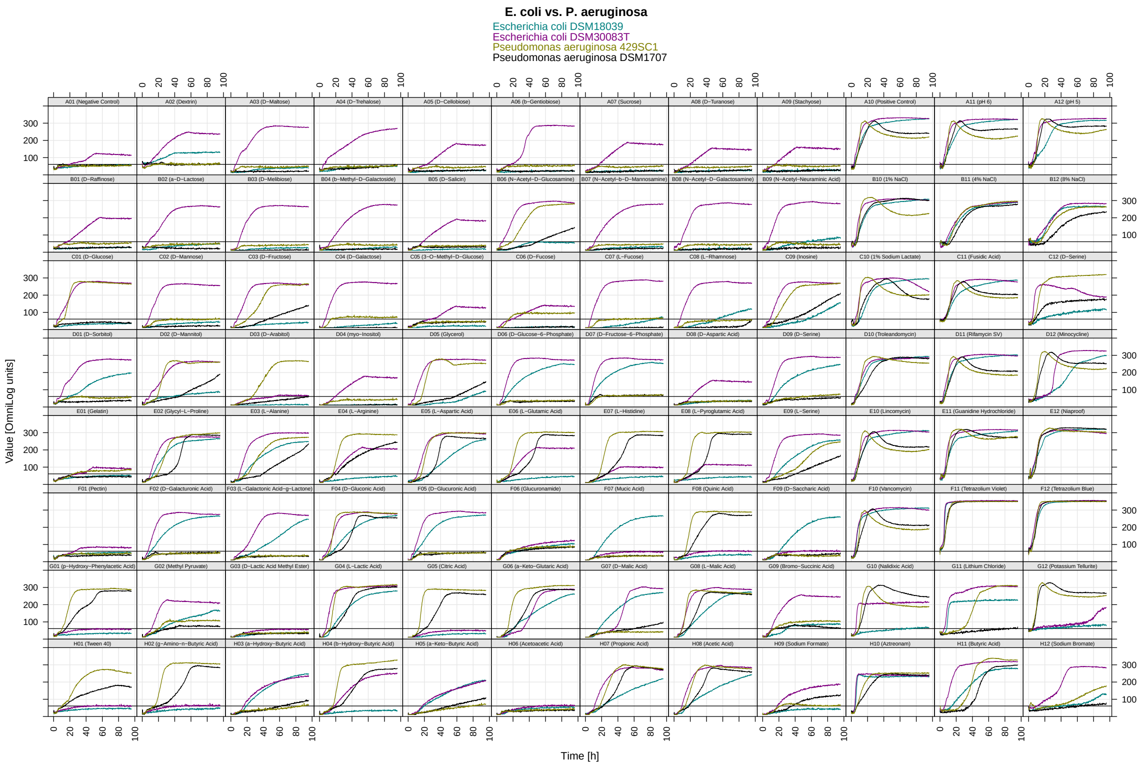


Figure 6: PM curves from the 6th technical repetition of the first biological repetition plotted using `xy_plot()` and by default arranged according to the factual plate layout. The respective curves from all four strains are superimposed; the affiliation to each strain is indicated by colour (see the legend). The x-axes show the measurement time in hours, the y-axes the measured colour intensities in OmniLog® units.

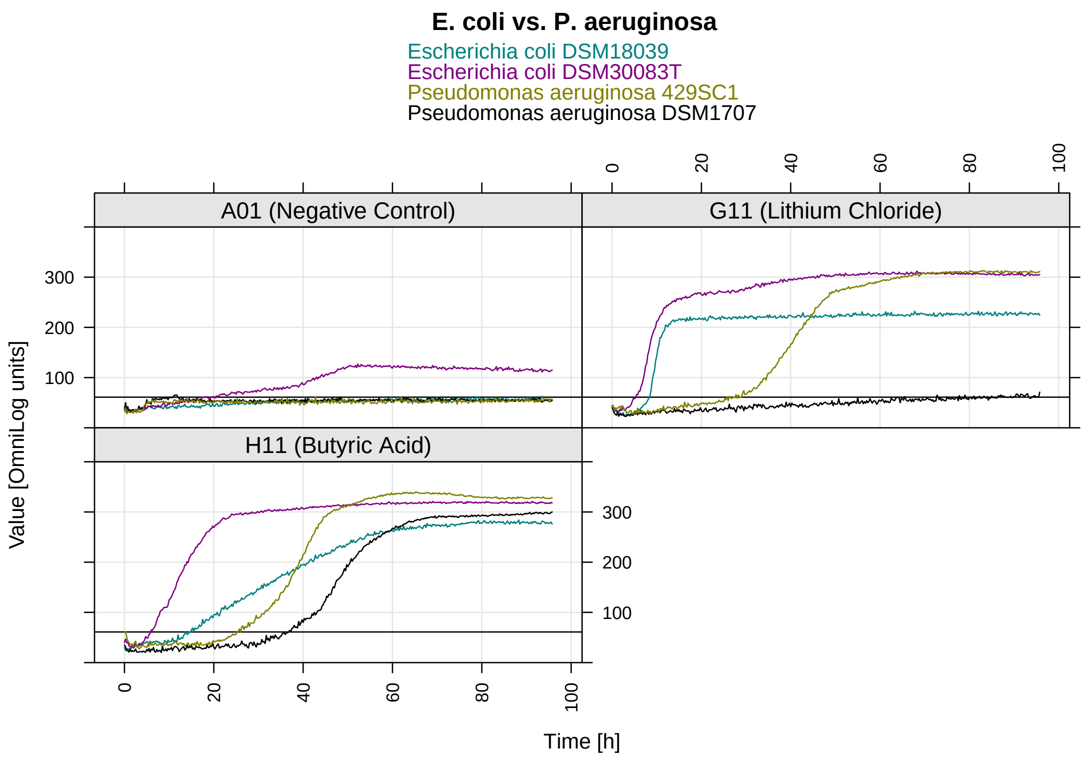


Figure 7: Selected PM curves from the 6th technical repetition from the first biological repetition plotted using `xy_plot()`. The respective curves from all four strains are superimposed, the affiliation to each strain indicated by colour (see the legend). The x-axes show the measurement time in hours, the y-axes the measured colour-value units.

to the curves) is also determined automatically. The argument `include` refers to the metadata and allows the user to choose which entries should be used for assigning curve colours and accordingly be included in the legend. In the example the combination of species and strain is used, yielding four distinct colours. If `include` is not used, the colours are assigned per plate.

The plotting of sub-panels (see Figure 7) works in the same way; the only difference is the previous manipulation of the dataset:

```
R> xy_plot(vaas.1.6[, , c("A01", "G11", "H11")],
     main = "E. coli vs. P. aeruginosa", include = list("Species", "Strain"))
```

The function `level_plot()` (see Figure 8) provides false-colour level plots from the raw respiration measurements over time:

```
R> level_plot(vaas.1.6, main = "E. coli vs. P. aeruginosa",
     include = list("Species", "Strain"))
```

Again, a main title can be set explicitly. Furthermore, the argument `include` again refers to the metadata and allows the user to choose the information to be included in the header for annotating the plates. In the example the combination of species and strain is used.

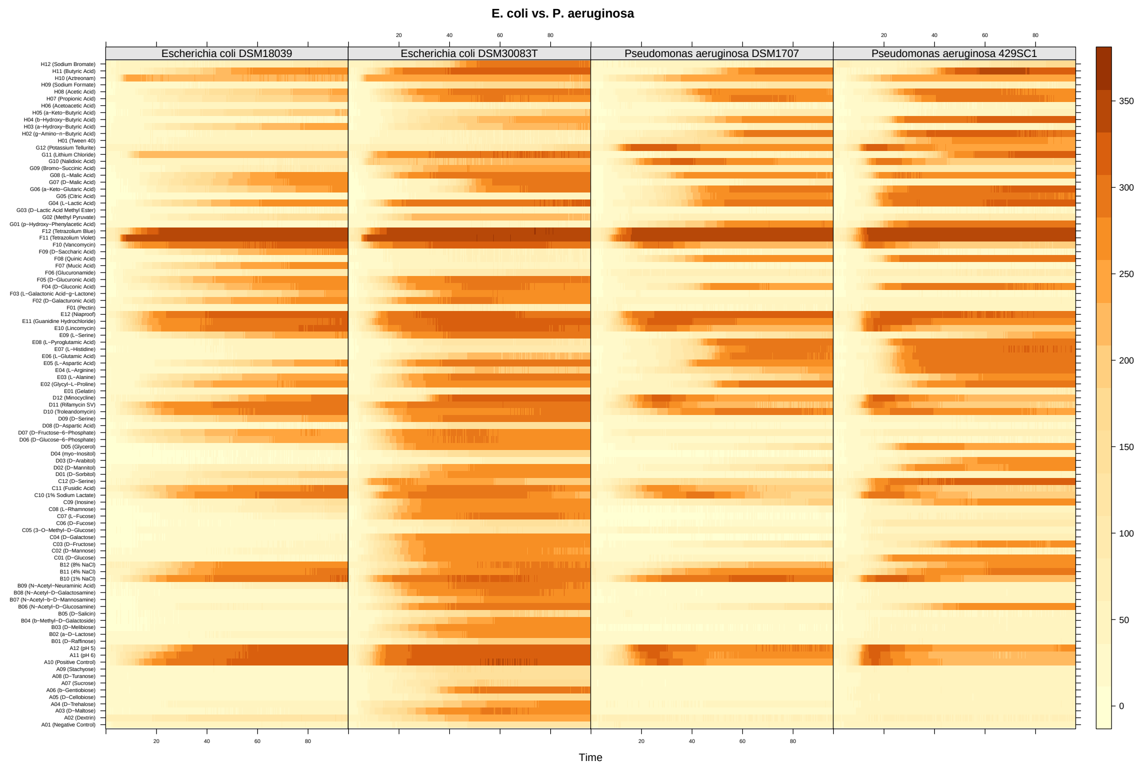


Figure 8: Visualization of PM curves using the function `level_plot()`. Each respiration curve is displayed as a thin horizontal line, in which the curve height as measured in colour-value units is represented by color intensity (darker parts indicate higher curves). The x-axes correspond to the measurement time in hours.

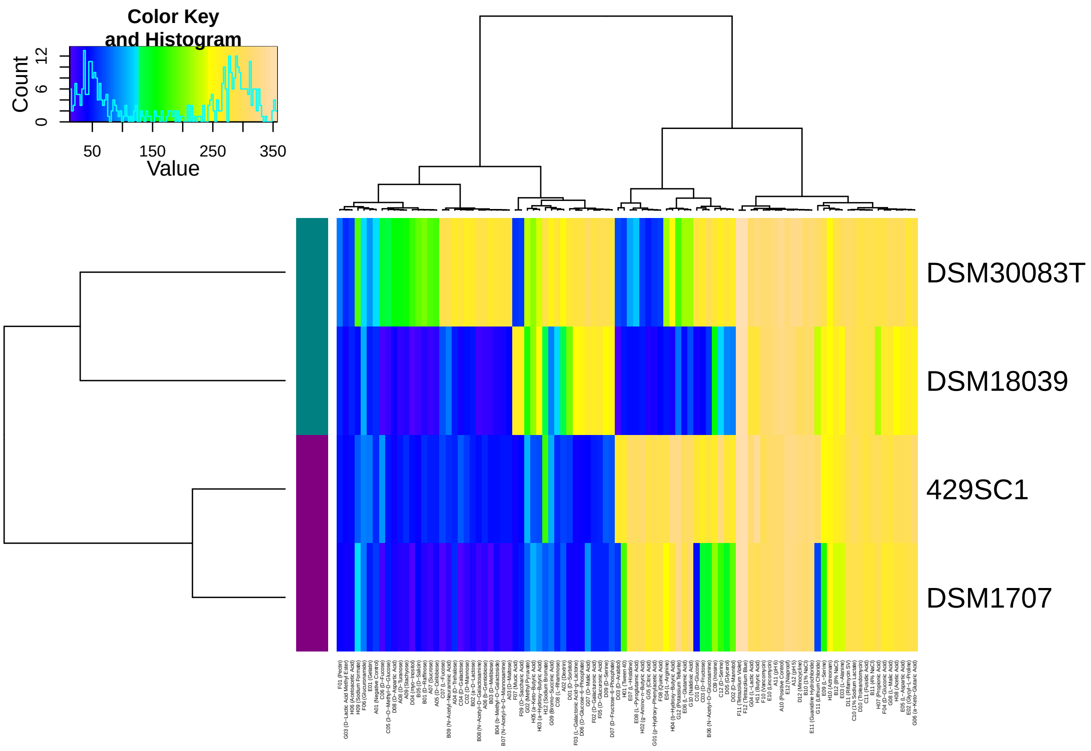


Figure 9: Visualization of the clustered results from the curve parameter maximum height (A) for each substrate using the function `heat_map()`. The x-axis corresponds to the substrates clustered according to the similarity of their values over all plates; the y-axis corresponds to the plates clustered to the similarity of their values over all substrates. As row labels, the strain names were selected, whereas the species affiliations was used to assign row group colours (bars at the left side). The central rectangle is a substrate x plate matrix in which the colours represent the classes of values. The default colour setting uses topological colours, with deep violet and blue indicating the lowest values and light brown indicating the highest values.

Plotting the aggregated data

The function `heat_map()` (see Figure 9) provides false-colour level plots in which both axes are rearranged according to clustering results. In the context of PM data, it makes most sense to apply it to the estimated curve parameters. This **opm** function is a wrapper for `heatmap()` from the **stats** and `heatmap.2()` from the **gplots** package with some adaptations to PM data. For instance, row groups can be automatically constructed from the metadata. The function must be applied to matrices or data frames constructed using `extract()`:

```
R> vaas.1.6.A <- extract(vaas.1.6, as.labels = list("Species", "Strain"),
     dataframe = TRUE)
R> vaas.1.6.A.hm <- heat_map(vaas.1.6.A, as.labels = "Strain",
     as.groups = "Species")
```

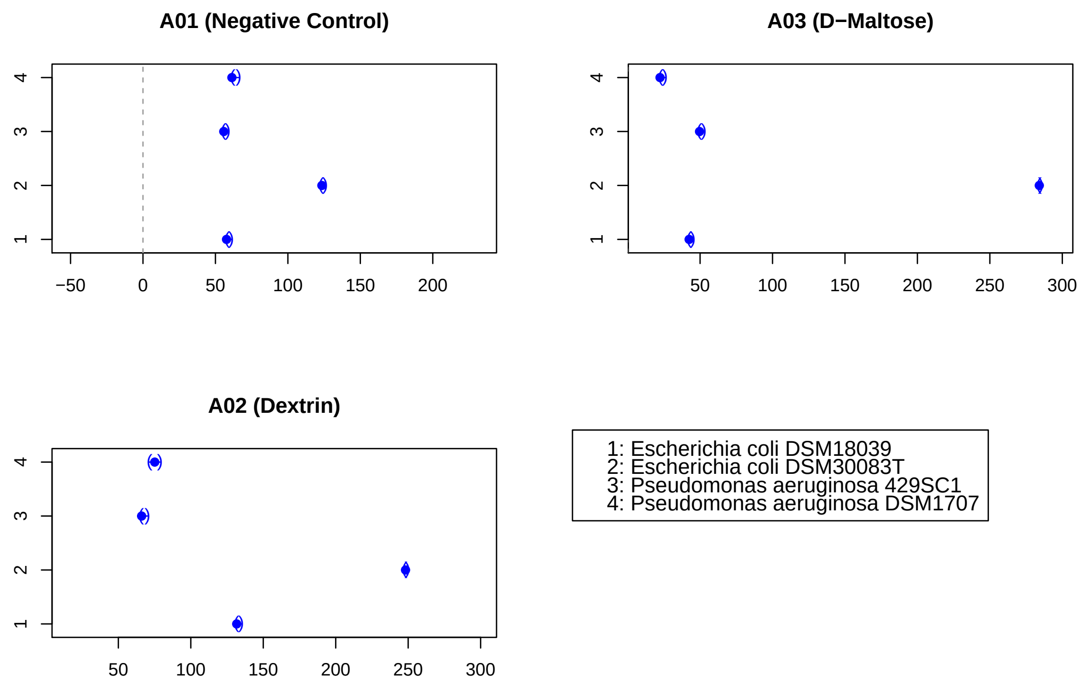


Figure 10: Comparison of point estimates and their 95% confidence intervals for the parameter maximum height (A) observed from four strains, using `ci_plot()`. Shown are the results on the three wells A01 (negative control), A02 (Dextrin) and A03 (D-Maltose) as indicated by the sub-plot titles.

Lea, please provide examples for the use of `ci_plot()` in conjunction with `extract()`. You can use the examples from the manual here because they are unsuitable for the manual. Afterwards, include suitable examples in the manual.

The `ci_plot()` function (see Figure 10) provides a visualization of the point estimator and its 95% confidence interval calculated *via* bootstrapping during aggregation of curves into parameters. The user is free to select the subsets of interest *via* the bracket operator as described above (see Section 3.6):

```
R> ci_plot.legend <- ci_plot(vaas.1.6[, , 1:3],
     as.labels = list("Species", "Strain"), subset = "A",
     legend.field = NULL, x = 150, y = 3)
```

3.8. Statistical comparisons of group means

Vorschlag Johannes: Wir sollten hier ein plakatives Beispiel nehmen, in dem wir sowohl Gruppenvergleiche haben, die nicht verschieden voneinander sind, wie auch Gruppenvergleiche, die signifikant verschieden voneinander sind und bei denen man auch einen Unterschied in der effect size sehen kann. Daher sollten wir auf den `vaas_et_al` Datensatz zurueckgreifen, wie unten in blauer Farbe beschrieben. Die beiden Beispiel-Figures von Lea (momentan Fig. 12 und 13) halte ich nicht fuer so ideal, denn Fig. 12 beinhaltet nur einen paarweisen Vergleich, kein multiple comparison, und in Fig. 13 gibt es keinen Unterschied bei irgendeinem der paarweisen Vergleich - es wird dem Leser das Potential einer solchen Abbildung nicht vollstaendig klar.Ich habe mir trotzdem erlaubt, bei diesen und einigen anderen Figures etwas die Groesse anzupassen.

Style in the next paragraphs seems a bit too colloquial. At lot of "let us", for instance. Other parts repeat a lot of text from the method section. The variable names should be in the style used above in this vignette. Please revise. Also, please consider the TODO entries in the code for opm_mcp().

To demonstrate the functionality of `opm_mcp()`, a comparison of four different bacteria, with each ten replicates using GEN-III microplates, will be utilized. Do these four bacteria differ in, e.g., the mean value of, e.g., curve parameter A of, e.g, well A1? Here, a statistical comparison of four groups (four organisms) of each 10 values (curve parameter A of 10 replicates of well A1) will be performed. First, from the data set `vaas_et_al`, a subset of the ten technical replicates of GEN-III plates from the first biological replicate and also for the curves from well G06 only will be made.

```
R> vaas.subset.mcp <- subset(vaas_et_al[, , "G06"],
     list(Experiment = "First replicate"))
```

For each of the four strains, ten curves are shown (see Figure 11).

```
R> xy_plot(vaas.subset.mcp, include = ~ Strain, neg.ctrl = FALSE,
     legend.fmt = list(space = "right"))
```

The statistical question to be addressed is the following. Is there a significant difference between any of the four strains, based on the mean of the curve parameter A values as determined for each of the ten curves? Here, a multiple comparison of group means is performed using `opm_mcp()`. The groups to be compared are defined by the argument "model". The argument "m.type" specifies the model types to use in model fitting, e.g., a linear model (lm), a generalized linar model (glm), or an analysis of variance (aov). The argument "mcp.def" basically defines the type of contrast matrix to be used for the multiple comparison. In principle, the contrast matrix defines what type of comparison should be made. Concrete, among the groups defined by the argument "model", which of these should actually be included in pariwise comparisons? A detailed explanation of contrast matrices is shown further below.
Lea, is this the correct idea?

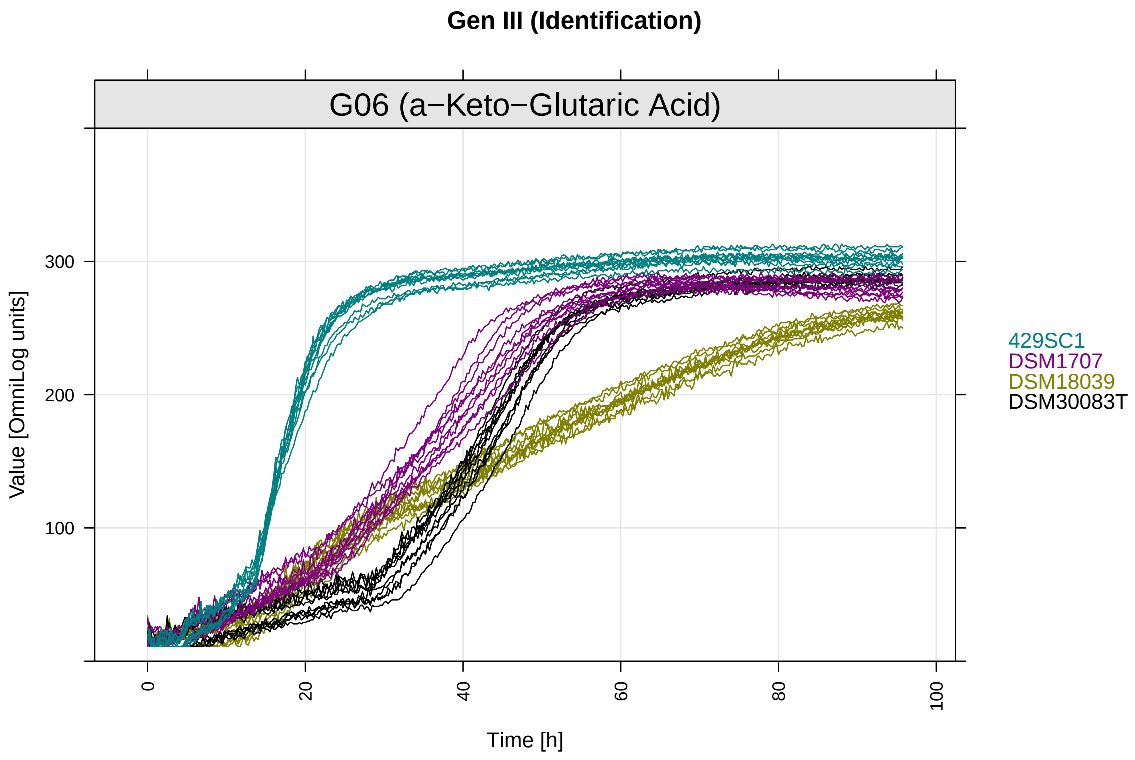


Figure 11: PM curves from the ten technical replicates of the first biological repetition plotted using `xy_plot()`. The respective curves from all four strains are superimposed; the affiliation to each strain is indicated by colour (see the legend). The x-axes show the measurement time in hours, the y-axes the measured colour intensities in OmniLog® units.

```
R> (x <- opm_mcp(vaas.subset.mcp, model = ~ Strain, m.type = "aov",
     mcp.def = mcp(Strain = "Tukey")))
```

The results shown below indicate that all group means are significantly different from each other, except for the comparison of strains DSM30083T and DSM1707:

```
R> summary(x)
```

Is it possible to show here at least part of the output from `summary(x)`? We surely do not need the long list at the beginning, but would it be possible to show the output containing the estimates, the p-values, etc?

The results of the statistical test can be plotted (see Figure 12). For example, the mean of curve parameter A values of strain DSM 30083T is 26.615 units larger than the mean of A values of strain DSM18039 (see the estimate results above, provided by `summary(x)`). Therefore, point estimator of difference between these two strain is plotted at position 26.615 on the x-axis. In pairwise comparisons where the 95% confidence interval crosses the vertical dashed line at zero there is no significant difference between the group means. The more distant the 95% confidence interval is from the dashed zero line, the larger the biological effect of the difference, irrespective of the size of the p-value. For example, the results from the statistical calculations indicate that all significant differences are highly significant ($p < 0.001$). However, the position of the 95% confidence interval indicates that the difference between strains DSM18039 and 429SC1 is much larger than between strains DSM30083T and 429SC1.

```
R> op <- par(no.readonly = TRUE) # default plotting settings
R> par(mar = c(3, 15, 3, 2))
R> plot(x)
R> par(op)
```

Kommentar Johannes: Bis hierhin hatten wir ein sehr simples und straight-forward example. Nun koennte Lea, wie schon von ihr angedeutet, die Spezialitaeten erlaeutern, z.B. was eine contrast matrix ist, ect. Anschliessend koennten wir nochmal ein echt komplexes Beispiel bringen, in dem man die volle Wucht der moeglichen Argumente in `opm_mcp()` ausnutzt, so dass der Leser nocheinmal wirklich vor Augen gefuehrt bekommt, was man mit `opm_mcp()` so alles Feines machen kann.

TODO Lea: here the application of `opm_mcp()` with example and output-graphics. reshaping the data

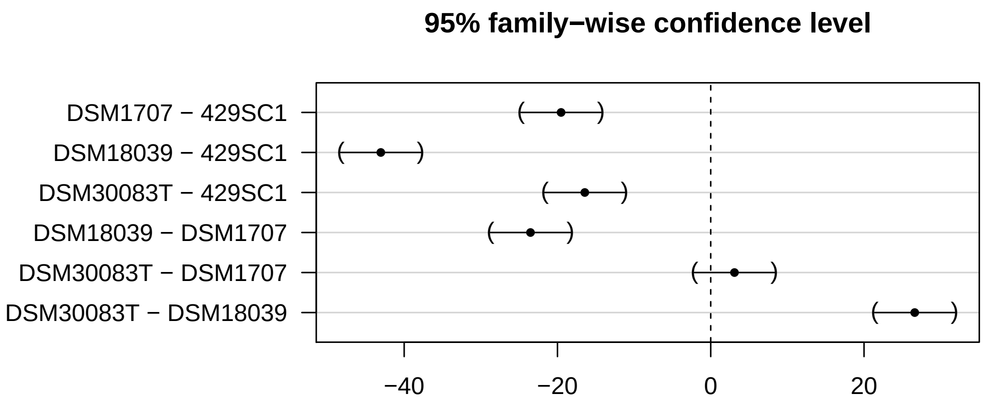


Figure 12: 95% confidence interval for comparison of group means between the strains. All pairwise comparisons are shown. The filled black circle indicates the point estimator of difference betweeen the mean of groups.

```
R> data(vaas_4)
R> x <- opm_mcp(vaas_4, model = ~ Species + Strain,
     do.mcp = FALSE)
```

Explain contrasts, refer to `contrMat() from multcomp`

comparison according metadata 'Species'

```
R> x <- opm_mcp(vaas_4, model = ~ Species, m.type = "lm",
     mcp.def = mcp(Species = "Dunnett"))
```

Explain Confidence intervals for differences of group means, point out the difference to the 95%CI of the curve-parameter estimators. Figure: informative plotting of the CI:

```
R> op <- par(no.readonly = TRUE) # default plotting settings
R> par(mar = c(3, 20, 3, 2))
R> plot(x)
R> par(op)
```

more sophisticated with maually defined set of contrast:

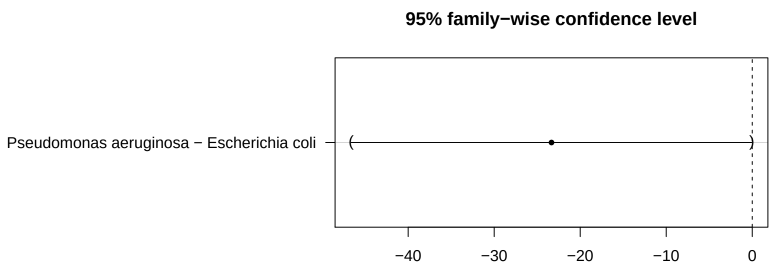


Figure 13: 95% confidence interval for comparison of group means between the species.

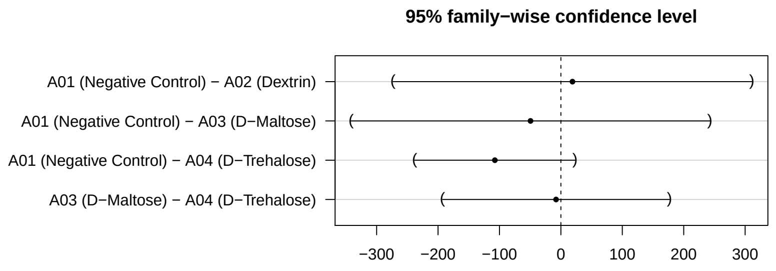


Figure 14: 95% confidence interval for manyally chosen comparison of group means on a specifically selected set of wells.

```
R>  contr <- rbind(
      "A01 (Negative Control) - A02 (Dextrin)" = c(1, -1, 0, 0),
      "A01 (Negative Control) - A03 (D-Maltose)" = c(-1, 0, 1, 0),
     "A01 (Negative Control) - A04 (D-Trehalose)" = c(-1, 0, 0, -1),
      "A03 (D-Maltose) - A04 (D-Trehalose)" = c(0, 0, 1, -1))
R>  x <- opm_mcp(vaas_4[, , 1:4],
       model = ~ Well, m.type = "lm", mcp.def = contr)
```

3.9. Discretization and phylogenetic data export

After suitable subsetting and extraction of one of the curve parameters, data can be discretized and optionally also be exported for analysis with external phylogeny software. In the **opm** manual, the functions relevant for either task are contained in the families "discretization-functions" and "phylogeny-functions" with according cross-references. Restricting the `vaas_et_al` example dataset to the two biological replicates yields an orthogonal dataset with 2×10 replicates for each of the four strains for which we can calculate discretized parameters:

```
R> vaas.repl <- subset(vaas_et_al,
     query = list(Experiment = c("First replicate", "Second replicate")))
R> vaas.repl <- do_disc(vaas.repl)
```

Note that the resulting objects is an OPMS object with OPMD objects as elements. Such objects contain discretized values available *via* `discretized()` and the discretization settings used, which can be obtained using `disc_settings()`. This works much like `aggregated()` and `aggr_settings()` explained above. `disc_settings()` also yields the computed discretization cutoffs. The `subset()` function has a `positive` argument that allows one to create a subset containing only the wells that were positive in at least one plate or in all plates, as well as a `negative` argument. See the manual for details.

Discretization and export of text

The `listing()` methods of the OPMD and OPMD classes create textual descriptions of the discretization results suitable for the direct inclusion in scientific manuscripts.

```
R> listing(vaas.repl, as.groups = NULL)
R> listing(vaas.repl, as.groups = list("Species"))
```

As usual, the results can be grouped according to specified metadata entries using the "as.groups" argument. If this yields ambiguities (such as a negative reaction of the same well on one plate and a positive reaction on another plate), the result is accordingly renamed. The "cutoff" argument can be used to define filters, keeping only those values that occur in a specified minimum proportion of wells. See the manual for details.

The `listing()` function returns a character vector or matrix with the S3 class "OPMD_listing" or "OPMS_listing", allowing for a special `phylo_data()` function that further formats these objects. Accordingly, the following code snippets

```
R> phylo_data(listing(vaas.repl, as.groups = NULL))
R> phylo_data(listing(vaas.repl, as.groups = list("Species")))
```

would yield character scalars better suitable for exporting into text files using `write`. It is also possible to generate HTML output, yielding formatted text. Try

```
R> phylo_data(listing(vaas.repl, as.groups = NULL, html = TRUE))
R> phylo_data(listing(vaas.repl, as.groups = list("Species"), html = TRUE))
```

and note that the `phylo_data()` function has a "html.args" argument. Textual HTML output supports most of the formatting instructions for the output of HTML tables described below (see 3.9.2). Note particularly how formatting *via* a CSS file works.

The default settings of `do_disc()` imply exact k-means partitioning into three groups ("negative", "ambiguous" and "positive"), treating all contained plates together. Let A_1 and A_2 be the maximum-height parameters from two curves C_1 and C_2, respectively, and let us assume that $A_1 \geq A_2$ holds. The algorithm then guarantees that if C_2 is judged as positive reaction then C_1 is also judged as positive; if C_2 is weak then C_1 is not negative; if C_1 is negative then C_2 is negative; and if C_1 is weak then C_2 is not positive. There are not many other things the algorithm guarantees. Note particularly that always three clusters result by default (one can omit the middle cluster, i.e. the "weak" reactions), irrespective of the input data. That is, additionally checking the curve heights and particularly the "cutoffs" entry obtained *via* `disc_settings()` should be mandatory.

The manual describes the other discretization approaches available in **opm**, such as using `best_cutoff()` instead of k-means partitioning, and using subsets of the plates, specified using stored metainformation.

Discretization and export of tables

The HTML created by **opm** deliberately contains no formatting instructions. Rather, it is possible (and recommended) to link it to a CSS file. As the generated HTML is richly annotated with "class" attributes, which not only provide information on the structure of the file but also on the depicted data, very specific formatting can be obtained just by modifying one to several associated CSS files.

For the following example, we set the default CSS file to be linked from the generated HTML to the file that comes with **opm**.

```
R> opm_opt(css.file = grep("[.]css$", opm_files("auxiliary"), value = TRUE))
```

One could now easily create an HTML table from the discretized data and write it to a file:

```
R> vaas.html <- phylo_data(vaas.repl, format = "html",
     as.labels = list("Species", "Strain"), outfile = "vaas.html")
```

A practical problem is that the resulting HTML file is linked to its CSS file with a fixed path. The formatting would thus get lost once the HTML file was copied to another system, without a warning. So users might want to copy the predefined CSS file to the working directory and set it as default:

```
R> file.copy(grep("[.]css$", opm_files("auxiliary"), value = TRUE),
     "opm_styles.css", overwrite = TRUE)
R> opm_opt(css.file = "opm_styles.css")
```

The generated HTML would subsequently be linked to this file, and the two files could be distributed together. The same mechanism works for text generation using `listing()` (see 3.9.1).

Users who want to define their own CSS files can start with modifying the file shipped with **opm**. Microsoft Windows users should consider that the path to the file must be provided in UNIX style, as obtained, e.g., using `normalizePath(x, winslash = "/")` if `x` is the path to the file. This is according to World Wide Web standards and not determined by **opm**.

By default columns with measurement repetitions as specified using `as.labels` are joined together. The `delete` argument specifies how to reduce the table: either not at all or keeping only the variable, parsimony-informative or non-ambiguous characters. The legend of the table is as used in taxonomic journals such as IJSEM but could also be adapted. Users can modify the headline, add sections before the table legend, or before or after the table. The title and the "meta" entries of the resulting HTML can also be modified. The `phylo_data()` methods have an auxiliary function, `html_args`, which assists in putting together the arguments that determine the shape and content of the HTML output. See the manual for further details.

Fine-tuning the discretization

One can also conduct discretization step-by-step by using the functions `best_cutoff()` or `discrete()` after extracting matrices from the OPMS object. This offers more flexibility (such as additional discretization approaches, e.g. the creation of multiple-state characters) but is also more tedious.

```
R> vaas.repl <- subset(vaas_et_al,
     query = list(Experiment = c("First replicate", "Second replicate")))
R> vaas.repl <- extract(vaas.repl,
     as.labels = list("Species", "Strain", "Experiment", "Plate number"))
```

The A parameter can be discretized into (per default) 32 states using the theoretical range of 0 to 400 OmniLog® units (see Section 2.9):

```
R> vaas.repl.disc <- discrete(vaas.repl, range = c(0, 400))
```

This yields (at most) 32 distinct character states corresponding to the 32 equal-width intervals within 0 and 400. Exporting the data in *extended PHYLIP format* readable by RAxML (Stamatakis *et al.* 2005) would work as follows:

```
R> phylo_data(vaas.repl.disc, outfile = "example_replicates.epf")
```

The other supported formats are PHYLIP, NEXUS and TNT (Goloboff *et al.* 2008). For discretizing the data not in equally spaced intervals but into binary characters including missing data, or ternary characters with a third, intermediary state between ”negative” and ”positive” the gap mode of `discrete()` can be used:

```
R> vaas.repl.disc <- discrete(vaas.repl, range = c(120.2, 236.6), gap = TRUE)
```

Here the range argument provides not the overall boundaries of the data as before (at least as large as the real range), but the boundaries of a zone within the real range of the data corresponding to an area of ambiguous affiliation. That is, values below 120.2 are coded as “0”, those above 236.6 as “1”, and those in between as “?”. The values used above were determined by k-means partitioning of the A values from the `vaas_et_al` dataset (Vaas *et al.* 2012); there is currently no conclusive evidence that they can generally be applied. The last command would result in the treatment of values within the given range as “missing data” (`NA` in R, “?” if exported). To treat them as a third, intermediary character state, set `middle.na` to `FALSE`:

```
R> vaas.repl.disc <- discrete(vaas.repl, range = c(120.2, 236.6),
     gap = TRUE, middle.na = FALSE)
```

The three resulting states, coded as “0”, “1” and “2” (in contrast to “0”, “?” and “1” above) would have to be interpreted as “negative”, “weak” and “positive”. Exporting the data in one of the supported phylogeny formats would work as described above. If the `do_disc()`

function described above calls `discrete()`, then only in gap mode and with `middle.na` set to `TRUE`, yielding a vector or logical matrix.

3.10. Accessing substrate information

The **opm** package contains a number of functions suitable for accessing precomputed information on the substrates of wells and plates. In the manual, these functions are contained in the family "naming-functions" with according cross-references. One usually would start a search by determining the exact spelling of an internally used name with `find_substrate()`:

```
R> (names <- find_substrate(c("Glutamine", "Glutamic acid")))
```

The results is a list containing character vectors with the results for each query name as values. Surprisingly, nothing was found for "Glutamic acid" but several values for "Glutamine". The default `search` argument is "exact", which is exact (case-sensitive) matching of *substrings* of the names. One might want to use "glob" searching mode:

```
R> (names <- find_substrate(c("L-Glutamine", "L-Glutamic acid"), "glob"))
```

But with so-called wildcards, i.e. "*" for zero to many and "?" for a single arbitrary character the search is more flexible:

```
R> (names <- find_substrate(c("*L-Glutamine", "*L-Glutamic acid"), "glob"))
```

This fetches all terms that end in either query character string, and does so case-insensitively. Advanced users can apply the much more powerful "regex" and "approx" search modes; see the manual for details.

Once the internally used names have been found, information on the substrates can be queried such as their occurrences and positions on plates:

```
R> (positions <- find_positions(names))
```

This yields a nested list containing two-column matrices with plate names in the first and well coordinates in the second column. References to external data resources for each substrate name can be obtained using `substrate_info()`:

```
R> (subst.info <- substrate_info(names))
```

By default this yields CAS numbers, but KEGG and Metacyc IDs have also been collected for the majority of the substrates. Another use of `substrate_info()` is to convert substrate names to lower case but protecting name components such as abbreviations or chemical symbols. See the manual for further details.

3.11. Global settings

It is possible to modify settings that have an effect on multiple functions and/or on frequently used arguments globally using `opm_opt()`. It is checked that the novel values inherit from the same class(es) than the old ones. See the manual for details.

4. Discussion and conclusion

The high-dimensional sets of longitudinal data collected by the OmniLog® PM system call for fast and easily applicable (and extendable) data organisation and analysis facilities. The here presented **opm** package for the free statistical software R (R Development Core Team 2011) features not only the calculation of aggregated values (curve parameters) including their (bootstrapped) confidence intervals, but also provides a rather complete infrastructure for the management of raw kinetic values and aggregated curve parameters together with any kind of meta-information of relevance for the user. The analysis toolbox of the package includes the implementation of a fully automated estimation of whether respiration kinetics should be classified as either a "positive" or "negative" (absent) physiological reaction. This dichotomization is apparently of high interest to many users of the OmniLog® PM system but would apparently be extremely biased as long as thresholds are chosen ad hoc and by eye. (Users should nevertheless be aware that loss of information is inherent to discretizing continuous data.) The **opm** package enables the user to produce highly informative and specialized graphical outputs from both the raw kinetic data as well as the curve parameter estimates. In combination with the functionality for annotating the data with meta-information and then selecting subsets of the data, straightforward analyses regarding specific analytical questions can be performed without the need to invoke other R packages.

But since the design of the **opm** objects is not intended to be limited to specific analysis frameworks, the **opm** package works as a data containment providing well organized and comprehensive PM data for further, more specialized analyses using methods from different R packages or other third party software tools including phylogeny software. The generation of S4 objects featuring a rich set of methods as containers for either single or multiple OmniLog® PM plates enables not only the transfer of raw kinetic data into R but also eases their further processing with, for example, other R packages. The complex data bundles can also be exported in YAML format (www.yaml.org), which is a human-readable data serialization format that can be read by most common programming languages and facilitates fast and easy data exchange between laboratories.

These features render the **opm** package to be the first comprehensive toolbox for the management and a broad range of analyses of OmniLog® PM data. Its usage requires some familiarity with R, but is otherwise intuitive and straightforward also for biologists who are not used to command-line based software.

An enhancement of the **opm** package would be to include much more precomputed information about the substrates, thus greatly facilitating data arrangement and hypothesis testing. At the moment only the translation of well coordinates to substrate names is provided, as well as access to CAS, KEGG and Metacyc IDs for the majority of the substrates. More substrate information could be integrated into the package, particularly for arranging the substrate into groups, thus easing testing of phenotypic hypotheses.

Potential for improvement can also be seen in the spline estimation and parameter calculation in the data-aggregation step. One main issue in the spline-fitting procedure is the selection of suitable smoothing parameters. The here presented methods provide a basic framework for this based on methods from the **grofit** package, but could also be improved by the incorporation of approaches to the selection of smoothing parameters *via* cross-validation (Eilers and Marx 1996), generalized cross-validation (Craven and Wahba 1979) and application of information criteria like AIC or BIC (Eilers and Marx 1996) into the fitting procedure (Vaas *et al.* 2012). Last but not least, it might also be useful to provide functionality for a direct cross-talk between **opm** and database management systems. The current version is entirely file-based, and whereas powerful selection mechanisms for both input files and container objects for previously imported PM plates have already been implemented, future version could directly include database access. In the meantime, however, the output YAML format is likely to facilitate the quick establishment of third-party software for importing PM data into a database.

To summarize, we are convinced that the **opm** package already enables the users to analyse OmniLog® PM data in rather unlimited exploratory directions.

5. Acknowledgements

We thank Barry Bochner (BIOLOG Inc.), John Kirkish (BIOLOG Inc.), Andre Chouankam (BIOLOG Inc.), Jan Meier-Kolthoff (DSMZ), and Stefan Ehrentraut (DSMZ) for helpful advice, as well as Victoria Michael (DSMZ) for technical support. This work was supported by the German Research Foundation (DFG) SFB/TRR 51 and by the Microme project within the Framework 7 programme of the European Commission, which is gratefully acknowledged. JS gratefully acknowledges his support by DFG grant SI 1352/1-2.

References

Berger S, Stamatakis A (2010). "Accuracy of Morphology-Based Phylogenetic Fossil Placement under Maximum Likelihood." In *8th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA-10)*. ACS/IEEE, Hammamet, Tunisia.

BiOLOG Inc (2009). *Converter, File Management Software, Parametric Software, Phenotype MicroArray, User Guide, Part 90333.* Biolog Inc., Hayward CA.

Bochner B (2009). "Global Phenotypic Characterization of Bacteria." *FEMS Microbiological Reviews*, **33**, 191–205.

Bochner B, Gadzinski P, Panomitros E (2001). "Phenotype MicroArrays for High Throughput Phenotypic Testing and Assay of Gene Function." *Genome Research*, **11**, 1246–1255.

Bochner B, Savageau M (1977). "Generalized Indicator Plate for Genetic, Metabolic, and Taxonomic Studies with Microorganisms." *Applied and Environmental Microbiology*, **33**, 434–444.

Bretz F, Hothorn T, Westfall P (2010). *Multiple Comparisons Using R.* CRC Press, Boca Raton.

Brisbin I, Collins C, White G, McCallum D (1987). "A New Paradigm for the Analysis and Interpretation of Growth Data: The Shape of Things to Come." *The Auk*, **104**, 552–553.

Broadbent J, Larsen R, Deibel V, Steele J (2010). "Physiological and Transcriptional Response of *Lactobacillus casei* ATCC 334 to Acid Stress." *Journal of Bacteriology*, **192**, 2445–2458.

Chambers J (1998). *Programming with Data.* Statistics and Computing. Springer-Verlag, New York.

Craven P, Wahba G (1979). "Smoothing Noisy Data with Spline Functions." *Numerische Mathematik*, **31**, 377–403.

Dougherty J, Kohavi R, Sahami M (1995). "Supervised and Unsupervised Discretization of Continuous Features." In A Prieditis, S Russell (eds.), *Machine Learning: Proceedings of the fifth international conference.*

Efron B (1979). "Bootstrap Methods: Another Look at the Jackknife." *The Annals of Statistics*, **7**, 1–26.

Eilers P, Marx B (1996). "Flexible Smoothing with B-splines and Penalties." *Statistical Sciences*, **11**, 89–121.

Farris J (1970). "Methods for Computing Wagner Trees." *Systematic Zoology*, **19**, 83–92.

Felsenstein J (2004). *Inferring Phylogenies.* Sinauer Associates, Inc., Sunderland, MA.

Fitch W (1971). "Towards Defining the Course of Evolution: Minimal Change for a Specified Tree Topology." *Systematic Zoology*, **20**, 406–416.

Goloboff P, Farris J, Nixon K (2008). "TNT, a Free Program for Phylogenetic Analysis." *Cladistics*, **24**, 774–786.

Hochberg A, Tamhane Y (1987). *Multiple Comparison Procedures.* Wiley.

Hothorn T, Bretz F, Westfall P (2008). "Simultaneous Inference in General Parametric Models." *Biometrical Journal*, **50**, 346–363. See vignette("generalsiminf", package = "multcomp").

Hsu J (1996). *Multiple Comparisons.* Chapman & Hall, London.

Kahm M, Hasenbrink G, Lichtenberg-Frate H, Ludwig J, Kschischo M (2010). "**grofit**: Fitting Biological Growth Curves with R." *Journal of Statistical Software*, **33**, 1–21. URL: http://cran.r-project.org/web/packages/grofit/.

Mahner M, Kary M (1997). "What Exactly are Genomes, Genotypes and Phenotypes? And what about Phenomes?" *Journal of Theoretical Biology*, **186**, 55–63.

Mayr E (1997). "The Objects of Selection." *Proceedings of the National Academy of Science USA*, **94**, 2091–2094.

Mithani A, Hein J, Preston G (2011). "Comparative Analysis of Metabolic Networks Provides Insight into the Evolution of Plant Pathogenic and Nonpathogenic Lifestyles in *Pseudomonas*." *Molecular Biology and Evolution*, **28**, 483–499.

Quackenbush J (2002). “Microarray data normalization and transformation.” *Nature Genetics*, **32**, 496–501.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL: http://www.R-project.org/.

Reinsch CH (1967). “Smoothing by spline functions.” *Numerische Mathematik*, **10**, 177–183.

Schaarschmidt F, Vaas L (2009). “Analysis of trials with complex treatment structure using multiple contrast tests.” *HortScience*, **44**, 188–195.

Searle SR (1971). *Linear Models*. John Wiley & Sons, New York.

Selezska K, Kazmierczak M, Müsken M, Garbe J, Schobert M, Häussler S, Wiehlmann L, Rohde C, Sikorski J (2012). “*Pseudomonas aeruginosa* Population Structure Revisited under Environmental Focus: Impact of Water Quality and Phage Pressure.” *Environmental Microbiology*, **14**, 1952–1967.

Stamatakis A, Ludwig T, Meier H (2005). “RAxML-III: A fast Program for Maximum Likelihood-Based Inference of Large Phylogenetic Trees.” *Bioformatics*, **21**, 456–463.

Swofford D (2003). *PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4.* Sinauer Associates, Sunderland, Massachusetts.

Tindall B, Kämpfer P, Euzéby J, Oren A (2006). “Valid publication of names of prokaryotes according to the rules of nomenclature: past history and current practice.” *International Journal of Systematic and Evolutionary Microbiology*, **56**, 2715–2720.

Tukey J (1994). “The problem of multiple comparisons.” *unpublished manuscript.* Reprinted in: Braun, H.I. (Ed.), The collected works of John W. Tukey. VIII Multiple Comparisons. Chapman & Hall, New York.

Vaas L, Sikorski J, Michael V, Göker M, Klenk H (2012). “Visualization and Curve Parameter Estimation Strategies for Efficient Exploration of Phenotype MicroArray Kinetics.” *PLoS ONE*, **7**, e34846.

Ventura D, Martinez T (1995). “An Empirical Comparison of Discretization Methods.” In *Proceedings of the Tenth International Symposium on Computer and Information Sciences*, pp. 443–450. Morgan Kaufmann Publishers, San Francisco, CA.

Wang H, Song M (2011). “Ckmeans.1d.dp: optimal k-means clustering in one dimension by dynamic programming.” *The R Journal*, **3**, 29–33.

Wood SN (2003). “Thin Plate Regression Splines.” *Journal of the Royal Statistical Society. Series B*, **65**, 95–114.

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