Introduction to the *TPP* package for analyzing Thermal Proteome Profiling experiments

Dorothee Childs
European Molecular Biology Laboratory (EMBL),
Heidelberg, Germany
dorothee.childs@embl.de

TPP version 1.0.3 (Last revision 2015-07-21)

Abstract

Detecting the binding partners of a drug is one of the biggest challenges in drug research. Thermal Proteome Profiling (TPP) addresses this question by combining the cellular thermal shift assay concept with mass spectrometry based proteome-wide protein quantitation [1]. Thereby, drug-target interactions can be inferred from changes in the thermal stability of a protein upon drug binding, or upon downstream cellular regulatory events, in an unbiased manner.

The analysis of TPP experiments requires several data analytic and statistical modeling steps. The package *TPP* facilitates this process by providing exectuable workflows that conduct all necessary steps. This vignette explains the use of the package. For details about the statistical methods, please refer to the paper [1].

Contents

1 Introduction

To install the package, type the following commands into the *R* console

```
source("http://bioconductor.org/biocLite.R")
biocLite("TPP")
```

The installed package can be loaded by

```
library("TPP")
```

2 TPP-TR and TPP-CCR analysis

The *TPP* package performs two analysis workflows:

1. **Analysis of temperature range (TR) experiments:** TPP-TR experiments combine the cellular thermal shift assay (CETSA) approach with high-throughput mass spectrometry (MS). They provide protein abundance measurements at increasing temperatures for different treatment conditions. The data analysis comprises cross-experiment normalization, melting curve fitting, and the statistical evaluation of the estimated melting points in order to detect shifts induced by drug binding.
2. **Analysis of compound concentration range (CCR) experiments:** TPP-CCR experiments combine the isothermal dose-response (ITDR) approach with high-throughput MS. The CCR workflow of the package performs median normalization, fits dose response curves, and determines the pEC50 values for proteins showing dose dependent changes in thermal stability upon drug treatment.

The following sections describe both functionalities in detail.

3 Analyzing TPP-TR experiments

3.1 Overview

The function `analyzeTPPTR` executes the whole workflow from data import through normalization and curve fitting to statistical analysis. Nevertheless, all of these steps can be invoked separately by the user. The corresponding functions can be recognized by their suffix `tpptr`. Here, we first show how to start the whole analysis using `analyzeTPPTR`. Afterwards, we demonstrate how to carry out single steps individually.

Before you can start your analysis, you need to specify information about your experiments. This information comprises the name, condition (treatment or vehicle), replicate, as well as the isobaric labels and corresponding temperature values for each experiment.

The package retrieves this information from a configuration table that you need to specify before starting the analysis. This table can either be a data frame that you define in your R session, or a spreadsheet in .xlsx or .csv format. In a similar manner, the measurements themselves can either be provided as a list of data frames, or imported directly from files during runtime.

We demonstrate the functionality of the package using the dataset `hdacTR_smallExample`. It contains an illustrative subset of a larger dataset which was obtained by TPP-TR experiments on K562 cells treated with the histone deacetylase (HDAC) inhibitor panobinostat in the treatment groups and with vehicle in the control groups. The experiments were performed for two conditions (vehicle and treatment), with two biological replicates each. The raw MS data were processed with the Python package *isobarQuant*, which provides protein fold changes relative to the protein abundance at the lowest temperature as input for the TPP package.

First, we load the data:

```
data("hdacTR_smallExample")
```

This command loads two objects:

1. `hdacTR_data`: a list of data frames that contain the measurements to be analyzed,

2. hdacTR_config: a configuration table with details about each experiment.

3.2 The configuration table

hdacTR_config is an example of a configuration table in data frame format. The corresponding spreadsheet file is provided together with the package and can be used as a template for your own analysis. It is stored in the folder example_data/TR_example_data in your package installation path. You can locate the example_data folder on your system by typing

```
system.file('example_data', package = 'TPP')
```

```
## [1] "/tmp/Rtmpwpuo2K/Rinst64465632547a/TPP/example_data"
```

Let's take a closer look at the content of the configuration table we just loaded:

```
print(hdacTR_config)
```

```
##       Experiment Condition Replicate 126 127L 127H 128L 128H 129L 129H 130L 130H 131L
## 1      Vehicle_1   Vehicle         1  67   63   59   56   53   50   47   44   41   37
## 2      Vehicle_2   Vehicle         2  67   63   59   56   53   50   47   44   41   37
## 3 Panobinostat_1 Treatment         1  67   63   59   56   53   50   47   44   41   37
## 4 Panobinostat_2 Treatment         2  67   63   59   56   53   50   47   44   41   37
```

It contains the following columns:

- Experiment: name of each experiment.
- Condition: experimental conditions (Vehicle or Treatment).
- Replicate: experimental replicates.
- Label columns: each isobaric label names a column that contains the temperature the label corresponds to in the individual experiments.

An additional Path column must be added to the table if the data should be imported from files instead of data frames.

3.3 The data tables

hdacTR_data is a list of data frames containing the measurements for each experimental condition and replicate:

```
summary(hdacTR_data)
```

```
##                Length Class      Mode
## Vehicle_1      13     data.frame list
## Vehicle_2      13     data.frame list
## Panobinostat_1 13     data.frame list
## Panobinostat_2 13     data.frame list
```

They each contain between 508 and 509 proteins each:

```
data.frame(hdacTR_config[, c("Experiment", "Condition", "Replicate")],
           "No.of Proteins"=sapply(hdacTR_data, nrow), row.names=NULL)
```

```
##       Experiment Condition Replicate No.of.Proteins
## 1      Vehicle_1   Vehicle         1            508
## 2      Vehicle_2   Vehicle         2            509
## 3 Panobinostat_1 Treatment         1            508
## 4 Panobinostat_2 Treatment         2            509
```

Each of the four data frames in hdacTR_data stores protein measurements in a row wise manner. For illustration, let's look at some example rows of the first vehicle group.

```
hdacVehicle1 <- hdacTR_data[["Vehicle_1"]]
head(hdacVehicle1)
```

```
##      gene_name qssm qupm rel_fc_126 rel_fc_127L rel_fc_127H rel_fc_128L rel_fc_128H
## 3286     HDAC1    5    4 0.00510359   0.0207088   0.0512665   0.0840443    0.158568
## 3584    HDAC10    2    1 0.00000000   0.0180900   0.2511430   0.4034580    0.582994
## 3000     HDAC2    7    5 0.02006570   0.0589077   0.0718648   0.1011260    0.556456
## 2089     HDAC3    2    2 0.08706000   0.0891621   0.2103700   0.3226950    0.459124
## 1602     HDAC4    4    4 0.04371190   0.1069100   0.1630480   0.2411050    0.382980
## 607      HDAC6    5    4 0.00176507   0.0260307   0.0449839   0.0759111    0.202110
##      rel_fc_129L rel_fc_129H rel_fc_130L rel_fc_130H rel_fc_131L
## 3286    0.410777    0.622789    0.750158    0.866156           1
## 3584    0.631114    0.769128    1.013330    1.093940           1
## 3000    0.850373    0.842952    0.885415    0.972225           1
## 2089    0.651561    0.626848    0.785872    0.740518           1
## 1602    0.596979    0.831169    0.936279    0.955209           1
## 607     0.366981    0.638239    0.891903    0.932266           1
```

The columns can be grouped into three categories:

- a column with a protein identifier. Called `gene_name` in the current dataset,
- the ten fold change columns all start with the prefix `rel_fc_`, followed by the isobaric labels 126 to 131L,
- other columns that contain additional information. In the given example, the columns `qssm` and `qupm` were produced by the python package *isobarQuant* when analyzing the raw MS data. This metadata will be included in the package's output table. Additionally, it can be filtered according to pre-specified quality criteria for construction of the normalization set. The original results of the *isobarQuant* package contain more columns of this type. They are omitted here to keep the size of the example data within reasonable limits.

3.4 Starting the whole workflow by `analyzeTPPTR`

The default settings of the *TPP* package are configured to work with the output of the python package *isobarQuant*, but you can adjust it for your own data, if desired. When analyzing data from *isobarQuant*, all you need to provide is:

- the configuration table,
- the experimental data, either as a list of data frames, or as tab-delimited `.txt` files,
- a desired output location, for example

```
resultPath = file.path(getwd(), 'Panobinostat_Vignette_Example')
```

If you want to use data from other sources than *isobarQuant*, see section 3.6 for instructions.

By default, plots for the fitted melting curves are produced and stored in pdf format for each protein during runtime and we highly recommend that you do this when you analyse your data. However, producing plots for all 510 proteins in our dataset can be time consuming and would slow down the execution of the current example. Thus, we first disable plotting by setting the argument `plotCurves=FALSE`. Afterwards, we will produce plots for individual proteins of interest. Note that, in practice, you will only be able to examine the results in an unbiased manner if you allow the production of all plots.

We start the workflow by typing

```
TRresults <- analyzeTPPTR(configTable=hdacTR_config, data=hdacTR_data, nCores=2,
                          resultPath=resultPath, plotCurves=FALSE)
```

This performs the melting curve fitting procedure in parallel on a maximum of two CPUs (requirement for package vignettes). Without specifying the `nCores` argument, fitting is performed by default on the maximum number of CPUs on your device.

`analyzeTPPTR` produces a table that summarizes the results for each protein. It is returned as a data frame and exported to an Excel spreadsheet at the specified output location. It contains the following information for each experiment:

- normalized fold changes,
- melting curve parameters,
- statistical test results,

- quality checks on the curve parameters and p-values,
- additional columns from the original input data.

The quality of the result for each protein is determined by four filters. Currently, these criteria are checked only when the experimental setup includes exactly two replicates:

Filter	Column name in result table
1. Is the minimum slope in each of the control vs. treatment experiments < -0.06?	`minSlopes_less_than_0.06`
2. Are both the melting point differences in the control vs treatment experiments greater than the melting point difference between the two untreated controls?	`meltP_diffs_T_vs_V_greater_V1_vs_V2`
3. Is one of the p values for the two replicate experiments < 0.05 and the other one < 0.1?	`min_pVals_less_0.05_and_max_pVals_less_0.1`
4. Do the melting point shifts in the two control vs treatment experiments have the same sign (i.e. protein was either stabilized or destabilized in both cases)?	`meltP_diffs_have_same_sign`

The current example revealed 7 out of 510 proteins that fulfilled all four requirements:

```
tr_targets <- subset(TRresults, fulfills_all_4_requirements)$Protein_ID
print(tr_targets)
```

```
## [1] "BAG2"   "DDB2"   "HDAC10" "HDAC6"  "HDAC8"  "IQSEC2" "STX4"
```

3 of the detected proteins belong to the HDAC family. Because Panobinostat is known to act as an HDAC inhibitor, we select them for further investigation.

```
hdac_targets <- grep("HDAC", tr_targets, value=TRUE)
print(hdac_targets)
```

```
## [1] "HDAC10" "HDAC6"  "HDAC8"
```

We next investigate these proteins by estimating their melting curves for the different treatment conditions. However, we can only reproduce the same curves as before if the data is normalized by the same normalization procedure. Although we only want to fit and plot melting curves for a few proteins, the normalization therefore needs to incorporate all proteins in order to obtain the same normalizaton coefficients as before. The following section explains how to invoke these and other steps of the workflow independently of each other.

3.5 Starting individual steps of the workflow

3.5.1 Data import

Currently, the *TPP* package stores the data in *ExpressionSet*s, and so we convert the data that we have into the needed format. An advantage of the *ExpressionSet* container is its consistent and standardized handling of metadata for the rows and columns of the data matrix. This ability is useful for the given data, because it enables the annotation of each fold change column by temperature values as well as the corresponding isobaric labels. Furthermore, each protein can be annotated with several additional properties which can be used for normalization or processing of the package output.

The function `tpptrImport` imports the data and converts it into *ExpressionSets*:

```
trData <- tpptrImport(configTable=hdacTR_config, data=hdacTR_data)
```

```
## Importing data...
##
## Importing TR dataset:  Vehicle_1
## Removing duplicate identifiers using quality column 'qupm'...
## 508 out of 508 rows kept for further analysis.
##  -> Vehicle_1 contains 508 proteins.
##  -> 504 out of 508 proteins (99.21%) suitable for curve fit (criterium:  > 2 valid fold changes
per protein).
## Importing TR dataset:  Vehicle_2
```

```
## Removing duplicate identifiers using quality column 'qupm'...
## 509 out of 509 rows kept for further analysis.
##  -> Vehicle_2 contains 509 proteins.
##  -> 504 out of 509 proteins (99.02%) suitable for curve fit (criterium:  > 2 valid fold changes
per protein).
## Importing TR dataset:  Panobinostat_1
## Removing duplicate identifiers using quality column 'qupm'...
## 508 out of 508 rows kept for further analysis.
##  -> Panobinostat_1 contains 508 proteins.
##  -> 504 out of 508 proteins (99.21%) suitable for curve fit (criterium:  > 2 valid fold changes
per protein).
## Importing TR dataset:  Panobinostat_2
## Removing duplicate identifiers using quality column 'qupm'...
## 509 out of 509 rows kept for further analysis.
##  -> Panobinostat_2 contains 509 proteins.
##  -> 499 out of 509 proteins (98.04%) suitable for curve fit (criterium:  > 2 valid fold changes
per protein).
```

The resulting object `trData` is a list of *ExpressionSet*s for each experimental condition and replicate. Going back to the example data shown above (vehicle group 1), the corresponding object looks as follows:

```
trData[["Vehicle_1"]]
```

```
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 508 features, 10 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: rel_fc_131L rel_fc_130H ... rel_fc_126 (10 total)
##   varLabels: label temperature normCoeff
##   varMetadata: labelDescription
## featureData
##   featureNames: AAK1 AAMDC ... ZFYVE20 (508 total)
##   fvarLabels: meltPoint inflPoint ... qupm (10 total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation: Vehicle_1 Vehicle 1
```

Each *ExpressionSet* S_i contains the fold change measurements (accessible by `exprs`(S_i)), column annotation for isobaric labels and temperatures (accessible by `phenoData`(S_i)), additional measurements obtained for each protein (accessible by `featureData`(S_i)), and the protein names (accessible by `featureNames`(S_i)).

3.5.2 Data normalization

Whether normalization needs to be performed and what method is best suited depends on the experiment. Currently, the *TPP* package offers the normalization procedure described by Savitski (2014)[1]. It comprises the following steps:

1. In each experiment, filter proteins according to predefined quality criteria.
2. Among the remaining proteins, identify those that were quantified in *all* experiments (jointP).
3. In each experiment, extract the proteins belonging to jointP. Subselect those proteins that pass the predefined fold change filters.
4. Select the biggest remaining set among all experiments (normP).
5. For each experiment, compute median fold changes over the proteins in normP and fit a sigmoidal melting curves through the medians.
6. Use the melting curve with the best R^2 value to normalize all proteins in each experiment.

The function `tpptrNormalize` performs all described steps. It requires a list of filtering criteria for construction of the normalization set. We distinguish between conditions on fold changes and on additional data columns. The function `tpptrDefaultNormReqs` offers an example object with default criteria for both categories:

```
print(tpptrDefaultNormReqs())
```

```
## $fcRequirements
##   fcColumn thresholdLower thresholdUpper
## 1        7            0.4            0.6
## 2        9            0.0            0.3
## 3       10            0.0            0.2
##
## $otherRequirements
##   colName thresholdLower thresholdUpper
## 1    qssm              4            Inf
```

By default, `tpptrNormalize` applies the filtering criteria in `tpptrDefaultNormReqs`. If you want to normalize a dataset in which the column indicating measurement quality has a different name than 'qssm', you have to change the column name and threshold accordingly. Because our example data was produced by *isobarQuant*, we can use the default settings here.

We normalize the imported data as follows:

```
normResults <- tpptrNormalize(data=trData)
```

```
## Creating normalization set:
## 1.  Filtering by non fold change columns:
## Filtering by annotation columns qssm in treatment group:  Vehicle_1
##  Column qssm between 4 and Inf-> 312 out of 508 proteins passed
## 312 out of 508 proteins passed in total.
##
## Filtering by annotation columns qssm in treatment group:  Vehicle_2
##  Column qssm between 4 and Inf-> 362 out of 509 proteins passed
## 362 out of 509 proteins passed in total.
##
## Filtering by annotation columns qssm in treatment group:  Panobinostat_1
##  Column qssm between 4 and Inf-> 333 out of 508 proteins passed
## 333 out of 508 proteins passed in total.
##
## Filtering by annotation columns qssm in treatment group:  Panobinostat_2
##  Column qssm between 4 and Inf-> 364 out of 509 proteins passed
## 364 out of 509 proteins passed in total.
##
## 2.  Find jointP:
## Detecting intersect between treatment groups (jointP).
## -> JointP contains 261 proteins.
##
## 3.  Filtering fold changes:
## Filtering fold changes in treatment group:  Vehicle_1
##  Column 7 between 0.4 and 0.6 -> 30 out of 261 proteins passed
##  Column 9 between 0 and 0.3 -> 223 out of 261 proteins passed
##  Column 10 between 0 and 0.2 -> 233 out of 261 proteins passed
## 22 out of 261 proteins passed in total.
##
## Filtering fold changes in treatment group:  Vehicle_2
##  Column 7 between 0.4 and 0.6 -> 21 out of 261 proteins passed
##  Column 9 between 0 and 0.3 -> 215 out of 261 proteins passed
##  Column 10 between 0 and 0.2 -> 227 out of 261 proteins passed
## 14 out of 261 proteins passed in total.
##
## Filtering fold changes in treatment group:  Panobinostat_1
##  Column 7 between 0.4 and 0.6 -> 34 out of 261 proteins passed
##  Column 9 between 0 and 0.3 -> 217 out of 261 proteins passed
##  Column 10 between 0 and 0.2 -> 224 out of 261 proteins passed
```

```
## 21 out of 261 proteins passed in total.
##
## Filtering fold changes in treatment group:  Panobinostat_2
##  Column 7 between 0.4 and 0.6 -> 15 out of 261 proteins passed
##  Column 9 between 0 and 0.3 -> 221 out of 261 proteins passed
##  Column 10 between 0 and 0.2 -> 225 out of 261 proteins passed
## 10 out of 261 proteins passed in total.
##
## Experiment with most remaining proteins after filtering:  Vehicle_1
## -> NormP contains 22 proteins.
## ----------------------------------
## Computing normalization coefficients:
## 1.  Computing fold change medians for proteins in normP.
## 2.  Fitting melting curves to medians.
## -> Experiment with best model fit:  Vehicle_1 (R2:  0.9919)
## 3.  Computing normalization coefficients
## Creating QC plots to illustrate median curve fits.
## ----------------------------------
## Normalizing all proteins in all experiments.
## Normalization successfully completed!
```

```
trDataNormalized <- normResults[["normData"]]
```

3.5.3 Melting curve fitting

Next we fit and plot melting curves for the detected HDAC targets. We first select the corresponding rows from the imported data:

```
trDataHDAC <- lapply(trDataNormalized, function(d) d[featureNames(d) %in% hdac_targets,])
```

We fit melting curves for these proteins using the function `tpptrCurveFit`:

```
trDataHDAC <- tpptrCurveFit(data=trDataHDAC, resultPath=resultPath, nCores=1)
```

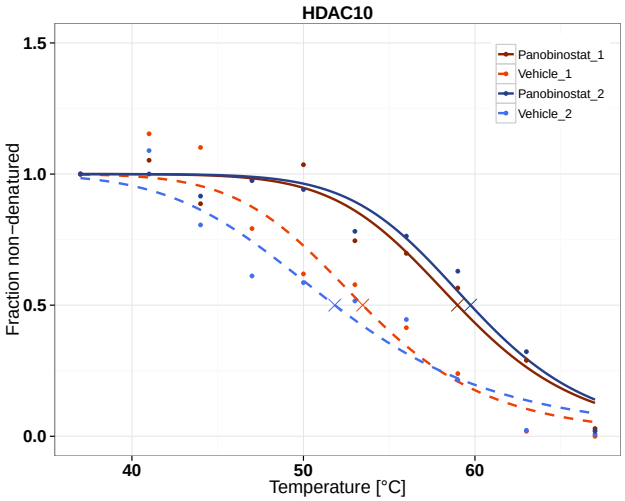
```
## Fitting melting curves to 3 proteins.
## Fitting melting curves for protein:  HDAC10
## Fitting melting curves for protein:  HDAC6
## Fitting melting curves for protein:  HDAC8
## Runtime (1 CPUs used):  1.17 secs
##
## Melting curves fitted sucessfully!
## 12 out of 12 models with sufficient data points converged (100 %).
```

The melting curve parameters are now stored within the `featureData` of the *ExpressionSet*s. For example, the melting curves estimated for the Vehicle group have the following parameters:

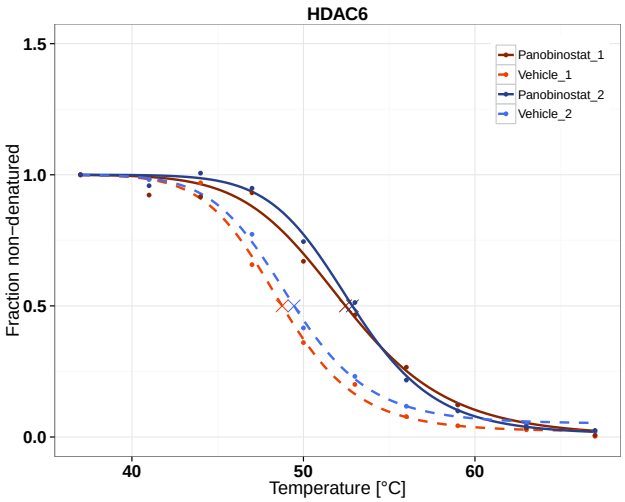
```
pData(featureData(trDataHDAC[["Vehicle_1"]]))[,1:5]
```

```
##        meltPoint inflPoint       slope    plateau      R_sq
## HDAC10  53.45376  52.42318 -0.06759108 0.00000000 0.9467492
## HDAC6   48.75631  48.24751 -0.10942668 0.02084157 0.9960670
## HDAC8   46.48942  45.84351 -0.09146307 0.00000000 0.9962177
```

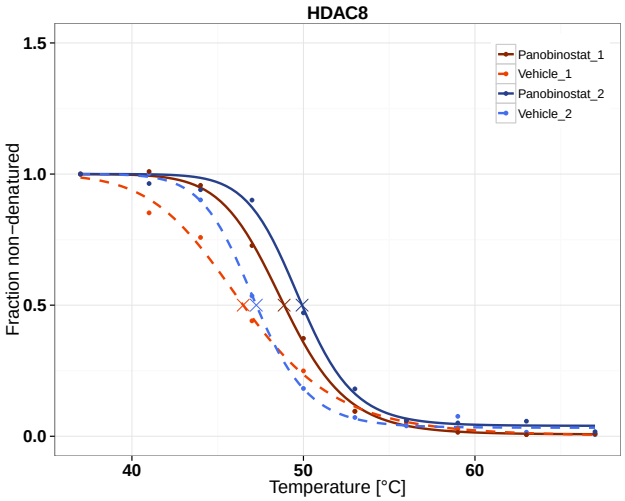
The melting curve plots were stored in subdirectory `Melting_Curves` in `resultPath`. You can browse this directory and inspect the melting curves and their parameters. In the following, you can see the plots that were placed in this directory for the 3 detected targets:



	meltPoint	slope	plateau	R2
Panobinostat_1	58.98	−0.069	0	0.95
Vehicle_1	53.45	−0.068	0	0.95
Panobinostat_2	59.75	−0.071	0	0.96
Vehicle_2	51.83	−0.052	0	0.93



	meltPoint	slope	plateau	R2
Panobinostat_1	52.45	−0.083	0	0.99
Vehicle_1	48.76	−0.11	0.02	1
Panobinostat_2	52.85	−0.1	0.01	1
Vehicle_2	49.46	−0.1	0.05	1



	meltPoint	slope	plateau	R2
Panobinostat_1	48.87	−0.13	0.01	1
Vehicle_1	46.49	−0.091	0	1
Panobinostat_2	49.92	−0.15	0.04	1
Vehicle_2	47.25	−0.15	0.03	1

3.5.4 Significance assessment of melting point shifts

Similar to the normalization explained earlier, significance assessment of melting point shifts has to be performed on the whole dataset due to the binning procedure used for p-value computation. For the given dataset, we have already analyzed all curve parameters by the function `analyzeTPPTR`. Here we show how you can start this procedure independently of the other steps. This can be useful when you only need to re-compute the p-values (for example with a different binning parameter) without the runtime intense curve fitting before.

Melting curve parameter analysis is performed by the function `tpptrResultTable`. It requires a list of *ExpressionSets* with melting curve parameters stored in the featureData. To avoid runtime intensive repetitions of the curve fitting procedure, `analyzeTPPTR` saved these objects as an intermediate result after curve fitting in the subdirectory `/dataObj`. We can access them by the command:

```
load(file.path(resultPath, "dataObj", "fittedData.RData"), verbose=TRUE)
```

```
## Loading objects:
##   trDataFitted
```

This loaded the object `trDataFitted`, which is a list of *ExpressionSets* in which the melting curve parameters have already been stored in the `featureData` by `tpptrCurveFit`.

Now we start the curve parameter evaluation, trying a new bin width of 200 instead of the default value 300:

```
bNew <- 200
TRresultsBNew <- tpptrResultTable(data=trDataFitted, binWidth=bNew)
```

```
## Creating results table.
## Results table created successfully!
```

We can then compare the outcome to the results that we previously obtained with a bin width of 300:

```
tr_targetsBNew <- subset(TRresultsBNew, fulfills_all_4_requirements==TRUE)$Protein_ID
onlyB300 <- setdiff(tr_targets, tr_targetsBNew)
onlyBNew <- setdiff(tr_targetsBNew, tr_targets)
print(onlyB300)
```

```
## [1] "DDB2"
print(onlyBNew)
## [1] "CDK16"
```

3.5.5 Output table

Finally, we export the new results to an Excel spreadsheet:

```
tppExport(tab=TRresultsBNew, file=file.path(resultPath, paste("targets_binWidth",bNew,".xlsx",sep="")))
## Writing results to file:  /tmp/RtmpwpuoZK/Rbuild64465a5b65e2/TPP/vignettes/Panobinostat_Vignette_Exam
## File created successfully!
```

3.6 Analyzing data not produced by the accompanying isobarQuant package

3.6.1 Specifying customized column names for data import

By default, `analyzeTPPTR` looks for a protein ID column named `gene_name`, and a quality control column named `qupm` to assist in the decision between proteins with the same identifier. If these columns have different names in your own dataset, you have to define the new names using the arguments `idVar` and `qualColName`. Similarly, the argument `fcStr` has to be set to the new prefix of the fold change columns.

3.6.2 Specifying customized filtering criteria for normalization

You can set the filtering criteria for normalization set construction by modifying the supplied default settings. Remember to adjust the fold change column numbers in case you have more/ less than ten fold changes per experiment.

```
trNewReqs <- tpptrDefaultNormReqs()
print(trNewReqs)
## $fcRequirements
##   fcColumn thresholdLower thresholdUpper
## 1        7            0.4            0.6
## 2        9            0.0            0.3
## 3       10            0.0            0.2
##
## $otherRequirements
##   colName thresholdLower thresholdUpper
## 1    qssm              4            Inf
trNewReqs$otherRequirements[1,"colName"] <- "mycolName"
trNewReqs$fcRequirements[,"fcColumn"] <- c(6,8,9)
print(trNewReqs)
## $fcRequirements
##   fcColumn thresholdLower thresholdUpper
## 1        6            0.4            0.6
## 2        8            0.0            0.3
## 3        9            0.0            0.2
##
## $otherRequirements
##     colName thresholdLower thresholdUpper
## 1 mycolName              4            Inf
```

4 Analyzing TPP-CCR experiments

First, we load the data:

```
data("hdacCCR_smallExample")
```

This command loads two objects: the configuration tables for two replicates (hdacCCR_config_repl1/2) and two data frames that contain the measurements of both TPP-CCR experiments to be analyzed (hdacCCR_data_repl1/2).

4.1 Starting the whole workflow by analyzeTPPCCR

We start the workflow for replicate 1 by typing

```
CCRresults <- analyzeTPPCCR(configTable=hdacCCR_config_repl1, data=hdacCCR_data_repl1,
                            resultPath=resultPath, plotCurves=FALSE)
```

The following proteins passed the criteria of displaying a clear response to the treatment, and enabling curve fitting with $R^2 > 0.8$:

```
ccr_targets <- subset(CCRresults, passed_filter)$Protein_ID
print(ccr_targets)
```

```
##  [1] "ALKBH1" "CHMP5"  "ECH1"   "HDAC1"  "HDAC10" "HDAC2"  "HDAC6"  "HSPB11" "TTC38"
## [10] "ZNF384"
```

4 of the selected proteins belong to the HDAC family. Because Panobinostat is known to act as an HDAC inhibitor, we select them for further investigation.

```
hdac_targets <- grep("HDAC", ccr_targets, value=TRUE)
print(hdac_targets)
```

```
## [1] "HDAC1"  "HDAC10" "HDAC2"  "HDAC6"
```

The following section explains how to invoke the individual steps of the workflow separately.

4.2 Starting individual steps of the workflow

4.2.1 Data import

The function tppccrImport imports the data and converts it into an *ExpressionSet*:

```
ccrData <- tppccrImport(configTable=hdacCCR_config_repl1, data=hdacCCR_data_repl1)
```

```
## Importing data...
##
## Importing CCR dataset:  Panobinostat_1
## Removing duplicate identifiers using quality column 'qupm'...
## 507 out of 507 rows kept for further analysis.
##  -> Panobinostat_1 contains 507 proteins.
##  -> 494 out of 507 proteins (97.44%) suitable for curve fit (criterium:  > 2 valid fold changes
per protein).
## Removing proteins with zero values in column qssm:
## 494 out of 507 proteins remaining after filtering.
```

4.2.2 Data normalization

Currently, the *TPP* package offers normalization by fold change medians for TPP-CCR experiments. We normalize the imported data by

```
ccrDataNormalized <- tppccrNormalize(data=ccrData)
```

```
## Normalizing data ...
## done.
```

4.2.3 Data transformation

We next have to specify the type of response for each protein, and transform the data accordingly:

```
ccrDataTransformed <- tppccrTransform(data=ccrDataNormalized)
```

```
## Transforming data ...
## done.
```

4.2.4 Dose response curve fitting

Next we fit and plot dose response curves for the detected HDAC targets. We first select the corresponding rows from the imported data:

```
ccrDataHDAC <- ccrDataTransformed[match(hdac_targets, featureNames(ccrDataTransformed)),]
```

We fit dose response curves for these proteins using the function `tppccrCurveFit`:

```
ccrResultsHDAC <- tppccrCurveFit(data=ccrDataHDAC, resultPath=resultPath)
```

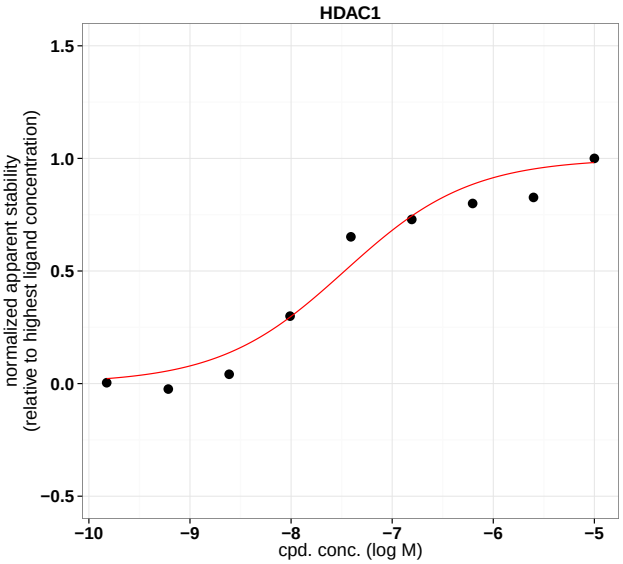
```
## Calculating pEC50s ...
## Fitting dose response curve for protein:  HDAC1
## Fitting dose response curve for protein:  HDAC10
## Fitting dose response curve for protein:  HDAC2
## Fitting dose response curve for protein:  HDAC6
## Runtime:  1.15 secs
##
## done.
```

This function produces a table that contains the dose response curve parameters and additional information about each protein:

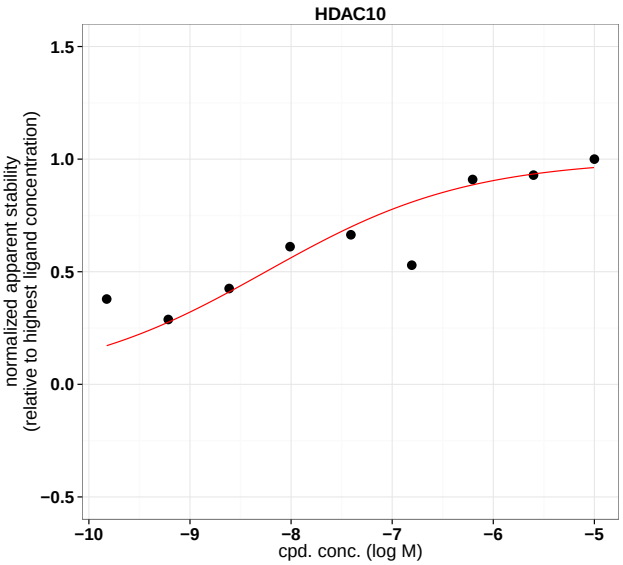
```
print(ccrResultsHDAC[,c(1, 12:15)])
```

```
##   Protein_ID pEC50 slope  R_sq passed_filter
## 1      HDAC1  7.47  1.61 0.963          TRUE
## 2     HDAC10  8.25  1.00 0.859          TRUE
## 3      HDAC2  7.87  1.88 0.968          TRUE
## 4      HDAC6  6.37  2.03 0.954          TRUE
```

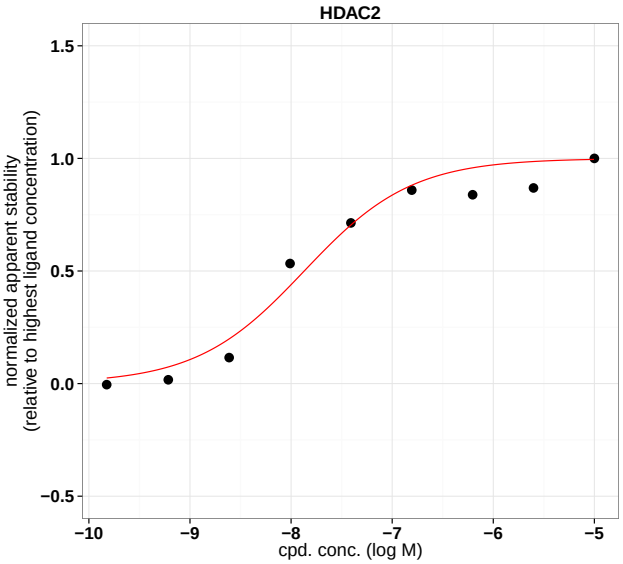
The dose response curve plots were stored in subdirectory `DoseResponse_Curves` in `resultPath`. You can browse this directory and inspect the fits and melting curve parameters. In the following, you can see the plot that were placed in this directory for the 4 detected targets:



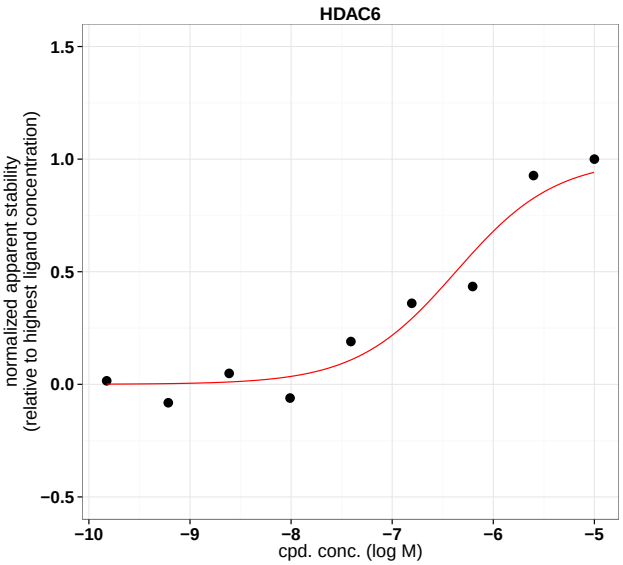
	pEC50	slope	R2
Panobinostat_1	7.5	1.6	0.96



	pEC50	slope	R2
Panobinostat_1	8.2	1	0.86



	pEC50	slope	R2
Panobinostat_1	7.9	1.9	0.97



	pEC50	slope	R2
Panobinostat_1	6.4	2	0.95

References

[1] Mikhail M Savitski, Friedrich BM Reinhard, Holger Franken, Thilo Werner, Maria Fälth Savitski, Dirk Eberhard, Daniel Martinez Molina, Rozbeh Jafari, Rebecca Bakszt Dovega, Susan Klaeger, et al. Tracking cancer drugs in living cells by thermal profiling of the proteome. *Science*, 346(6205):1255784, 2014.