Non-parametric analysis of thermal proteome profiles

Dorothee Childs, Nils Kurzawa

15 August 2018, 17:36:58

Contents

1 Introduction

This workflow shows how to reproduce the analysis described by Childs, Bach, Franken et al. (2018): Non-parametric analysis of thermal proteome profiles reveals novel drug-binding proteins.

2 Preparation

Load necessary packages:

```r
library(tidyverse)
library(broom)
library(knitr)
```

3 Data import

First we load the data from the different TPP experiments. All data have been downloaded from the supplements of the respective publications (Franken et al. 2015, Reinhard et al. (2015), Savitski et al. (2014)), converted into tidy format, and concatenated into one table. This table will be made available as supplementary material to the paper. Until then, it can be found in the same folder as this vignette.

```r
tppData <- readRDS("tppData.Rds")
```

Let's take a look at the first lines of the imported data:

```r
tppData %>% head %>% kable()
```

dataset	uniqueID	relAbundance	temperature	molarDrugConcentration	replicate	unique
ATP	12 KDA PROTEIN._IPI00879767	NA	25	0.002	1	
ATP	12 KDA PROTEIN._IPI00879767	NA	41	0.002	1	
ATP	12 KDA PROTEIN._IPI00879767	NA	44	0.002	1	
ATP	12 KDA PROTEIN._IPI00879767	NA	47	0.002	1	
ATP	12 KDA PROTEIN._IPI00879767	NA	50	0.002	1	
ATP	12 KDA PROTEIN._IPI00879767	NA	53	0.002	1	

And a data summary:

```r
tppData %>%
  mutate(molarDrugConcentration = factor(molarDrugConcentration),
         replicate = factor(replicate),
         dataset = factor(dataset)) %>%
  summary() %>%
  kable()
```

	dataset	uniqueID	relAbundance	temperature	molarDrugConcentration	replicate	unic
	ATP :268000	Length:1432280	Min. : 0.0	Min. :25.00	0 :716140	1:716140	Min
	Dasatinib 0.5:308520	Class :character	1st Qu.: 0.1	1st Qu.:44.00	5e-07:154260	2:716140	1st
	Dasatinib 5 :308520	Mode :character	Median : 0.6	Median :52.50	1e-06:120080	NA	Mec
	Panobinostat :240160	NA	Mean : 0.6	Mean :51.86	5e-06:154260	NA	Mea
	Staurosporine:307080	NA	3rd Qu.: 1.0	3rd Qu.:59.00	2e-05:153540	NA	3rd
	NA	NA	Max. :577.6	Max. :67.00	0.002:134000	NA	Max
	NA	NA	NA's :372809	NA	NA	NA	NA'

4 Data preprocessing

First, we remove all decoy proteins remaining in the panobinostat data. They can be recognized by the prefix `###`, which was assigned by the quantification software `isobarQuant`.

```
tppData <- tppData %>% filter(!grepl("###[[:alnum:]]*###", uniqueID))
```

Next, we remove all proteins that were not found with at least one unique peptide.

```
tppData <- filter(tppData, uniquePeptideMatches >= 1)
```

Next, we remove all proteins that only contain missing values.

```
tppData <- tppData %>% filter(!is.na(relAbundance))
```

Finally, we remove all proteins not reproducibly observed with full melting curves in both replicates and treatment groups per dataset. A full melting curve is defined by the presence of measurements at all 10 temperatures for the given experimental group.

```
tppData <- tppData %>%
  group_by(dataset, uniqueID) %>%
  mutate(n = n()) %>%
  group_by(dataset) %>%
  mutate(max_n = max(n)) %>%
  filter(n == max_n) %>%
  dplyr::select(-n, -max_n) %>%
  ungroup
```

4.1 Reproduce Table 1 of the paper

Count the numbers of proteins remaining in each dataset. They coincide with the values reported in Table 1.

```
tppData %>%
  distinct(dataset, uniqueID) %>%
  distinct %>%
  group_by(dataset) %>%
  tally %>%
  kable()
```

dataset	n
ATP	4177
Dasatinib 0.5	4625
Dasatinib 5	4154
Panobinostat	3649
Staurosporine	4505

5 Illustrative example

We first illustrate the principles of nonparametric analysis of response curves (NPARC) on an example protein (STK4) from the staurosporine dataset. The same protein is shown in Figures 1 and 2 of the paper.

5.1 Select data

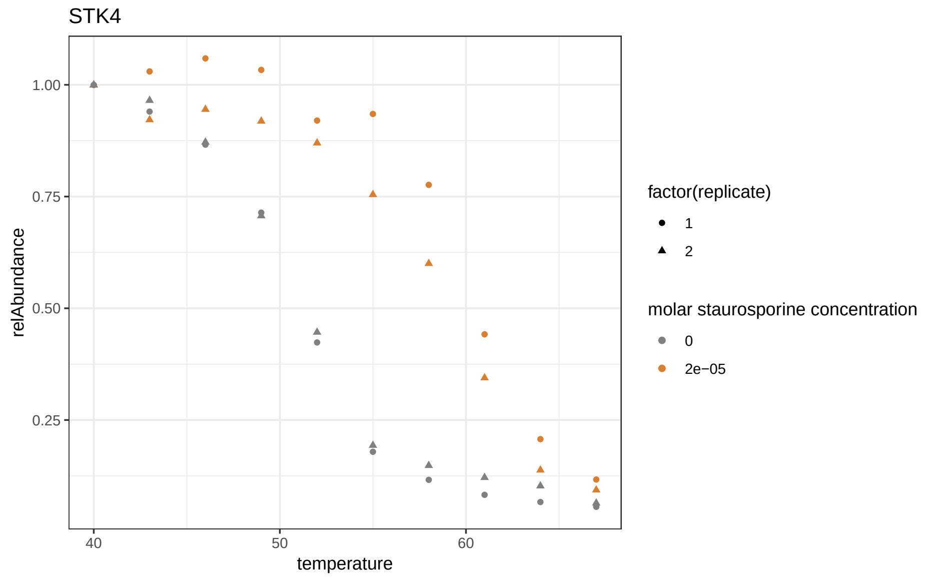
We first select all data entries belonging to the desired protein and dataset:

```
stk4 <- filter(tppData, dataset == "Staurosporine", grepl("STK4", uniqueID))
```

To obtain a first impression of the measurements in each experimental group, we generate a plot of the measurements:

```
stk4_plot <- ggplot(stk4, aes(x = temperature, y = relAbundance)) +
  geom_point(aes(shape = factor(replicate), color = factor(molarDrugConcentration))) +
  theme_bw() +
  ggtitle("STK4") +
  scale_color_manual("molar staurosporine concentration",
                     values = c("#808080", "#da7f2d"))

print(stk4_plot)
```



We will show how to add the fitted curves to this plot in the following steps.

5.2 Define function for model fitting

To assess whether there is a significant difference between both treatment groups, we will fit a null model and an alternative models to the data. The null model fits a sigmoid melting curve through all data points irrespective of experimental condition. The alternative model fits separate melting curves per experimental group (vehicle: 0 muM staurosporine, treatment: 20 muM staurosporine).

Because we have to repeat the fitting several times in this workflow, we define a function that we can call repeatedly:

```r
fitSingleSigmoid <- function(x, y, start=c(Pl = 0, a = 550, b = 10)){
  try(nls(formula= y ~ (1 - Pl)  / (1+exp((b - a/x))) + Pl,
          start=start,
          data=list(x=x, y=y),
          na.action = na.exclude,
          algorithm = "port",
          lower = c(0.0, 1e-5, 1e-5),
          upper = c(1.5, 15000, 250),
          control = nls.control(maxiter=50)),
      silent = TRUE)
}
```

5.3 Fit null models

Now, we can use the function defined in the previous Section to fit the null model. This function will return an `R` object of class `nls`. To obtain the predictions at each temperature in tabular format, we will use the function `augment` from the `broom` package. It also returns the corresponding residuals which we will need later for the hypothesis test. By appending the returned predictions and residuals to our data frame with the measurements for STK4, we ensure that relevant data is collected in the same table and can be added to the plot for visualization.

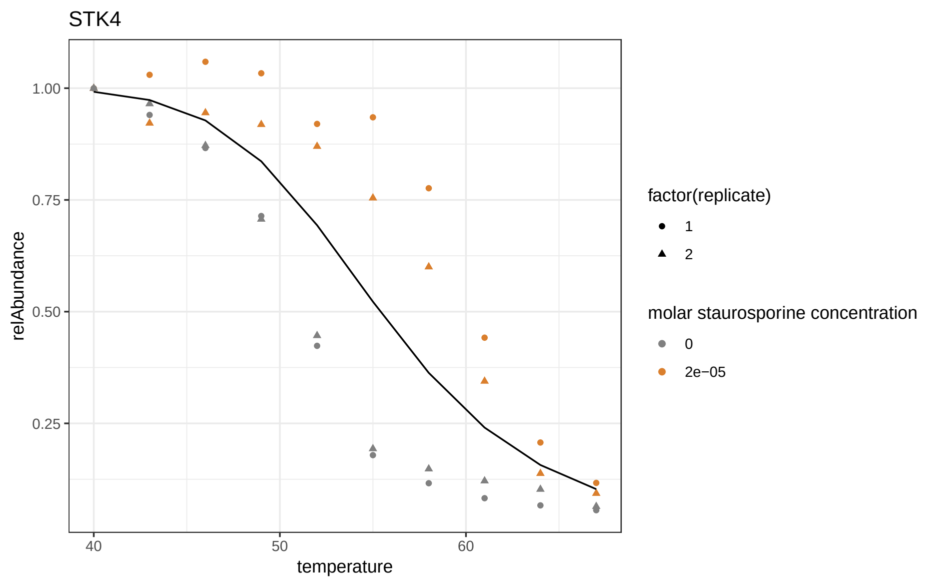
```r
nullFit <- fitSingleSigmoid(x = stk4$temperature, y = stk4$relAbundance)
nullPredictions <- broom::augment(nullFit)

stk4$nullPrediction <- nullPredictions$.fitted
stk4$nullResiduals <- nullPredictions$.resid
```

Plot the curve predicted by the null model:

```r
stk4_plot <- stk4_plot +
  geom_line(data = stk4, aes(y = nullPrediction))

print(stk4_plot)
```



5.4 Fit alternative models

Next we fit the alternative model. Again, we compute the predicted values and the corresponding residuals by the `broom` package.

```
alternativePredictions <- stk4 %>%
# Fit separate curves per treatment group:
  group_by(molarDrugConcentration) %>%
  do({
    fit = fitSingleSigmoid(x = .$temperature,
                           y = .$relAbundance)
    broom::augment(fit)
  }) %>%
  ungroup %>%
  # Rename columns for merge to data frame:
  dplyr::rename(alternativePrediction = .fitted,
              alternativeResiduals = .resid,
              temperature = x,
              relAbundance = y)
```

Add the predicted values and corresponding residuals to our data frame:

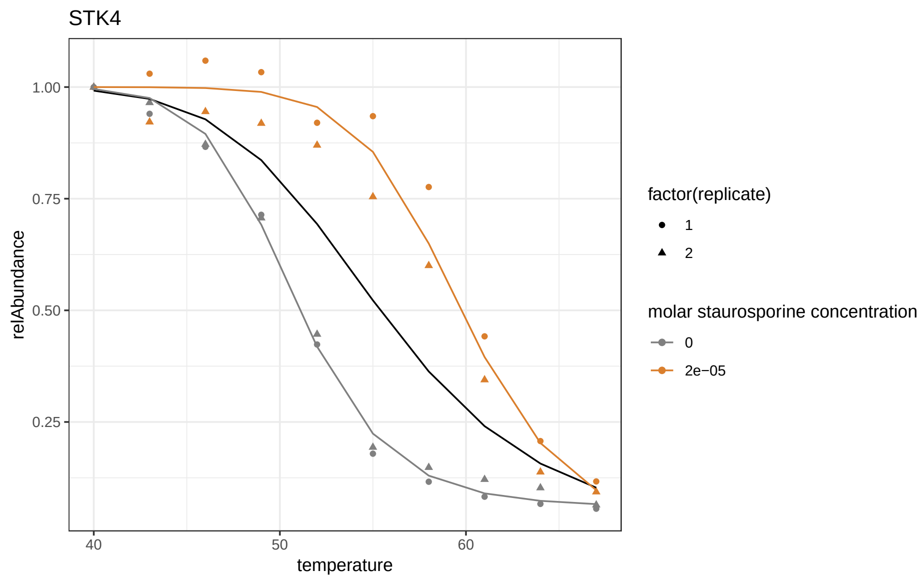
```
stk4 <- stk4 %>%
  left_join(alternativePredictions,
            by = c("relAbundance", "temperature", "molarDrugConcentration")) %>%
  distinct()
```

5.5 Reproduce Figure 2 (A)/(B) of the paper

Add the curves predicted by the alternative model to the plot:

```
stk4_plot <- stk4_plot +
  geom_line(data = distinct(stk4, temperature, molarDrugConcentration, alternativePrediction),
            aes(y = alternativePrediction, color = factor(molarDrugConcentration)))

print(stk4_plot)
```



This plot corresponds to Figures 2(A) and 2(B) in the paper.

5.6 Compute RSS values

In order to quantify the improvement in goodness-of-fit of the alternative model relative to the null model, we compute the sum of squared residuals (RSS).

```
rssPerModel <- stk4 %>%
  summarise(rssNull = sum(nullResiduals^2),
            rssAlternative = sum(alternativeResiduals^2))

kable(rssPerModel, digits = 4)
```

rssNull	rssAlternative
1.2181	0.0831

These values will be used to construct the F-statistic according to

$$\mathrm{F}_i = \frac{\mathrm{d}_{i2}}{\mathrm{d}_{i1}} \cdot \frac{\mathrm{RSS}_i^0 - \mathrm{RSS}_i^1}{\mathrm{RSS}_i^1}. \quad (1)$$

To compute this statistic and to derive a p-value, we need the degrees of freedom $\mathrm{d}_{i1}, \mathrm{d}_{i2}$. They cannot be analytically derived due to the correlated nature of the measurements. The paper describes how to estimate these values from the RSS-values of all proteins in the dataset. In the following Section, we illustrate how to repeat the model fitting for all proteins of a dataset and how to perform hypothesis testing on these models.

6 Extending the analysis to all proteins

In order to analyze all datasets as described in the paper, we fit null and alternative models to each protein in each dataset, as shown in the following.

Before starting the model fitting, we combine both dasatinib datasets into one dataset with four replicates of the vehicle experiments, and two replicates in each of two treatment groups. In one treatment group, dasatinib was administered with 0.5 muM concentration, and with 5 muM in the other group.

```
# Remove suffix from dataset names that distinguishes both dasatinib datasets
tppData <- tppData %>%
  mutate(replicate = ifelse(dataset == "Dasatinib 5",
                            yes = replicate + 2,
                            no = replicate)) %>%
  mutate(dataset = gsub(" 0.5| 5", "", dataset))

# Check result: List all dataset names and the administered drug concentrations
tppData %>%
  distinct(dataset, replicate, molarDrugConcentration) %>%
  filter(molarDrugConcentration > 0) %>%
  dplyr::rename(`drug concentration (treatment groups)` = molarDrugConcentration) %>%
  kable()
```

dataset	drug concentration (treatment groups)	replicate
ATP	2e-03	1
ATP	2e-03	2
Dasatinib	5e-07	1
Dasatinib	5e-07	2
Dasatinib	5e-06	3
Dasatinib	5e-06	4
Panobinostat	1e-06	1
Panobinostat	1e-06	2
Staurosporine	2e-05	1
Staurosporine	2e-05	2

6.1 Define functions

We fit the models by the same function as illustrated on the STK4 example above. In order to iterate over all proteins and experimental factors, we split the data by the `dplyr::group_by` function, and loop over all subsets by the `dplyr::do` function. For each model, we retrieve the residuals by the function `residuals()` and compute the sum of their squared values (RSS). Because we will need to re-use the code for model fitting and RSS computation, we encapsulate it into a function that we can re-use for the null and alternative model fits of each protein. It will also make debugging easier if the code lives within a separate function.

For a few proteins, the nonlinear least-squares optimization will not converge with the given start parameters. For some of these proteins, however, convergence can be obtained after adding a small random noise to the start paramters. To this purpose, we write a function that starts the optimization repeatedly with randomly perturbed start parameters for such proteins:

```
repeatFits <- function(x, y, seed = NULL, alwaysPermute = FALSE, maxAttempts = 100){

  start <- c(Pl = 0, a = 550, b = 10)
  i <- 0
  doFit <- TRUE
  doVaryPars <- alwaysPermute

  if (!is.null(seed)){
    set.seed(seed)
  }

  while (doFit){
    startTmp <- start * (1 + doVaryPars*runif(1, -0.5, 0.5))
    m <- fitSingleSigmoid(x = x, y = y, start = startTmp)
    i <- i + 1
    doFit <- inherits(m, "try-error") & i < maxAttempts
    doVaryPars <- TRUE
  }

  return(m)
}
```

```
computeRSS <- function(x, y, seed = NULL, alwaysPermute = FALSE, maxAttempts = 100){

  # Start model fitting
  fit <- repeatFits(x = x, y = y, seed = seed,
                    alwaysPermute = alwaysPermute,
                    maxAttempts = maxAttempts)

  if (!inherits(fit, "try-error")){
    # If model fit converged, obtain data frame containing predicted values and residuals
    resid <- residuals(fit)
    rss <- sum(resid^2, na.rm = TRUE)
    nResid <- sum(!is.na(resid))
    isConverged <- TRUE
  } else {
    # If model fit did not converge, return default values
    rss <- NA
    nResid <- NA
    isConverged <- FALSE
  }

  return(data.frame(rss = rss, nResid = nResid, isConverged = isConverged))
}
```

6.2 Fit null models

Now we can fit the null models to each protein in each dataset:

```
nullRSS <- tppData %>% #filter(dataset == "Panobinostat") %>% #filter(uniqueID == "ABHD10_NA", dataset == "Pa
  # Iterate over all proteins per dataset:
  group_by(dataset, uniqueID) %>%
  do(
    # Invoke model fitting and RSS computation for the current protein:
    computeRSS(x = .$temperature, y = .$relAbundance, seed = 123)
    ) %>%
  ungroup
```

Show a data summary:

```
nullRSS %>%
  mutate(dataset = factor(dataset), nResid = factor(nResid)) %>%
  summary()
##          dataset       uniqueID               rss              nResid
##  ATP          :4177   Length:17120       Min.   :    0.005   38   :  162
##  Dasatinib    :4789   Class :character   1st Qu.:    0.077   40   :12952
##  Panobinostat :3649   Mode  :character   Median :    0.159   78   : 3976
##  Staurosporine:4505                      Mean   :    4.586   NA's:   30
##                                          3rd Qu.:    0.390
##                                          Max.   :16290.763
##                                          NA's   :30
##  isConverged
##  Mode :logical
##  FALSE:30
##  TRUE :17090
##
##
##
##
```

6.3 Fit alternative models

Next we fit the alternative models:

```
alternativeRSS <- tppData %>% #filter(dataset == "Panobinostat") %>%
  group_by(dataset, uniqueID, molarDrugConcentration) %>%
  do(
    computeRSS(x = .$temperature, y = .$relAbundance, seed = 123)
    ) %>%
  ungroup
```

Show a data summary:

```
alternativeRSS %>%
  mutate(dataset = factor(dataset), nResid = factor(nResid)) %>%
  summary()
##          dataset        uniqueID          molarDrugConcentration
##  ATP          : 8354   Length:38230       Min.   :0.0000000
##  Dasatinib    :13568   Class :character   1st Qu.:0.0000000
##  Panobinostat : 7298   Mode  :character   Median :0.0000005
##  Staurosporine: 9010                      Mean   :0.0002216
##                                           3rd Qu.:0.0000050
##                                           Max.   :0.0020000
##
##       rss              nResid      isConverged
##  Min.   :    0.001   19   : 4295   Mode :logical
##  1st Qu.:    0.026   20   :29868   FALSE:92
##  Median :    0.056   39   : 3975   TRUE :38138
##  Mean   :    2.101   NA's:   92
##  3rd Qu.:    0.143
##  Max.   :14939.823
##  NA's   :92
```

Summarize RSS over all treatment groups:

```
alternativeRSSCollated <- alternativeRSS %>%
  group_by(dataset) %>%
  mutate(maxGroups = length(unique(molarDrugConcentration))) %>%
  group_by(dataset, uniqueID, maxGroups) %>%
  summarise(rss = sum(rss, na.rm = FALSE),
            nResid = sum(nResid, na.rm = FALSE),
            groupsConverged = sum(isConverged)) %>%
  mutate(allConverged = groupsConverged == maxGroups) %>%
  ungroup
```

Count numbers of fitted values occuring per dataset:

```
alternativeRSSCollated %>%
  group_by(dataset, nResid, allConverged, maxGroups) %>%
  tally() %>%
  kable()
```

dataset	nResid	allConverged	maxGroups	n
ATP	40	TRUE	2	4165
ATP	NA	FALSE	2	12
Dasatinib	38	FALSE	3	162
Dasatinib	40	FALSE	3	630
Dasatinib	78	TRUE	3	3950
Dasatinib	NA	FALSE	3	47
Panobinostat	40	TRUE	2	3630
Panobinostat	NA	FALSE	2	19
Staurosporine	40	TRUE	2	4495
Staurosporine	NA	FALSE	2	10

6.4 Combine results from both model fits

Combine the RSS values of all proteins for which the models converged in all groups:

```
# Make unique columns for merge to a common data frame:
dat1 <- nullRSS %>% dplyr::rename(rss0 = rss, n0 = nResid)
dat2 <- alternativeRSSCollated %>% dplyr::rename(rss1 = rss, n1 = nResid)

# Perform merge of both tables:
allRSS <- full_join(dat1, dat2, by = c("dataset", "uniqueID")) %>%
  filter(isConverged, allConverged)
```

In order to quantify the improvement in goodness-of-fit of the alternative model relative to the null model, we compute the difference in RSS between both models for each protein. Proteins for which RSS^1 is not smaller than RSS^0 are excluded from p-value calculation.

```
allRSS <- allRSS %>%
  mutate(rssDiff = rss0 - rss1) %>%
  mutate(rssDiff = ifelse(rssDiff < 0, NA, rssDiff))
```

Let us look at the columns of the data frame `allRSS`:

```
allRSS %>%
  mutate(dataset = factor(dataset), n0 = factor(n0), n1 = factor(n1), maxGroups = factor(maxGroups)) %>%
  summary()
##           dataset       uniqueID              rss0             n0
##  ATP          :4161   Length:16224       Min.   :    0.005   40:12281
##  Dasatinib    :3943   Class :character   1st Qu.:    0.075   78: 3943
##  Panobinostat :3626   Mode  :character   Median :    0.153
##  Staurosporine:4494                      Mean   :    4.228
##                                          3rd Qu.:    0.365
##                                          Max.   :16290.763
##
##  isConverged    maxGroups      rss1             n1        groupsConverged
##  Mode:logical   2:12281   Min.   :    0.004   40:12281   Min.   :2.000
##  TRUE:16224     3: 3943   1st Qu.:    0.062   78: 3943   1st Qu.:2.000
##                           Median :    0.126              Median :2.000
##                           Mean   :    4.172              Mean   :2.243
##                           3rd Qu.:    0.312              3rd Qu.:2.000
##                           Max.   :16290.423              Max.   :3.000
##
##  allConverged      rssDiff
##  Mode:logical   Min.   : 0.00000
##  TRUE:16224     1st Qu.: 0.00486
##                 Median : 0.01438
##                 Mean   : 0.06552
##                 3rd Qu.: 0.04081
##                 Max.   :33.97808
##                 NA's   :51
```

For how many proteins could we compute valid RSS differences per dataset?

```
allRSS %>%
  group_by(dataset) %>%
  summarize(n = sum(!is.na(rssDiff))) %>%
  kable()
```

dataset	n
ATP	4161
Dasatinib	3901
Panobinostat	3619
Staurosporine	4492

6.5 Compute test statistics

6.5.1 Why we need to estimate the degrees of freedom

In order to compute F-statistics per protein and dataset according to Equation (1), we need to know the degrees of freedom of the corresponding null distribution. If we could assume independent and identically distributed (iid) residuals, we could compute them from the number of fitted values and model parameters. In the following, we will show why this simple equation is not appropriate for the curve data we are working with.

First, we compute the degrees of freedom that we would assume for iid residuals:

```
DOF <- allRSS %>%
  mutate(paramsNull = 3,
         paramsAlternative = ifelse(n1 > 40, yes = 9, no = 6)) %>%
  mutate(DOF1 = paramsAlternative - paramsNull,
         DOF2 = n1 - paramsAlternative)
```

Let us take a look at the computed degrees of freedom:

```
DOF %>%
  distinct(dataset, n0, n1, converged0, allConverged1, paramsNull, paramsAlternative, DOF1, DOF2) %>%
  kable()
## Warning: Trying to compute distinct() for variables not found in the data:
## - `converged0`, `allConverged1`
## This is an error, but only a warning is raised for compatibility reasons.
## The following variables will be used:
## - dataset
## - n0
## - n1
## - paramsNull
## - paramsAlternative
## - (2 more)
```

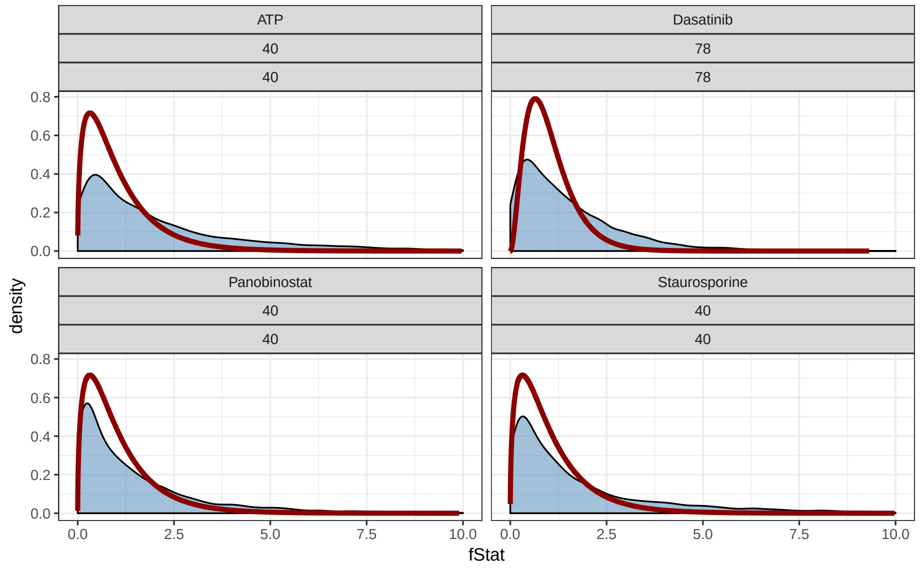
dataset	n0	n1	paramsNull	paramsAlternative	DOF1	DOF2
ATP	40	40	3	6	3	34
Dasatinib	78	78	3	9	6	69
Panobinostat	40	40	3	6	3	34
Staurosporine	40	40	3	6	3	34

No we calculate the F-statistics per protein and compare them to the corresponding F-distribution to derive p-values:

```
testResults <- DOF %>%
  mutate(fStat = (rssDiff/DOF1) / (rss1/DOF2),
         pVal = 1 - pf(fStat, df1 = DOF1, df2 = DOF2),
         pAdj = p.adjust(pVal, "BH"))
```

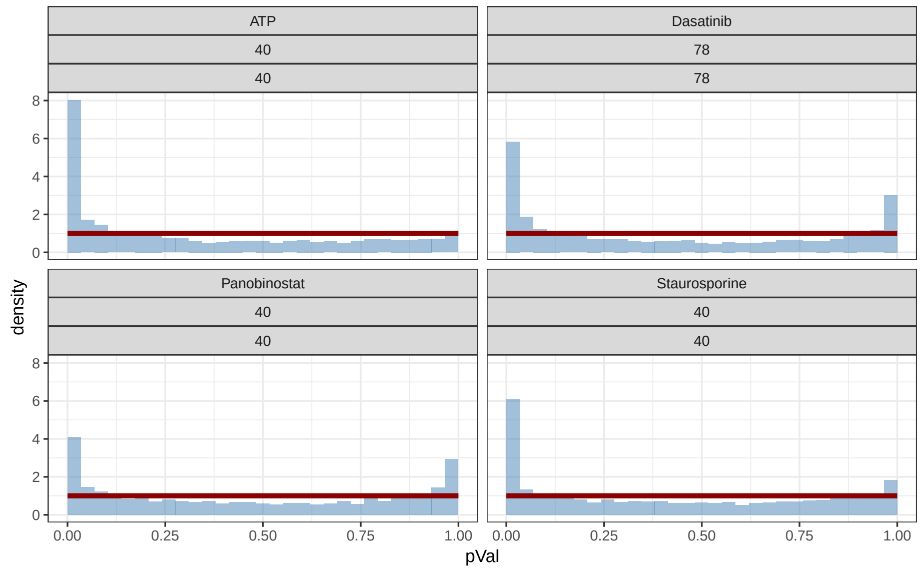
We plot the F-statistics against the theoretical F-distribution to check how well the null distribution is approximated now:

```
ggplot(testResults) +
  geom_density(aes(x = fStat), fill = "steelblue", alpha = 0.5) +
  geom_line(aes(x = fStat, y = df(fStat, df1 = DOF1, df2 = DOF2)), color = "darkred", size = 1.5) +
  facet_wrap(~ dataset + n0 + n1) +
  theme_bw() +
  # Zoom in to small values to increase resolution for the proteins under H0:
  xlim(c(0, 10))
## Warning: Removed 648 rows containing non-finite values (stat_density).
## Warning: Removed 176 rows containing missing values (geom_path).
```



The densities of the theoretical F-distribution (red) do not fit the observed values (blue) very well. While the theoretical distribution tends to overestimate the number of proteins with test statistics smaller than 2.5, it appears to underestimate the amount of proteins with larger values. This would imply that even for highly specific drugs, we observe many more significant differences than we would expect by chance. This hints at an anti-conservative behaviour of our test with the calculated degree of freedom parameters. This is reflected in the p-value distributions. If the distribution assumptions were met, we would expect the null cases to follow a uniform distribution, with a peak on the left for the non-null cases. Instead, we observe a tendency to obtain fewer values than expected in the middle range (around 0.5), but distinct peaks to the left.

```
ggplot(testResults) +
  geom_histogram(aes(x = pVal, y = ..density..), fill = "steelblue", alpha = 0.5, boundary = 0) +
  geom_line(aes(x = pVal, y = dunif(pVal)), color = "darkred", size = 1.5) +
  facet_wrap(~ dataset + n0 + n1) +
  theme_bw()
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 51 rows containing non-finite values (stat_bin).
## Warning: Removed 2 rows containing missing values (geom_path).
```



6.5.2 How to estimate the degrees of freedom

In the paper, we describe an alternative way to estimate the degrees of freedom by fitting χ^2 distributions to the numerator and denominator across all proteins in a dataset. To enable fitting of the distributions, we first need to re-scale the variables by a scaling factor. Because the scaling factors are characteristic for each dataset (it depends on the variances of the residuals in the respective dataset), we estimate them from the data according to:

$$\sigma_0^2 = \frac{1}{2}\frac{V}{M}, \qquad (2)$$

where V is the variance of the distribution, and M is the mean of the distribution.

We estimate V and M from the empirical distributions of the RSS differences $(\mathrm{RSS}^1 - \mathrm{RSS}^0)$. To increase robustness, we estimate M and V by their D-estimates Marazzi (2002) (median and median absolute deviation).

```
scalingFactors <- allRSS %>%
  filter(!is.na(rssDiff)) %>%
  group_by(dataset) %>%
  summarise(M = median(rssDiff, na.rm = T), V = mad(rssDiff, na.rm = T)^2) %>%
  ungroup %>%
  mutate(s0_sq = 1/2 * V/M)

scalingFactors %>% kable(digits = 10)
```

dataset	M	V	s0_sq
ATP	0.012221151	0.0002103447	0.008605764
Dasatinib	0.032566774	0.0011668280	0.017914393
Panobinostat	0.009891643	0.0001495811	0.007560985
Staurosporine	0.009655322	0.0001297794	0.006720615

We scale the numerator and denominator in Equation (1) by these scaling factors and estimate the degree of freedom parameters by fitting unscaled χ^2 distributions.

First we add the scaling factors to the filtered RSS data as a separate column:

```
rssScaled <- scalingFactors %>%
  dplyr::select(dataset, s0_sq) %>%
  left_join(allRSS, by = "dataset") %>%
  mutate(rssDiff = rssDiff/s0_sq,
         rss1 = rss1/s0_sq)
```

Then we fit the degrees of freedom parameters numerically. This estimation proves to be fairly robust regarding the choice of the initial values, so we choose a small value of 1 for each optimization.

```
newDOF <- rssScaled %>%
  filter(!is.na(rssDiff)) %>%
  group_by(dataset) %>%
  do(
    data.frame(
      DOF1 = MASS::fitdistr(x = .$rssDiff, densfun = "chi-squared", start = list(df = 1))[["estimate"]],
      DOF2 = MASS::fitdistr(x = .$rss1, densfun = "chi-squared", start = list(df = 1))[["estimate"]]
    ))

newDOF %>%
  kable(digits = 10)
```

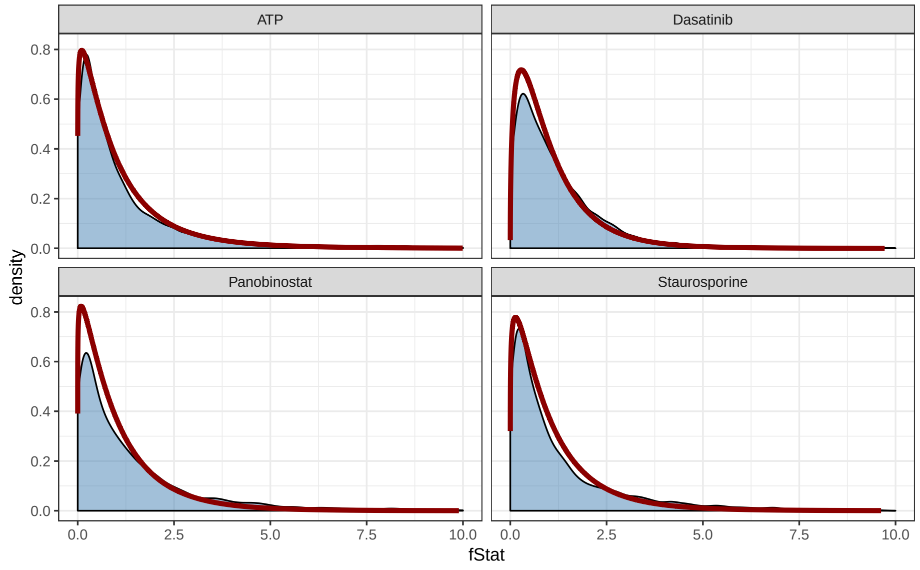
dataset	DOF1	DOF2
ATP	2.278906	11.89375
Dasatinib	2.885937	25.20000
Panobinostat	2.211719	22.32500
Staurosporine	2.355469	17.66875

Finally, we can compute the test statistics according to Equation (1) and compare them to the F-distribution:

```
newFStatistics <- newDOF %>%
  left_join(rssScaled, by = "dataset") %>%
  mutate(fStat = (rssDiff/DOF1) / (rss1/DOF2),
         pVal = 1 - pf(fStat, df1 = DOF1, df2 = DOF2),
         pAdj = p.adjust(pVal, "BH"))
```

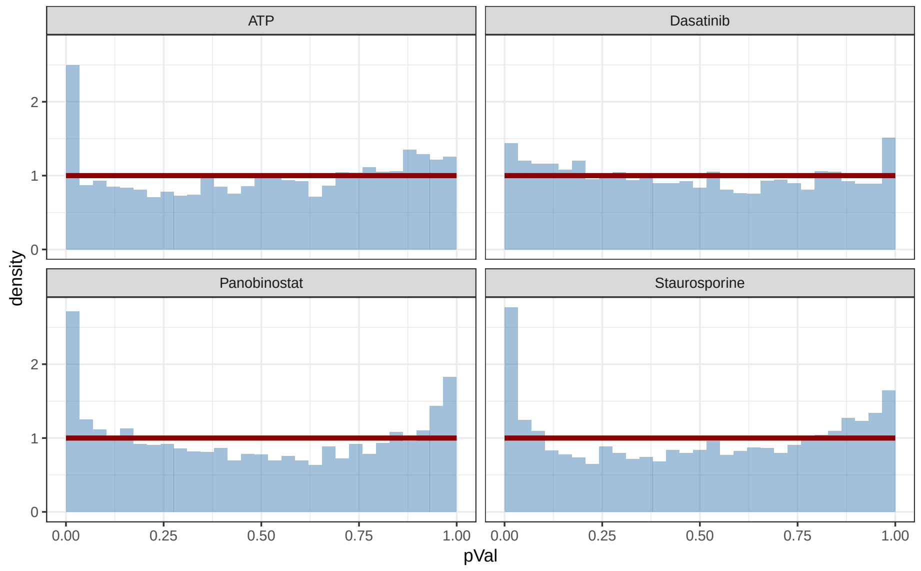
We plot the F-statistics against the theoretical F-distribution to check how well the null distribution is approximated now:

```
ggplot(newFStatistics) +
  geom_density(aes(x = fStat), fill = "steelblue", alpha = 0.5) +
  geom_line(aes(x = fStat, y = df(fStat, df1 = DOF1, df2 = DOF2)), color = "darkred", size = 1.5) +
  facet_wrap(~ dataset) +
  theme_bw() +
  # Zoom in to small values to increase resolution for the proteins under H0:
  xlim(c(0, 10))
## Warning: Removed 358 rows containing non-finite values (stat_density).
## Warning: Removed 110 rows containing missing values (geom_path).
```



Also check the p-value histograms. We expect the null cases to follow a uniform distribution, with a peak on the left for the non-null cases:

```
ggplot(newFStatistics) +
  geom_histogram(aes(x = pVal, y = ..density..), fill = "steelblue", alpha = 0.5, boundary = 0) +
  geom_line(aes(x = pVal, y = dunif(pVal)), color = "darkred", size = 1.5) +
  facet_wrap(~ dataset) +
  theme_bw()
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## Warning: Removed 51 rows containing non-finite values (stat_bin).
## Warning: Removed 2 rows containing missing values (geom_path).
```



The new F-statistics and p-values fit the expected distributions substantially better than before.

6.6 Detect significantly shifted proteins

Finally, we can select proteins that are significantly shifted by putting a threshold on the Benjamini-Hochberg corrected p-values.

```
topHits <- newFStatistics %>%
  filter(pAdj <= 0.01)
```

How many proteins were found per dataset?

```
topHits %>%
  group_by(dataset) %>%
  summarise(hits = n()) %>%
  kable()
```

dataset	hits
ATP	69
Dasatinib	6
Panobinostat	15
Staurosporine	80

Let us look at the targets detected in each dataset. These are the same proteins whose melting curves are shown in Fig. S3, S4, S6, and S7.

```
topHits %>%
  dplyr::select(dataset, uniqueID, fStat, pVal, pAdj) %>%
  arrange(-fStat) %>%
  split(., .$dataset) %>%
  lapply(., kable)
```

$ATP

dataset	uniqueID	fStat	pVal	pAdj
ATP	NARS_IPI00306960	73.88431	0.0000001	0.0002186
ATP	PRKCQ_IPI00029196	71.73357	0.0000002	0.0002186
ATP	ABCF1_IPI00873899	71.46904	0.0000002	0.0002186
ATP	RPS6KB1_IPI00216132	54.88794	0.0000007	0.0006361
ATP	MARK3_IPI00183118	51.45883	0.0000009	0.0006361
ATP	EHD2_IPI00100980	51.23744	0.0000010	0.0006361
ATP	RFK_IPI00099995	50.32031	0.0000011	0.0006361
ATP	PRKCA_IPI00385449	46.12280	0.0000017	0.0008873
ATP	RIOK1_IPI00171336	44.44750	0.0000021	0.0009603
ATP	KIF3A_IPI00867739	42.67870	0.0000026	0.0010526
ATP	MCM5_IPI00018350	42.05890	0.0000028	0.0010526
ATP	VPS4A_IPI00411356	40.15753	0.0000035	0.0012309
ATP	ABCF2_IPI00068506	37.88793	0.0000048	0.0015404
ATP	EHD4_IPI00005578	35.99389	0.0000063	0.0018671
ATP	SYK_IPI00018597	34.83491	0.0000074	0.0020637
ATP	DARS2_IPI00100460	34.13168	0.0000083	0.0020790
ATP	HARS_IPI00021808	33.94944	0.0000085	0.0020790
ATP	HSPA9_IPI00007765	33.42727	0.0000092	0.0021261
ATP	RFC2_IPI00017412	32.65686	0.0000104	0.0022213
ATP	RIOK2_IPI00306406	32.46723	0.0000107	0.0022213
ATP	MAP2K5_IPI00158248	31.79940	0.0000119	0.0023520
ATP	GALK1_IPI00019383	30.45855	0.0000148	0.0027939
ATP	HARS2_IPI00027445	29.53588	0.0000173	0.0030230
ATP	RECQL_IPI001784312	29.47413	0.0000174	0.0030230
ATP	RG9MTD1_IPI00099996	28.68023	0.0000200	0.0033288
ATP	EHD1_IPI00017184	27.93468	0.0000228	0.0036516
ATP	MARK2_IPI00555838	27.67958	0.0000239	0.0036810
ATP	MAP2K3_IPI00218858	27.36263	0.0000253	0.0037405
ATP	NSUN2_IPI00306369	27.19723	0.0000261	0.0037405
ATP	RFC5_IPI00031514	26.69304	0.0000286	0.0038822
ATP	DDX19A_IPI00008943	26.48886	0.0000297	0.0038822
ATP	RFC4_IPI00017381	26.43355	0.0000300	0.0038822
ATP	RPS6KB2_IPI00217069	26.29950	0.0000308	0.0038822
ATP	IARS2_IPI00017283	25.59923	0.0000352	0.0043044
ATP	MAP2K4_IPI00024674	25.11405	0.0000386	0.0043980
ATP	RIOK3_IPI00298199	25.08341	0.0000389	0.0043980
ATP	NEK7_IPI00152658	24.96306	0.0000398	0.0043980
ATP	PRKAA1_IPI00410287	24.68849	0.0000420	0.0043980
ATP	MCM7_IPI00299904	24.62801	0.0000425	0.0043980
ATP	STK25_IPI00893500	24.56923	0.0000430	0.0043980
ATP	GAPDH_IPI00788737	24.52063	0.0000434	0.0043980
ATP	PFKFB2_IPI00305589	24.41119	0.0000444	0.0043980
ATP	TRIP13_IPI00003505	24.11473	0.0000471	0.0045593
ATP	MCM4_IPI00018349	23.60989	0.0000522	0.0046796
ATP	MYO6_IPI00816452	23.60924	0.0000522	0.0046796
ATP	ACSM3_IPI00297635	23.60327	0.0000523	0.0046796
ATP	MAP2K1_IPI00219604	23.55062	0.0000529	0.0046796
ATP	FER_IPI00029263	23.28989	0.0000558	0.0048359
ATP	PRKCD_IPI00329236	23.15240	0.0000574	0.0048747
ATP	VPS4B_IPI00182728	22.11836	0.0000715	0.0059516
ATP	NUBPL_IPI00384517	21.90644	0.0000749	0.0061100
ATP	IDH1_IPI00027223	21.79555	0.0000767	0.0061396
ATP	CCNB1_IPI00745793	21.62588	0.0000796	0.0062526
ATP	DSTYK_IPI00465346	21.50325	0.0000818	0.0063054
ATP	MCM3_IPI00013214	21.28267	0.0000859	0.0065020
ATP	KIF2C_IPI00290435	21.19502	0.0000876	0.0065124

$Dasatinib

dataset	uniqueID	fStat	pVal	pAdj
Dasatinib	CRKL_IPI00004839	84.00830	0.0e+00	0.0000000
Dasatinib	YES1_IPI00013981	46.29650	0.0e+00	0.0000005
Dasatinib	MAPK14_IPI00221141	40.19671	0.0e+00	0.0000016
Dasatinib	BTK_IPI00029132	22.45262	3.0e-07	0.0003285
Dasatinib	AKAP9_IPI00220628	16.45990	4.8e-06	0.0037208
Dasatinib	GAB2_IPI00749276	15.28195	8.6e-06	0.0056102

$Panobinostat

dataset	uniqueID	fStat	pVal	pAdj
Panobinostat	HDAC1_NA	86.89568	0.00e+00	0.0000001
Panobinostat	TTC38_NA	78.75205	0.00e+00	0.0000001
Panobinostat	HDAC6_NA	65.80324	0.00e+00	0.0000003
Panobinostat	HDAC2_NA	45.77831	0.00e+00	0.0000061
Panobinostat	H2AFV|H2AFZ_NA	38.27073	0.00e+00	0.0000245
Panobinostat	ZFYVE28_NA	27.15088	6.00e-07	0.0003749
Panobinostat	HDAC8_NA	23.28485	2.10e-06	0.0010877
Panobinostat	HDAC10_NA	18.86740	1.02e-05	0.0046122
Panobinostat	GNB1L_NA	17.71987	1.60e-05	0.0059064
Panobinostat	SMTN_NA	17.66667	1.63e-05	0.0059064
Panobinostat	C5orf51_NA	17.15533	2.01e-05	0.0066012
Panobinostat	WDR26_NA	16.38570	2.76e-05	0.0078195
Panobinostat	NUP93_NA	16.34231	2.81e-05	0.0078195
Panobinostat	GTF2B_NA	15.95300	3.31e-05	0.0085613
Panobinostat	RNASEH2C_NA	15.54337	3.95e-05	0.0095282

$Staurosporine

dataset	uniqueID	fStat	pVal	pAdj
Staurosporine	CDK5_IPI00023530	369.70353	0.0000000	0.0000000
Staurosporine	MAP2K2_IPI00003783	148.48125	0.0000000	0.0000000
Staurosporine	CSK_IPI00013212	138.66150	0.0000000	0.0000000
Staurosporine	PMPCA_IPI00166749	137.15524	0.0000000	0.0000000
Staurosporine	AURKA_IPI00298940	131.15558	0.0000000	0.0000000
Staurosporine	FECH_IPI00554589	128.64839	0.0000000	0.0000000
Staurosporine	IRAK4_IPI00007641	122.25268	0.0000000	0.0000000
Staurosporine	CAMKK2_IPI00290239	116.68423	0.0000000	0.0000000
Staurosporine	PAK4_IPI00014068	113.34324	0.0000000	0.0000000
Staurosporine	STK4_IPI00011488	102.39278	0.0000000	0.0000000
Staurosporine	STK38_IPI00027251	99.69296	0.0000000	0.0000000
Staurosporine	PDPK1_IPI00002538	96.91349	0.0000000	0.0000000
Staurosporine	GSK3B_IPI00216190	93.03510	0.0000000	0.0000001
Staurosporine	ADRBK1_IPI00012497	82.42851	0.0000000	0.0000002
Staurosporine	BMP2K_IPI00337426	74.64797	0.0000000	0.0000003
Staurosporine	FER_IPI00029263	70.40135	0.0000000	0.0000005
Staurosporine	MAP2K1_IPI00219604	62.25164	0.0000000	0.0000012
Staurosporine	MAP2K7_IPI00302112	59.76483	0.0000000	0.0000015
Staurosporine	MAP4K2_IPI00149094	57.38529	0.0000000	0.0000020
Staurosporine	MAPK12_IPI00296283	55.76645	0.0000000	0.0000023
Staurosporine	PRKCE_IPI00024539	54.55703	0.0000000	0.0000025
Staurosporine	ADK_IPI00290279	54.49588	0.0000000	0.0000025
Staurosporine	STK3_IPI00411984	53.30688	0.0000000	0.0000029
Staurosporine	MAPK8_IPI00220306	50.13069	0.0000000	0.0000042
Staurosporine	MAPKAPK5_IPI00160672	50.11063	0.0000000	0.0000042
Staurosporine	PKN1_IPI00412672	50.01186	0.0000000	0.0000042
Staurosporine	PTK2_IPI00413961	49.33391	0.0000000	0.0000043
Staurosporine	AAK1_IPI00916402	49.32396	0.0000000	0.0000043
Staurosporine	CHEK2_IPI00423156	46.53685	0.0000000	0.0000065
Staurosporine	CDK2_IPI00031681	45.28064	0.0000001	0.0000077
Staurosporine	MAP2K4_IPI00024674	44.27708	0.0000001	0.0000086
Staurosporine	CPOX_IPI00093057	44.23314	0.0000001	0.0000086
Staurosporine	PHKG2_IPI00012891	43.01355	0.0000001	0.0000103
Staurosporine	TNIK_IPI00514275	42.43947	0.0000001	0.0000111
Staurosporine	VRK1_IPI00019640	41.36259	0.0000001	0.0000130
Staurosporine	MAPKAPK2_IPI00026054	38.00061	0.0000002	0.0000237
Staurosporine	MARK2_IPI00555838	37.78614	0.0000002	0.0000240
Staurosporine	CAMK2G_IPI00908444	36.17242	0.0000003	0.0000322
Staurosporine	GSK3A_IPI00292228	36.03375	0.0000003	0.0000323
Staurosporine	PRKAR2B_IPI00554752	35.79923	0.0000003	0.0000330
Staurosporine	RIOK2_IPI00306406	35.27343	0.0000003	0.0000358
Staurosporine	PRKACA_IPI00396630	33.88421	0.0000004	0.0000461
Staurosporine	RPS6KA3_IPI00020898	33.84602	0.0000004	0.0000461
Staurosporine	MAPK3_IPI00018195	33.27453	0.0000005	0.0000509
Staurosporine	STK24_IPI00872754	32.20468	0.0000006	0.0000629
Staurosporine	MARK3_IPI00183118	30.04718	0.0000010	0.0001006
Staurosporine	TTK_IPI00151170	29.46182	0.0000012	0.0001130
Staurosporine	MKNK1_IPI00304048	26.02004	0.0000028	0.0002613
Staurosporine	PDCD10_IPI00298558	25.80161	0.0000030	0.0002711
Staurosporine	OSBPL3_IPI00023555	25.52905	0.0000032	0.0002856
Staurosporine	HEBP1_IPI00148063	25.11716	0.0000035	0.0003126
Staurosporine	RPS6KA1_IPI00477982	24.34095	0.0000044	0.0003789
Staurosporine	PIK3CD_IPI00384817	24.06929	0.0000047	0.0004008
Staurosporine	MAP3K2_IPI00513803	23.84959	0.0000050	0.0004183
Staurosporine	SGK3_IPI00655852	23.33736	0.0000058	0.0004746
Staurosporine	PRKCB_IPI00219628	23.00379	0.0000064	0.0005129

7 Compare to the Tm-based approach

8 Session info

```
devtools::session_info()
## Session info ---------------------------------------------------------------
##  setting  value
##  version  R version 3.5.1 (2018-07-02)
##  system   x86_64, darwin15.6.0
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  tz       Europe/Berlin
##  date     2018-08-15
## Packages -------------------------------------------------------------------
##  package    * version date       source
##  assertthat   0.2.0   2017-04-11 CRAN (R 3.5.0)
##  backports    1.1.2   2017-12-13 CRAN (R 3.5.0)
##  base       * 3.5.1   2018-07-05 local
##  bindr        0.1.1   2018-03-13 CRAN (R 3.5.0)
##  bindrcpp   * 0.2.2   2018-03-29 CRAN (R 3.5.0)
##  BiocStyle  * 2.9.3   2018-06-13 Bioconductor
##  bookdown     0.7     2018-02-18 CRAN (R 3.5.0)
##  broom      * 0.5.0   2018-07-17 cran (@0.5.0)
##  cellranger   1.1.0   2016-07-27 CRAN (R 3.5.0)
##  cli          1.0.0   2017-11-05 CRAN (R 3.5.0)
##  colorspace   1.3-2   2016-12-14 CRAN (R 3.5.0)
##  compiler     3.5.1   2018-07-05 local
##  crayon       1.3.4   2017-09-16 CRAN (R 3.5.0)
##  datasets   * 3.5.1   2018-07-05 local
##  devtools     1.13.6  2018-06-27 CRAN (R 3.5.0)
##  digest       0.6.15  2018-01-28 CRAN (R 3.5.0)
##  dplyr      * 0.7.6   2018-06-29 CRAN (R 3.5.1)
##  evaluate     0.11    2018-07-17 cran (@0.11)
##  forcats    * 0.3.0   2018-02-19 CRAN (R 3.5.0)
##  ggplot2    * 3.0.0   2018-07-03 CRAN (R 3.5.0)
##  glue         1.3.0   2018-07-17 cran (@1.3.0)
##  graphics   * 3.5.1   2018-07-05 local
##  grDevices  * 3.5.1   2018-07-05 local
##  grid         3.5.1   2018-07-05 local
##  gtable       0.2.0   2016-02-26 CRAN (R 3.5.0)
##  haven        1.1.2   2018-06-27 CRAN (R 3.5.0)
##  hms          0.4.2   2018-03-10 CRAN (R 3.5.0)
##  htmltools    0.3.6   2017-04-28 CRAN (R 3.5.0)
##  httr         1.3.1   2017-08-20 CRAN (R 3.5.0)
##  jsonlite     1.5     2017-06-01 CRAN (R 3.5.0)
##  knitr      * 1.20    2018-02-20 CRAN (R 3.5.0)
##  labeling     0.3     2014-08-23 CRAN (R 3.5.0)
##  lattice      0.20-35 2017-03-25 CRAN (R 3.5.1)
##  lazyeval     0.2.1   2017-10-29 CRAN (R 3.5.0)
##  lubridate    1.7.4   2018-04-11 CRAN (R 3.5.0)
##  magrittr     1.5     2014-11-22 CRAN (R 3.5.0)
##  MASS         7.3-50  2018-04-30 CRAN (R 3.5.1)
##  memoise      1.1.0   2017-04-21 CRAN (R 3.5.0)
##  methods    * 3.5.1   2018-07-05 local
##  modelr       0.1.2   2018-05-11 CRAN (R 3.5.0)
##  munsell      0.5.0   2018-06-12 CRAN (R 3.5.0)
##  nlme         3.1-137 2018-04-07 CRAN (R 3.5.1)
##  pillar       1.3.0   2018-07-14 cran (@1.3.0)
##  pkgconfig    2.0.1   2017-03-21 CRAN (R 3.5.0)
##  plyr         1.8.4   2016-06-08 CRAN (R 3.5.0)
##  purrr      * 0.2.5   2018-05-29 CRAN (R 3.5.0)
##  R6           2.2.2   2017-06-17 CRAN (R 3.5.0)
##  Rcpp         0.12.18 2018-07-23 CRAN (R 3.5.0)
##  readr      * 1.1.1   2017-05-16 CRAN (R 3.5.0)
##  readxl       1.1.0   2018-04-20 CRAN (R 3.5.0)
##  rlang        0.2.1   2018-05-30 CRAN (R 3.5.0)
##  rmarkdown    1.10    2018-06-11 CRAN (R 3.5.0)
##  rprojroot    1.3-2   2018-01-03 CRAN (R 3.5.0)
##  rstudioapi   0.7     2017-09-07 CRAN (R 3.5.0)
##  rvest        0.3.2   2016-06-17 CRAN (R 3.5.0)
##  scales       0.5.0   2017-08-24 CRAN (R 3.5.0)
##  stats      * 3.5.1   2018-07-05 local
##  stringi      1.2.4   2018-07-20 CRAN (R 3.5.0)
##  stringr    * 1.3.1   2018-05-10 CRAN (R 3.5.0)
##  tibble     * 1.4.2   2018-01-22 CRAN (R 3.5.0)
##  tidyr      * 0.8.1   2018-05-18 CRAN (R 3.5.0)
##  tidyselect   0.2.4   2018-02-26 CRAN (R 3.5.0)
##  tidyverse  * 1.2.1   2017-11-14 CRAN (R 3.5.0)
##  tools        3.5.1   2018-07-05 local
##  utils      * 3.5.1   2018-07-05 local
##  withr        2.1.2   2018-03-15 CRAN (R 3.5.0)
##  xfun         0.3     2018-07-06 CRAN (R 3.5.0)
##  xml2         1.2.0   2018-01-24 CRAN (R 3.5.0)
##  yaml         2.2.0   2018-07-25 CRAN (R 3.5.0)
```

Bibliography

Franken, Holger, Toby Mathieson, Dorothee Childs, Gavain M A Sweetman, Thilo Werner, Ina Tögel, Carola Doce, et al. 2015. “Thermal Proteome Profiling for Unbiased Identification of Direct and Indirect Drug Targets Using Multiplexed Quantitative Mass Spectrometry.” *Nat. Protoc.* 10 (10): 1567–93.

Marazzi, A. 2002. “Bootstrap Tests for Robust Means of Asymmetric Distributions with Unequal Shapes.” *Computational Statistics & Data Analysis* 39 (4). Elsevier: 503–28.

Reinhard, Friedrich B M, Dirk Eberhard, Thilo Werner, Holger Franken, Dorothee Childs, Carola Doce, Maria Fälth Savitski, et al. 2015. “Thermal Proteome Profiling Monitors Ligand Interactions with Cellular Membrane Proteins.” *Nat. Methods* 12 (12): 1129–31.

Savitski, Mikhail M, Friedrich B M Reinhard, Holger Franken, Thilo Werner, Maria Fälth Savitski, Dirk Eberhard, Daniel Martinez Molina, et al. 2014. “Tracking Cancer Drugs in Living Cells by Thermal Profiling of the Proteome.” *Science* 346 (6205): 1255784.