tcplfit2 Vignette

Center for Computational Toxicology and Exposure

Getting started with tcplfit2

The package tcplfit2 contains the core concentration-response functionality of the package tcpl (The ToxCast Pipeline) built to process all of the ToxCast high-throughput screen (HTS) data at the US EPA. Much of the rest of the code in tcpl is used to do data processing, normalization, and database storage. We wanted to reuse the core concentration-response code for other projects, and add extensions to it, which was the origin of the current package `tcplfit2`. The main set of extensions was to include all of the concentration-response models that are contained in the program BMDExpress. These include exponential, polynomial, and power functions in addition to the original Hill, gain-loss and constant models. Additionally, we wanted to include BMD (Benchmark Dose Modeling) outputs, which is simply defining a Benchmark Response (BMR) level and setting the BMD to the concentration where the curve crosses the BMR level. One final addition was to let the hitcall value be a continuous number ranging from 0 to 1. The hitcall number is a product of three probabilities. This vignette describes some functionality of the `tcplfit2` package with a few simple examples.

Example 1: Running a single concentration-response calculation

All calculations use the function `concRespCore` which has several key inputs. The first set are put into a named list called 'row':

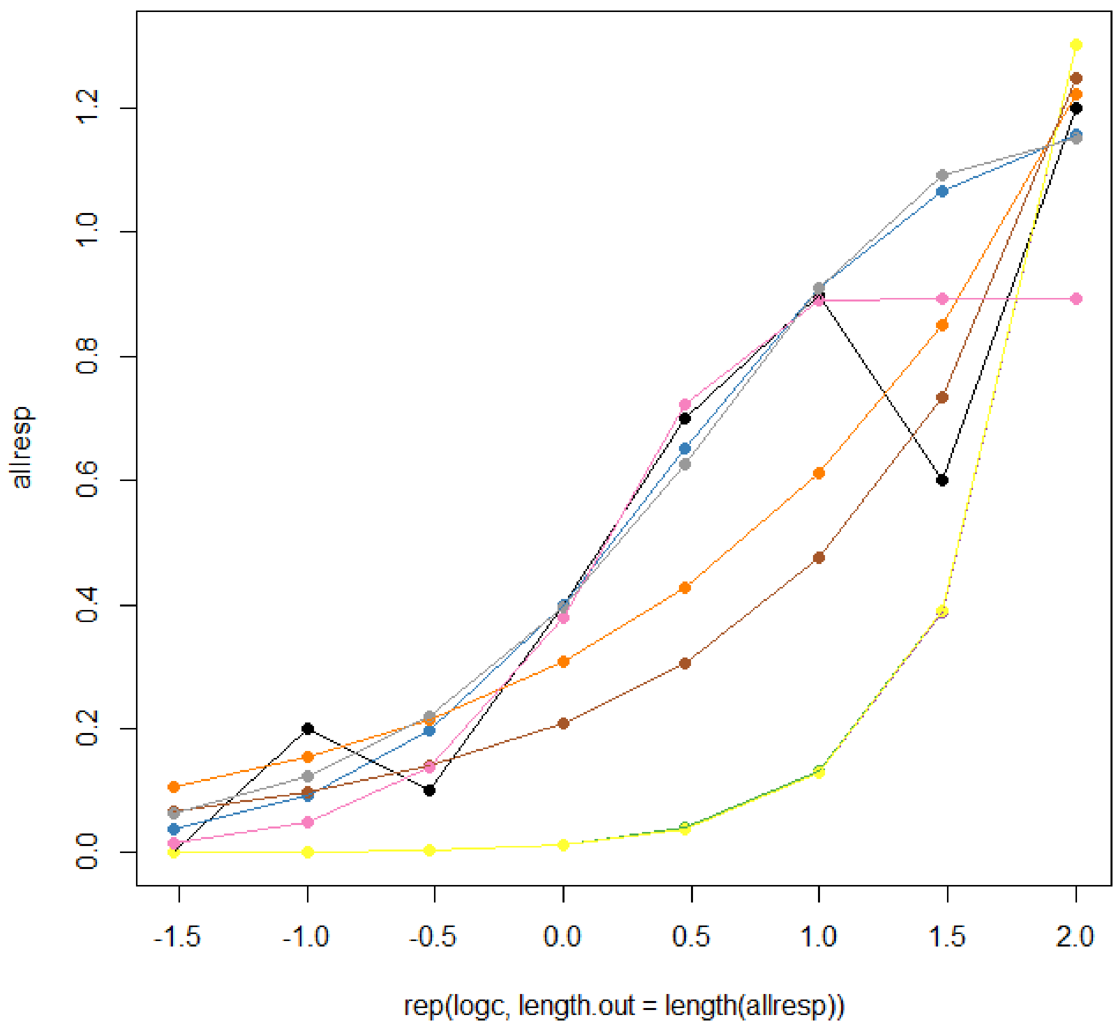
- `conc` - a vector of concentrations (not log concentrations)

- `resp` - a vector of responses, of the same length as `conc`. Note that replicates are allowed, i.e. there can be multiple pairs of conc and resp with the same concentration value.
- `cutoff` - this is the value that the response must exceed before a a curve can be called a hit. For ToxCast, this is usually some multiple (typically 3) of the median absolute deviation (BMAD) around baseline for the lowest two concentration. The user is free to make other choices
- `bmed` - this is the median of the baseline. The entire response series will be shifted by this amount. Set to zero if the data is zero-centered.
- `onesd` - This is one standard deviation of the noise around the baseline. The BMR value = `onesd` * `bmr_scale`. The default `bmr_scale` is 1.349.

The function `concRespCore` can also have other optional elements which will be included in the output. These can be, for instance, the name of the chemical (or other identifiers) or the name of the assay being modeled. Two other parameters might be used. The first is a Boolean `conthits`. If TRUE (the default, and recommended usage), the hitcall returned will be a continuous value between 0 and 1. The other is `do.plot`. If this is set to TRUE (default is FALSE), a plot of the curve will be generated. The user can also select only a subset of the models to be run. The example below has all of the possible ones included. the model `cnst` always needs to be included. For some applications, we exclude the `gnls` model.

To run a simple example, use the following code ...

```
conc <- list(.03,.1,.3,1,3,10,30,100)
resp <- list(0,.2,.1,.4,.7,.9,.6, 1.2)
row = list(conc = conc, resp = resp, bmed = 0, cutoff = 1, onesd = .5,name="some
      chemical")
res <- concRespCore(row,fitmodels = c("cnst", "hill", "gnls", "poly1", "poly2", "pow",
      "exp2", "exp3",
                                     "exp4", "exp5"),conthits = T, do.plot=T)
```



The output of this run will be a data frame with one row, summarizing the results for the winning model.

Show 10 entries Search:

name	n_gt_cutoff	cutoff	fit_method	top_over_cutoff	rmse	a
some chemical	1	1	hill	1.225599329606842	0.1750311682092355	

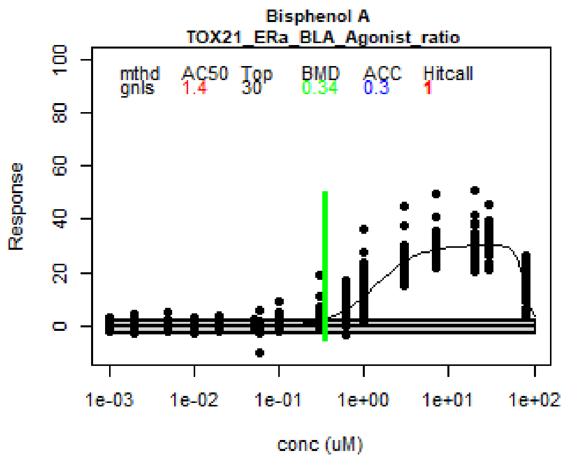
Showing 1 to 1 of 1 entries Previous 1 Next

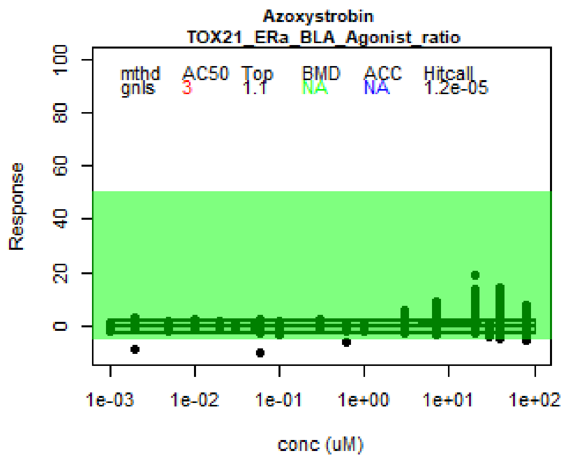
Example 2: Running a series of concentration-response models for a single assay

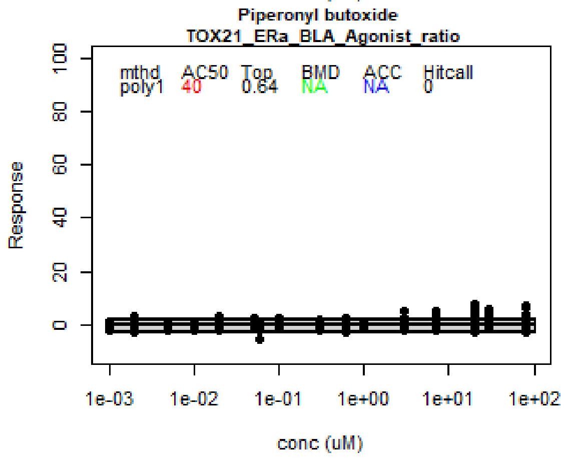
The input data for this example is taken from one of the Tox21 HTS assays, for estrogen receptor (ER) agonist activity. The data is from the mc3 table in the database `invitrodb`, which is the back end for `tcpl`. The input data is Level 3 data, which mean they have already been through some prepossessing steps. The concentration data have been normalized and response data have been transformed. This example will run 6 chemicals out of the 100 that are included in the data set, and will create plots for these. The plotting routine `concRespPlot` is somewhat generic, and we anticipate that users will make their own version of this. To run this example, use the following code …

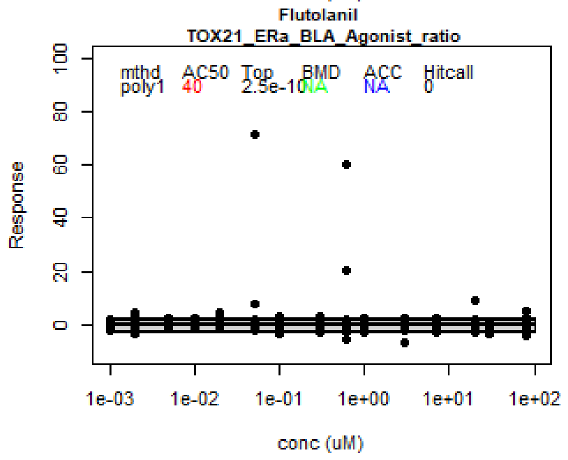
```
# read in the data
# Loading in the level 3 example data set from invitrodb
data("mc3")

# set up a 3 x 2 grid for the plots
oldpar <- par(no.readonly = TRUE)
on.exit(par(oldpar))
par(mfrow=c(3,2),mar=c(4,4,2,2))

# determine the background variation
temp <- mc3[mc3$logc<= -2,"resp"]
bmad <- mad(temp)
onesd <- sd(temp)
cutoff <- 3*bmad

# select six samples. Note that there may be more than one sample processed for a given
       chemical
spid.list <- unique(mc3$spid)
spid.list <- spid.list[1:6]

for(spid in spid.list) {
  # select the data for just this sample
  temp <- mc3[is.element(mc3$spid,spid),]

  # The data file has stored concentration in log10 form, so fix that
  conc <- 10**temp$logc
  resp <- temp$resp

  # pull out all of the chemical identifiers and the name of the assay
  dtxsid <- temp[1,"dtxsid"]
  casrn <- temp[1,"casrn"]
  name <- temp[1,"name"]
  assay <- temp[1,"assay"]

  # create the row object
  row <- list(conc = conc, resp = resp, bmed = 0, cutoff = cutoff, onesd =
       onesd,assay=assay,dtxsid=dtxsid,casrn=casrn,name=name)
```

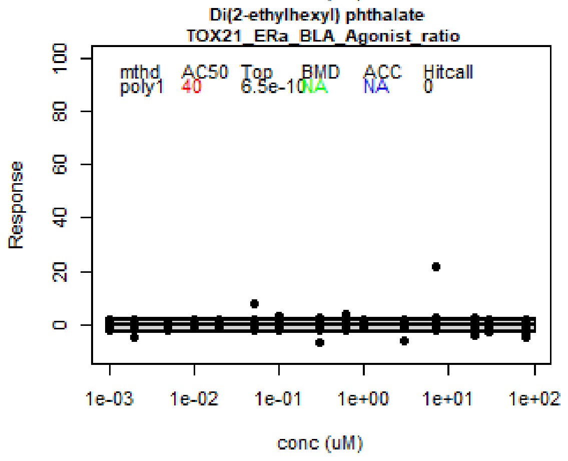
```
    # run the concentration-response modeling for a single sample
    res <- concRespCore(row,fitmodels = c("cnst", "hill", "gnls", "poly1", "poly2", "pow",
         "exp2", "exp3",
                                               "exp4", "exp5"),conthits = T, aicc =
         F,bidirectional=F)

    # plot the results
    concRespPlot(res,ymin=-10,ymax=100)
  }
```

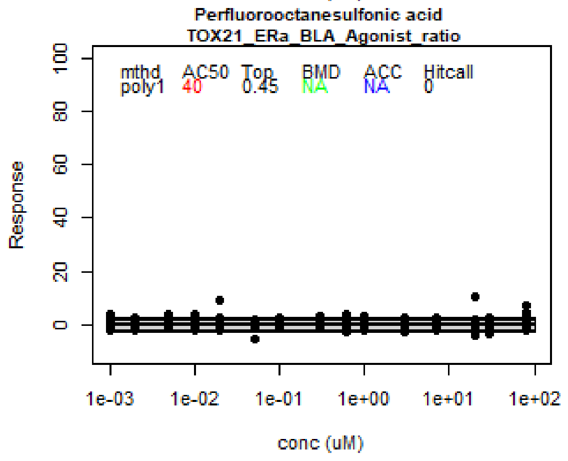










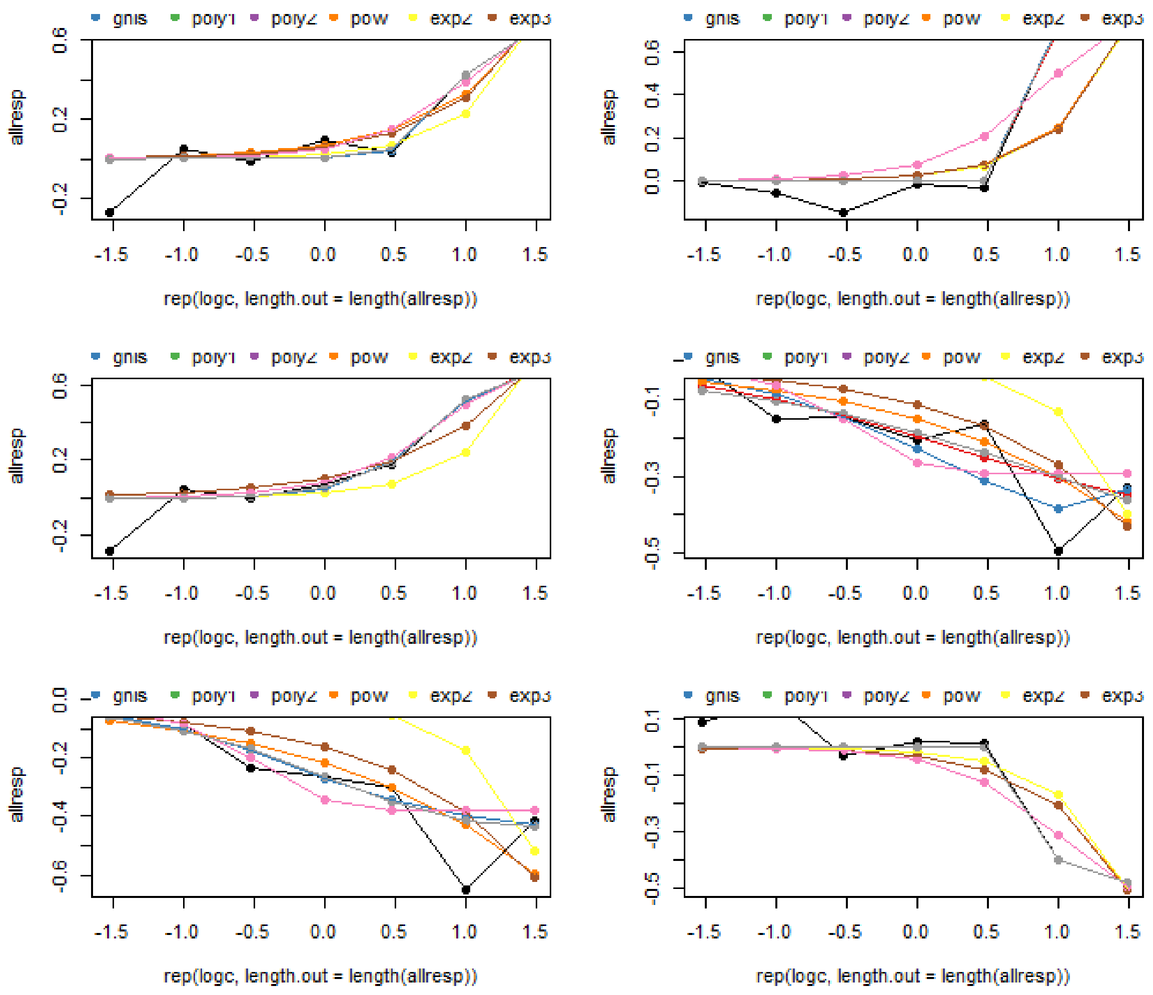


One would typically save the result rows in a data frame end export these for further analysis. You could remove the plotting function from the current loop and have a loop that read from the overall results data frame and only plot selected results (e.g. those with significant responses).

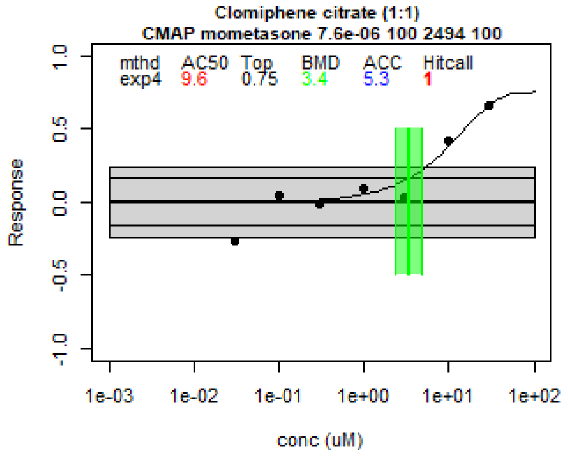
Example 3: Plotting concentration-response modeling on transcriptional signatures

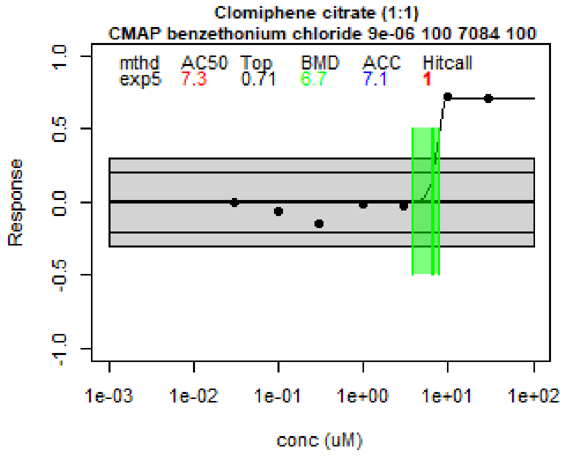
The input data for this example contains 6 signatures for one chemical in a transcriptomics data set. Each signature is a different assay endpoint of the given chemical and is stored as one row in the data. This data set is a sample from the signature scoring method that provides the cutoff, one standard deviation, and the concentration-response data. The example illustrates two kinds of plots available in `tcplfit2`. In the call to `concRespCore()`, the argument `do.plot` is set to `TRUE`, which provides a simple plot showing results of all the different curve fitting methods. Next, utilizing the function `concRespPlot()` provides a more informative plot for the winning model.

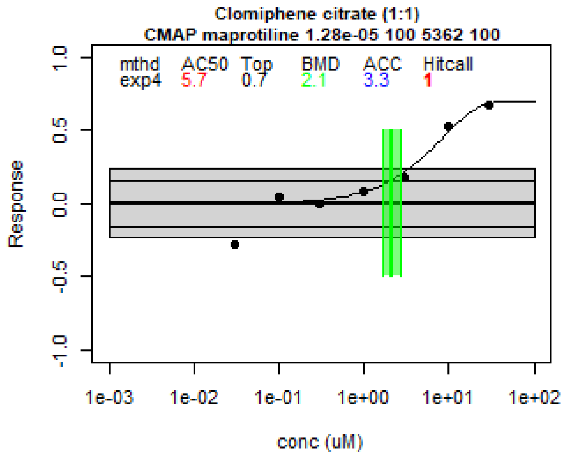
```r
# call additional R packages
library(stringr)  # string management package

# read in the file
data("signatures")

# set up a 3 x 2 grid for the plots
oldpar <- par(no.readonly = TRUE)
on.exit(par(oldpar))
par(mfrow=c(3,2),mar=c(4,4,2,2))

# fit 6 observations in signatures
for(i in 1:nrow(signatures)){
  # set up input data
  row = list(conc=as.numeric(str_split(signatures[i,"conc"],"\\|")[[1]]),
             resp=as.numeric(str_split(signatures[i,"resp"],"\\|")[[1]]),
             bmed=0,
             cutoff=signatures[i,"cutoff"],
             onesd=signatures[i,"onesd"],
             name=signatures[i,"name"],
             assay=signatures[i,"signature"])
  # run concentration-response modeling (1st plotting option)
  out = concRespCore(row,conthits=F,do.plot=T)
  if(i==1){
    res <- out
  }else{
    res <- rbind.data.frame(res,out)
  }
}
```

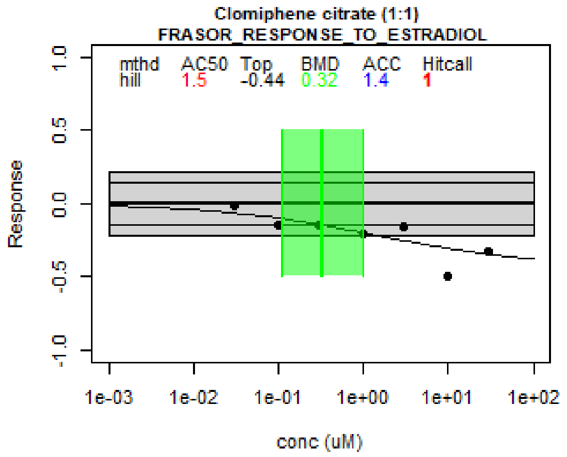


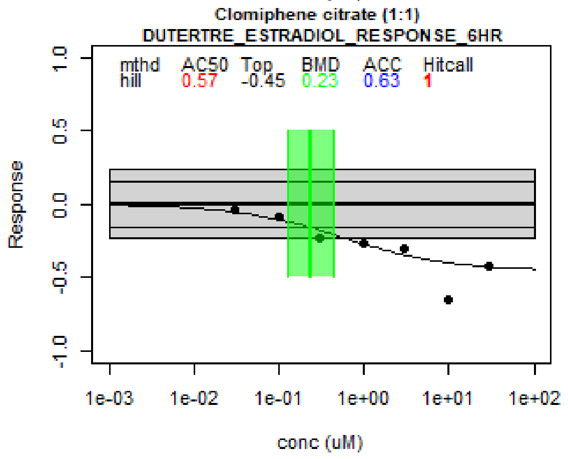
```
# set up a 3 x 2 grid for the plots
oldpar <- par(no.readonly = TRUE)
on.exit(par(oldpar))
par(mfrow=c(3,2),mar=c(4,4,2,2))
# plot results using `concRespPlot`(2nd plotting option)
for(i in 1:nrow(res)){
  concRespPlot(res[i,],ymin=-1,ymax=1)
}
```

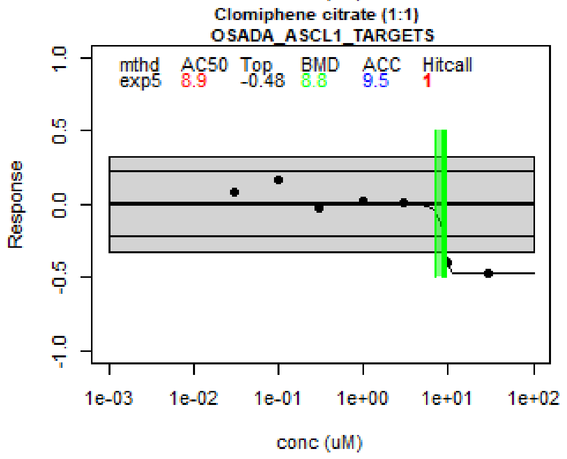












Example 4: Running tcpl-like multi-concentration response data without a database connection

The ToxCast pipeline `tcpl` is an R package that manages, curve-fits, plots, and stores ToxCast data to populate its linked MySQL database, InvitroDB. The original `tcplFit()` function within `tcpl` performed basic concentration response curve fitting. Processing with tcpl_v3 and beyond depends on `tcplfit2` to allow a wider variety of concentration-response models when using `invitrodb` in the 4.0 schema and beyond. Within this update, `tcplLite` became deprecated within `tcpl` because `tcplFit2` can be used to curve-fit data and make hitcalls independent of invitrodb, as the example below illustrates. For additional information, please consult vignettes for `library(tcpl)` at https://CRAN.R-project.org/package=tcpl.

The input for this example comes from the ACEA_AR assay. Data from the assay component ACEA_AR_agonist_80hr was analyzed in the positive analysis fitting direction relative to DMSO as the neutral control and baseline of activity. Using a electrical impedance as a cell growth reporter, increased activity can be used to infer increased signaling at the pathway-level for the androgen receptor (as encoded by the AR gene). Given heterogeneous assay data, source data often must go through pre-processing steps to transform into a uniform data format, often like this level 0. The below table is identical to the multi-concentration level 0 data (mc0) table that would be seen in `invitrodb` and recognized by `tcpl`. Columns include:

- m0id = Level 0 id
- spid = Sample id
- acid = Unique assay component id; unique numeric id for each assay component
- apid = Assay plate id
- coli = Column index (location on assay plate)
- rowi = Row index (location on assay plate)
- wllt = well type
- wllq = well quality
- conc = concentration
- rval = raw value
- srcf = Source file name
- clowder_uid = clowder unique id for source files
- git_hash = hash key for pre-processing scripts

```
# Loading in the level 0 example data set from invitrodb
data("mc0")
library(data.table)
dat <- mc0
DT::datatable(head(dat[wllt=='t',]),rownames= FALSE, options = list(scrollX = T))
```

Show 10 entries Search:

m0id	spid	acid	apid	rowi
519762672	TP0001364A01	1829	Experiment.ID:1502051323HT1_A113641_AP01_RA_P09	
519762768	TP0001364A02	1829	Experiment.ID:1502051323HT1_A113641_AP01_RA_P09	
519762864	TP0001364A03	1829	Experiment.ID:1502051323HT1_A113641_AP01_RA_P09	
519762960	TP0001364A04	1829	Experiment.ID:1502051323HT1_A113641_AP01_RA_P09	
519763056	TP0001364A05	1829	Experiment.ID:1502051323HT1_A113641_AP01_RA_P09	

519763152	TP0001364A06	1829	Experiment.ID:1502051323HT1_A113641_AP01_RA_P09

Showing 1 to 6 of 6 entries Previous 1 Next

To run standalone `tcplfit2` fitting without the need for a MySQL database connection like `invitrodb`, the user will replicate stepping through the multiple levels of processing. A detailed explanation of processing levels can be found within `tcpl`'s Data Processing vignette.

Level 1 importantly establishes the concentration index. The concentration index is simply the distinct concentrations ranked from lowest to highest, and this index can be used to calculate the baseline median absolute deviation for an assay.

```r
library(tcpl)
#> Warning: package 'tcpl' was built under R version 4.2.3
#> tcpl (v3.0.1) loaded with the following settings:
#>    TCPL_DB:    C:/Program Files/R/R-4.2.2/library/tcpl/csv
#>    TCPL_USER:  NA
#>    TCPL_HOST:  NA
#>    TCPL_DRVR:  tcplLite
#> Default settings stored in tcpl config file. See ?tcplConf for more information.
## Order by the following columns
setkeyv(dat, c('acid', 'srcf', 'apid', 'coli', 'rowi', 'spid', 'conc'))

## Define replicate id (rpid) column for test compound wells
nconc <- dat[wllt == "t" , ## denotes test well as the well type (wllt)
             list(n = lu(conc)), #total number of unique concentrations
             by = list(acid, apid, spid)][ , list(nconc = min(n)), by = acid]
dat[wllt == "t" & acid %in% nconc[nconc > 1, acid],
    rpid := paste(acid, spid, wllt, srcf, apid, "rep1", conc, sep = "_")]
dat[wllt == "t" & acid %in% nconc[nconc == 1, acid],
    rpid := paste(acid, spid, wllt, srcf, "rep1", conc, sep = "_")]

## Define rpid column for non-test compound wells
dat[wllt != "t",
    rpid := paste(acid, spid, wllt, srcf, apid, "rep1", conc, sep = "_")]

## set repid based on rowid
dat[, dat_rpid := rowid(rpid)]
dat[, rpid := sub("_rep[0-9]+.*", "",rpid, useBytes = TRUE)]
dat[, rpid := paste0(rpid,"_rep",dat_rpid)]

# Define concentration index
```

```
indexfunc <- function(x) as.integer(rank(unique(x))[match(x, unique(x))])
dat[ , cndx := indexfunc(conc), by = list(rpid)]
```

Adjustments

Levels 2 and 3 are used for data adjustments and normalization. Generally if the response values (rval) need to be logged or transformed in some way from their original values this is where that adjustment would occur. Response values that have been transformed are referred as the corrected value and is represented by `cval`. However, in this case, the corrected value (cval) is identical to the original response values (rval).

```
# If no adjustments are required for the data, the corrected value (cval) should be set as
        original rval
dat[,cval := rval]

## Poor well quality (wllq) wells should be removed
dat <- dat[!wllq == 0,]

## Fitting generally cannot occur if response values are NA therefore values need to be
        removed
dat <- dat[!is.na(cval),]

## A column for log10 concentration is added as some of the mc3 methods require logc. Given
        logging concentration, conc=0 are not allowed therefore a dummy non-zero value
        should be used
dat[conc == 0 , conc := 0.0001]
dat[ , logc := log10(conc)]

#As a final step to prepare the dataset tcplfit2 processing, a dummy aeid is required if
        using mc3_mthds from tcpl
dummy_aeid <- 99999
dat[,aeid := dummy_aeid]

## Set aeid as a key
setkey(dat,aeid)
```

Once the data is initialized to a point where the required fields are available, the methods included in the `tcpl` package can be identified and applied without the need for a database connection. You can see the list of available methods for Level 3 in the table below:

```
mthd_funcs <- tcpl:::mc3_mthds()
DT::datatable(tcpl::tcplMthdList(3),rownames= FALSE, options = list(scrollX = T))
```

Show 10 entries Search:

mc3_mthd_id	mc3_mthd	desc
1	none	apply no level 3 method
2	bval.apid.lowconc.med	plate-wise baseline based on low conc median value
3	pval.apid.medpcbyconc.max	plate-wise median response of positive control (max)
4	pval.apid.medpcbyconc.min	plate-wise median response of positive control (min)
5	resp.pc	response percent activity
6	resp.multneg1	multiply the response by -1
7	resp.log2	take the log base 2 of the response
8	resp.mult25	multiply the response by 25
9	resp.fc	calculate response as fold-change
11	bval.apid.nwlls.med	plate-wise baseline based on neutral ctrl median value

Showing 1 to 10 of 37 entries Previous 1 2 3 4 Next

Normalization

Here three normalization methods are selected and applied to the data. Note because of the way `tcpl` handles the application of functions, the dataframe must be called `dat`. In the future, `tcpl` will export these functions so that they can be applied to any dataset without the need for a specific name or dummy aeid.

```
# apply level 3 methods
## These methods directly apply the normalization methods from tcpl without the need for a
        DB connection
lapply(mthd_funcs[["bval.apid.nwlls.med"]](dummy_aeid), eval)
lapply(mthd_funcs[["pval.apid.medncbyconc.min"]](dummy_aeid),eval)
lapply(mthd_funcs[["resp.pc"]](dummy_aeid),eval)
```

Level 4 determines the baseline variability, or noise, that will later be used for cutoff calculation. Using the established concentration index, the level 4 methods can be loaded in a similar way to level 3.

```
mthd_funcs_l4 <- tcpl:::mc4_mthds()
DT::datatable(tcpl::tcplMthdList(4), rownames= FALSE, options = list(scrollX = T))
```

Show 10 entries Search:

mc4_mthd_id	mc4_mthd	desc
1	bmad.aeid.lowconc.twells	bmad based on two lowest concentration of treatment wells
2	bmad.aeid.lowconc.nwells	bmad based on two lowest concentration of nwells

Showing 1 to 2 of 2 entries Previous 1 Next

There are much fewer level 4 methods, but generally it is a requirement to assign a method that calculates the bmad and assign a method that calculates the standard deviation of the noise for `tcplfit2` fitting.

```
# apply level 4 methods
## These methods directly apply the noise calculation and fitting methods from tcpl without
      the need for a DB connection
lapply(mthd_funcs_l4[["bmad.aeid.lowconc.twells"]](),eval)
lapply(mthd_funcs_l4[["onesd.aeid.lowconc.twells"]](),eval)
lapply(mthd_funcs_l4[["bidirectional.false"]](),eval)
```

Dose-Response Curve Fitting

After methods up to level 4 have been applied, the model fitting can begin. In `tcpl`, this would be considered level 4, and is where `tcplfit2` is used to fit all of the models as a dependency for `tcpl`.

```
#do tcplfit2 fitting
myfun <- function(y) {
  res <- tcplfit2::tcplfit2_core(y$conc,
                                 y$resp,
                                 cutoff = unique(y$bmad),
                                 bidirectional = TRUE,
                                 verbose = FALSE,
                                 force.fit = TRUE,
                                 fitmodels = c("cnst", "hill", "gnls", "poly1",
                                          "poly2", "pow", "exp2", "exp3",
                                          "exp4", "exp5")
                               )
  list(list(res)) #use list twice because data.table uses list(.) to look for values to
         assign to columns
}
```

The following code performs dose-response modeling for all spids in the dataset. **Warning: The fitting step for the full data set, `dat`, can take 7-10 minutes to run.** Hence the code chunk following provides a subset example of data for curve fitting and hitcalling. The subset data only contains records of six samples.

```
# only want to run tcplfit2 for test wells in this case
# this chunk doesn't run, fit the curves on the subset below
dat[wllt == 't',params:= myfun(.SD), by = .(spid)]
```

```
# create a subset that contains 6 samples and run curve fitting
subdat <- dat[spid %in% unique(spid)[10:15],]
subdat[wllt == 't',params:= myfun(.SD), by = .(spid)]
```

Continuous Hitcalling

After all of the models have been fit, hitcalling can occur. The output of level 4 can be fed directly into the `tcplhit2_core` function. The results are then pivoted and shown in the resulting datatable.

```
myfun2 <- function(y) {
  res <- tcplfit2::tcplhit2_core(params = y$params[[1]],
                                 conc = y$conc,
                                 resp = y$resp,
                                 cutoff = 3*unique(y$bmad),
                                 onesd = unique(y$osd)
                                 )
  list(list(res))
}

# continute with hitcalling
res <- subdat[wllt == 't', myfun2(.SD), by = .(spid)]

#pivot wider
res_wide <- rbindlist(Map(cbind, spid = res$spid, res$V1))
```

```
DT::datatable(res_wide,options = list(scrollX = T))
```

Show 10 entries Search:

	spid	n_gt_cutoff	cutoff	fit_method	top_over_cutoff	
1	TP0001366A03	0	49.28306384522267	gnls	0.2819010005117448	8.
2	TP0001366A04	0	49.28306384522267	poly1	0.2851168135965973	
3	TP0001366A05	0	49.28306384522267	gnls	0.2305796630362356	9.
4	TP0001366A06	0	49.28306384522267	poly1	0.1301443807450612	1
5	TP0001366A07	0	49.28306384522267	gnls	0.2548054752540804	8.
6	TP0001366A08	0	49.28306384522267	gnls	0.4400229249416137	7.

Showing 1 to 6 of 6 entries Previous 1 Next

The same hitcalling can be done with the full data set, `dat`, as well.

This output table is the same format as the `res` table in example 3. Users can use the plot code in the chunk that demonstrates the use of `concRespPlot` in example 3 to visualize fits from this output table.