Introduction to tcplfit2

Center for Computational Toxicology and Exposure, US EPA

Introduction

The package `tcplfit2` is used to perform basic concentration response curve fitting. The original tcplFit() functions in the ToxCast Pipeline (tcpl) package performed basic concentration response curve fitting to 3 models: Hill, gain-loss [a modified Hill], and constant. With `tcplfit2`, the core concentration-response functionality of the package

`tcpl` has been expanded built to process high-throughput screen (HTS) data generated at the US Environmental Protection Agency, including targeted ToxCast, high-throughput transcriptomics (HTTr) and high-throughput phenotypic profiling (HTPP) screening results. The `tcpl` R package continues to be used to manage, curve-fit, plot, and populate its linked MySQL database, InvitroDB. Processing with `tcpl` version 3.0 and beyond depends on the stand-alone tcplFit2 package to allow a wider variety of concentration-response models when using invitrodb in the 4.0 schema and beyond.

The main set of extensions includes all of the concentration-response models that are contained in the program BMDExpress. These include exponential, polynomial (1 & 2), and power functions in addition to the original Hill, gain-loss and constant models. Similar to the program BMDExpress, a defined Benchmark Response (BMR) level is used to estimate a benchmark dose (BMD), which is the concentration where the curve-fit intersects with this BMR threshold. One final addition was to let the hitcall value be a continuous number ranging from 0 to 1 (in contrast to binary hitcall values from tcplFit(). Continuous hitcall in `tcplfit2` are defined as the product of three proportional weights: 1) the AIC of the winning model is better than the constant model (i.e. winning model is not fit to background noise), 2) at least one concentration has a median response that exceeds cutoff, and 3) the top from the winning model exceeds the cutoff.

While developed primarily to extend of ToxCast curve-fitting, the tcplfit2 package is written to be generally applicable to the chemical-screening community for standalone applications.

Getting Started with tcplfit2

This vignette describes some functionality of the `tcplfit2` package with a few simple standalone examples.

Example 1: Concentration-response Modeling for a Single Series with `concRespCore`

`concRespCore` is the main wrapper function utilizing two other utility functions, `tcplfit2_core` and `tcplhit2_core`, to perform curve-fitting, hit-calling and potency estimation. This example shows how to use the `concRespCore` function, refer to Example 2 to see how `tcplfit2_core` and `tcplhit2_core` may be used separately. The first argument for `concRespCore` is a named list, called ‘row’, containing the following inputs:

- `conc` - a numeric vector of concentrations (not log concentrations).
- `resp` - a numeric vector of responses, of the same length as `conc`. Note that replicates are allowed, i.e. there may be multiple response values (`resp`) for one concentration dose group. These should be input as pairs of `conc` and `resp` values.
- `cutoff` - a single numeric value indicating the response at which a relevant level of biological activity occurs. This value is typically used to determine if a curve is classified as a "hit". In ToxCast, this is usually some multiple (typically 3) of the median absolute deviation around the baseline (BMAD), where the baseline is defined as the two lowest concentrations. However, users are free to make other choices more appropriate for their given assay and data.
- `bmed` - a single numeric value giving the baseline median response. If set to zero then the data is already zero-centered. Otherwise, this value is used to zero-center the data by shifting the entire response series by the specified amount.
- `onesd` - a single numeric value giving one standard deviation around the baseline. This value is used to calculate the benchmark response (BMR), where $BMR = \frac{\text{`onesd`}}{\text{bmr_scale}}$. The `bmr_scale` defaults to 1.349.

The `row` object may include other elements which provide annotation which will be included as part of the `concRespCore` function output – for example, chemical names (or other identifiers), assay name, name of the response being modeled, etc.

A user may also need to include other arguments in the `concRespCore` function, which internally control the execution of curve-fitting, hit-calling, and potency estimation:

- `conthits` - Boolean argument. If TRUE (the default, and recommended usage), the hitcall returned will be a continuous value between 0 and 1.
- `errfun` - Allows user to specify the distribution of errors. The default is "dt4", models are fitted assuming the errors follow a Student's t-distribution with a degree of freedom of 4. Can assume normal distributed errors by changing it to "dnorm".
- `poly2.biphasic` - If TRUE (the default, and recommended usage), the polynomial 2 model will attempt fit a biphasic curve to the response (i.e. increase then decrease or vice versa). Can force monotonic fitting with FALSE (parabola with the vertex centers at 0).
- `do.plot` - If this is set to TRUE (default is FALSE), a plot of all fitted curves will be generated. This plotting functionality is outdated by another plotting function in this package, `plot_allcurves`. More on this can be found in the Appendix and the plotting vignette.

- `fitmodels` - a character vector indicating which models to fit the concentration-response data with. If the `fitmodels` parameter is specified, the constant model (`cnst`) model must be included since it is used for comparison in the hitcalling process. However, all other models may be omitted by the user, for example the gain-loss (`gnls`) model is excluded in some applications. For a full list of potential arguments, refer the function documentation (`?concRespCore`).

The following code provides a simple example for setting up the input and executing the modeling with `concRespCore`.

```
# tested concentrations
  conc <- list(.03,.1,.3,1,3,10,30,100)
# observed responses at respective concentrations
  resp <- list(0,.2,.1,.4,.7,.9,.6, 1.2)
# row object with relevant parameters
  row = list(conc = conc, resp = resp, bmed = 0, cutoff = 1, onesd =
.5,name="some chemical")
# execute concentration-response modeling through potency estimation
  res <- concRespCore(row,
                      fitmodels = c("cnst", "hill", "gnls",
                                    "poly1", "poly2", "pow", "exp2", "exp3",
                                        "exp4", "exp5"),
                      conthits = 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	
some chemical	1	1	hill	1.225599329606842	0.175031168

Showing 1 to 1 of 1 entries Previous 1 Next

One can plot winning curve by passing the output (`res`) to the function `concRespPlot2`.

This function returns a basic `ggplot` object, which is meant to leverage the flexibility and modularity of `ggplot2` object to allow users the ability to customize the plot by adding layers of detail. For more information on customizing plotts we refer users to the plotting vignette.

```
# plot the winning curve from example 1, add a title
concRespPlot2(res, log_conc = TRUE) + ggtitle("Example 1: Chemical A")
```

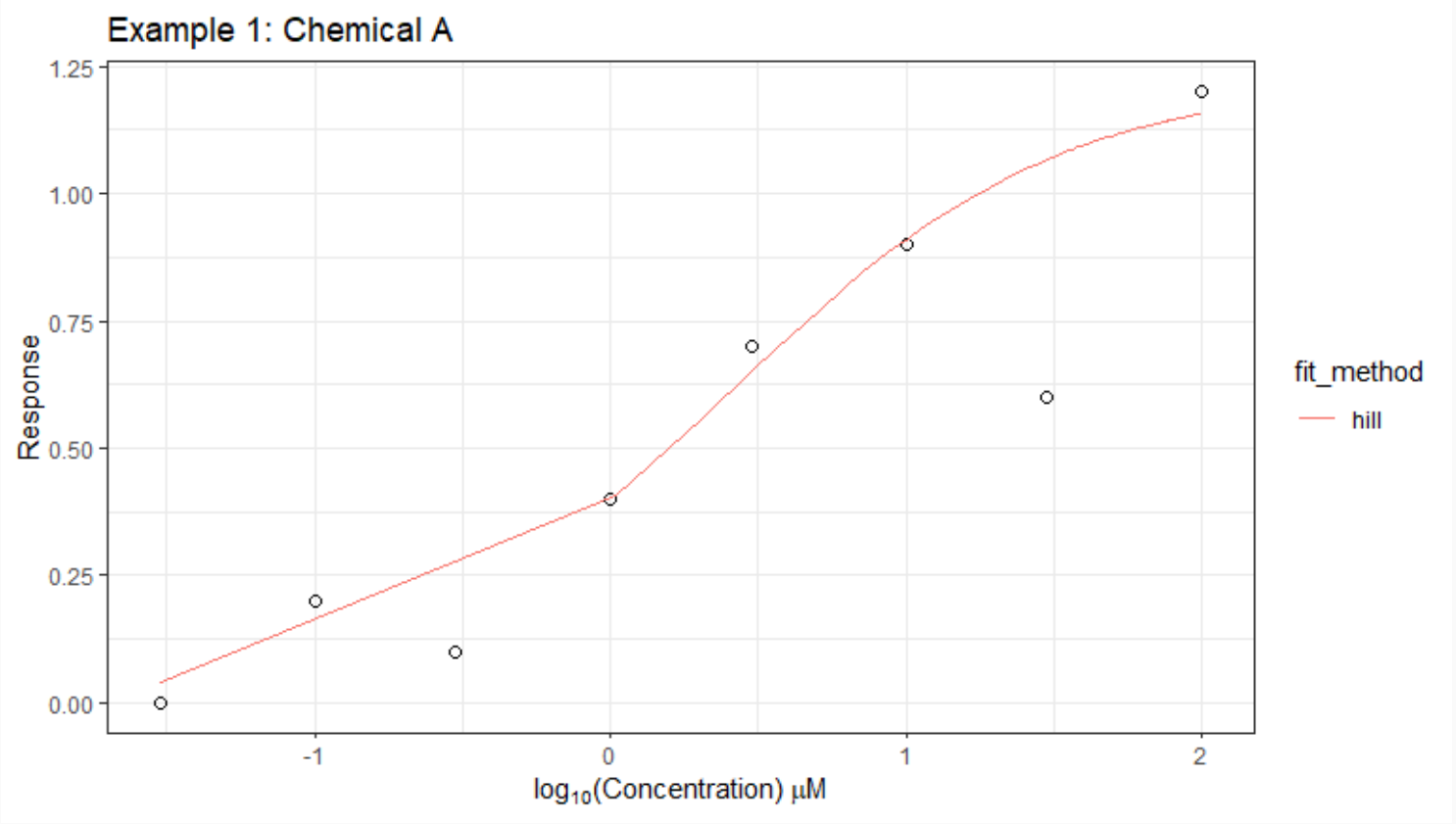


Figure 1: *The winning model fit for a single concentration-response series. The concentrations (x-axis) are in \(log_{10}\) units.*

Example 2: Concentration-response Modeling for Multiple Series with `tcplfit2_core` and `tcplhit2_core`

This example shows fitting a set of concentration-response series from a single assay using the `tcplfit2_core` and `tcplhit2_core` sequentially. Using the functions sequentially allows users greater flexibility to examine the intermediate output. For example, the output from `tcplfit2_core` contains model parameters for all model fit to concentration-response

series provided. Furthermore, `tcplfit2_core` results may be passed to `plot_allcurves`, which generates a comparative plot of all curves fit to a concentration-response series.

Here, data from a Tox21 high-throughput screening (HTS) assays measuring for estrogen receptor (ER) agonist activity is examined. The data was processed by the ToxCast pipeline (`tcpl`) is stored and from the Level 3 (mc3) table in the database `invitrodb`. At Level 3, data has already undergone pre-processing steps (prior to `tcpl`) and been processed through to Level 3, including concentration normalization and transformation of response values. For this example, 6 out of 100 available from samples from `mc3` are selected. Example 3 highlights how to process from source data.

The following code demonstrates how to set up the input data and execute curve-fitting and hitcalling with the `tcplfit2_core` and `tcplhit2_core` functions, respectively.

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

The output from `tcplfit2_core` is a nested list containing the following elements:

- `modelnames` - a vector of the model names fit to the data.
- `errfun` - a character string specifying the assumed error distribution for model fitting.
- Nested list elements, specified by its model name, containing the estimated model parameters and other details when the corresponding model is fit to the provided data.

```
# shows the structure of the output object from tcplfit2_core (only top level)
str(model_fits[[1]],max.lev = 1)
#> List of 12
#>  $ cnst      :List of 5
#>  $ hill      :List of 17
#>  $ gnls      :List of 22
#>  $ poly1     :List of 13
#>  $ poly2     :List of 15
#>  $ pow       :List of 15
#>  $ exp2      :List of 15
#>  $ exp3      :List of 17
#>  $ exp4      :List of 15
#>  $ exp5      :List of 17
```

```
#>  $ modelnames: chr [1:10] "cnst" "hill" "gnls" "poly1" ...
#>  $ errfun    : chr "dt4"
```

Below the structure of the “hill” element are shown as an example of details contained in each of the model name elements:

- `success` - a binary indicator, where 1 indicates the fit was successful.
- `aic` - the Akaike Information Criterion (AIC)
- `cov` - a binary indicator, where 1 indicates the estimating the inverted hessian was successful
- `rme` - the root mean square error around the curve
- `modl` - a numeric vector of model predicted responses at the given concentrations
- `tp`, `ga`, `p` - estimated model parameters for the “hill” model
- `tp_sd`, `ga_sd`, `p_sd` - standard deviations of the model parameters for the “hill” model
- `er` - the numeric error term
- `er_sd` - the numeric value for the standard deviation of the error term
- `pars` - a character vector containing the name of model parameters estimated for the “hill” model
- `sds` - a character vector containing the name of parameters storing the standard deviation of model parameters for the “hill” model
- `top` - the predicted maximal response
- `ac50` - the concentration inducing 50% of the maximal predicted response

All of these details are provided for other models, except for the constant model. The constant model only includes the `success`, `aic`, `rme`, and `er` elements.

```
str(model_fits[[1]][["hill"]])
#> List of 17
#>  $ success: int 1
#>  $ aic    : num 3931
#>  $ cov    : int 1
#>  $ rme    : num 5.4
#>  $ modl   : num [1:675] 2.84e+01 1.96e-03 1.22e+01 1.87e-05 2.04e-01 ...
#>  $ tp     : num 28.5
#>  $ ga     : num 1.15
#>  $ p      : num 2.02
#>  $ er     : num 1.07
```

```
#>  $ tp_sd  : num 0.386
#>  $ ga_sd  : num 0.0591
#>  $ p_sd   : num 0.133
#>  $ er_sd  : num 0.0425
#>  $ pars   : chr [1:4] "tp" "ga" "p" "er"
#>  $ sds    : chr [1:4] "tp_sd" "ga_sd" "p_sd" "er_sd"
#>  $ top    : num 28.5
#>  $ ac50   : num 1.15
```

The code below allows us to compile and display all the plots generated by `plot_allcurves` above:

```
grid.arrange(grobs=plt_lst,ncol=2)
```

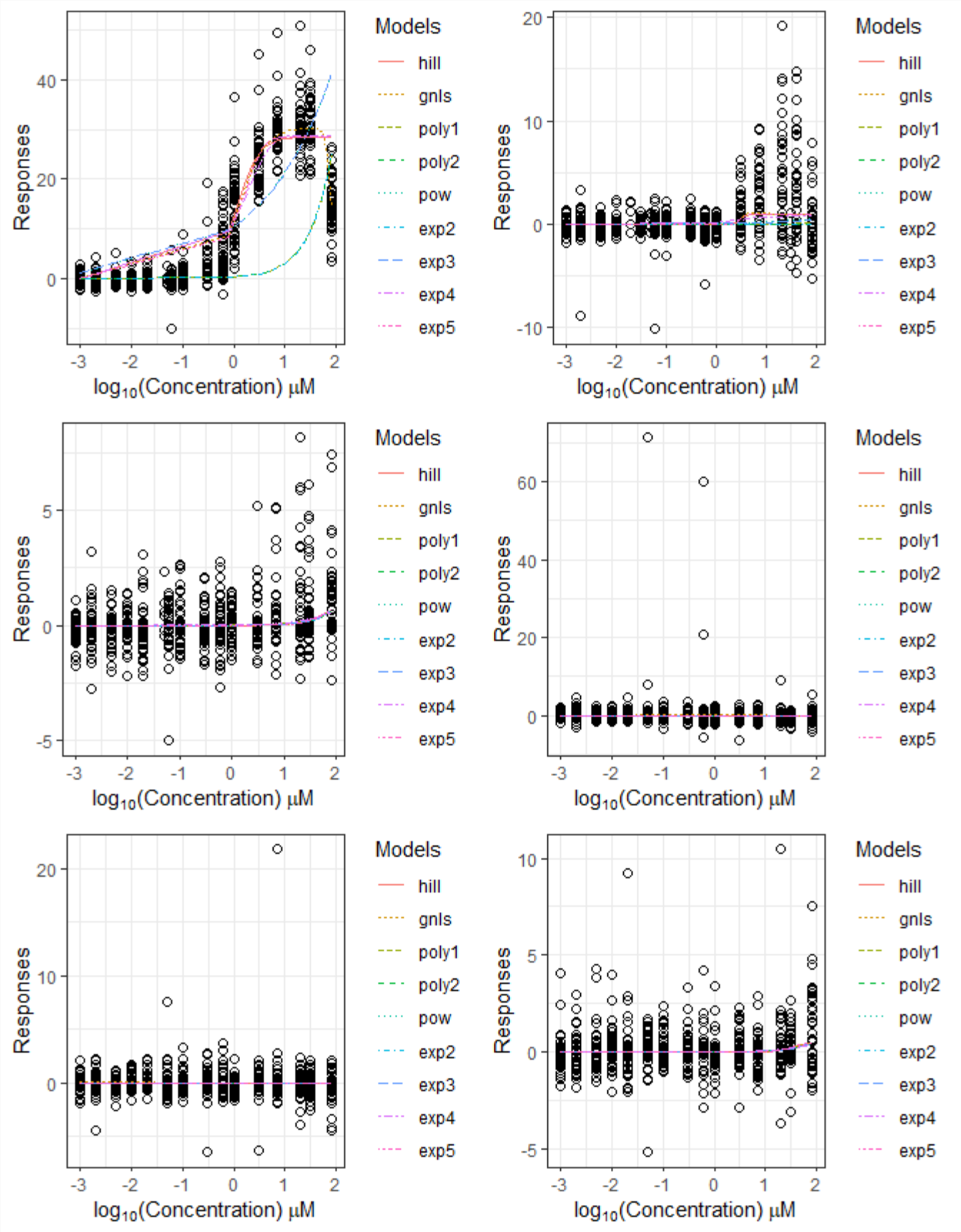


Figure 2: *Example plots generated from* `plot_allcurves`*. Each plot depicts all model fits for a given sample (i.e. concentration-response series). In the plots, observed values are represented by the open circles and each model fit to the data is represented with a different*

color and line type. Concentrations (x-axis) are displayed in \(log_{10}\) units.

When running the fitting and hitcalling functions sequentially, one can save the result rows from `tcplhit2_core` in a data frame structure and export it for further analysis, see for loop above.

Show 10 entries Search:

dtxsid	casrn	name	ass
DTXSID7020182	80-05-7	Bisphenol A	TOX21_ERa_BLA_Agonist_ratio
DTXSID0032520	131860-33-8	Azoxystrobin	TOX21_ERa_BLA_Agonist_ratio
DTXSID1021166	51-03-6	Piperonyl butoxide	TOX21_ERa_BLA_Agonist_ratio
DTXSID8024109	66332-96-5	Flutolanil	TOX21_ERa_BLA_Agonist_ratio
DTXSID5020607	117-81-7	Di(2-ethylhexyl) phthalate	TOX21_ERa_BLA_Agonist_ratio
DTXSID3031864	1763-23-1	Perfluorooctanesulfonic acid	TOX21_ERa_BLA_Agonist_ratio

Showing 1 to 6 of 6 entries Previous 1 Next

One can also pass output from `tcplhit2_core` directly to `concRespPlot2` to plot the best model fit, as shown in Example 1. The code below demonstrates how to select a single row/result (e.g. one with a significant concentration-responses) and plot the winning model with `concRespPlot2`, along with a minor customization using `ggplot2` layers.

```
# plot the first row
concRespPlot2(result_table[1,],log_conc = TRUE) +
  ggtitle(paste(result_table[1,"dtxsid"], result_table[1,"name"]))
```

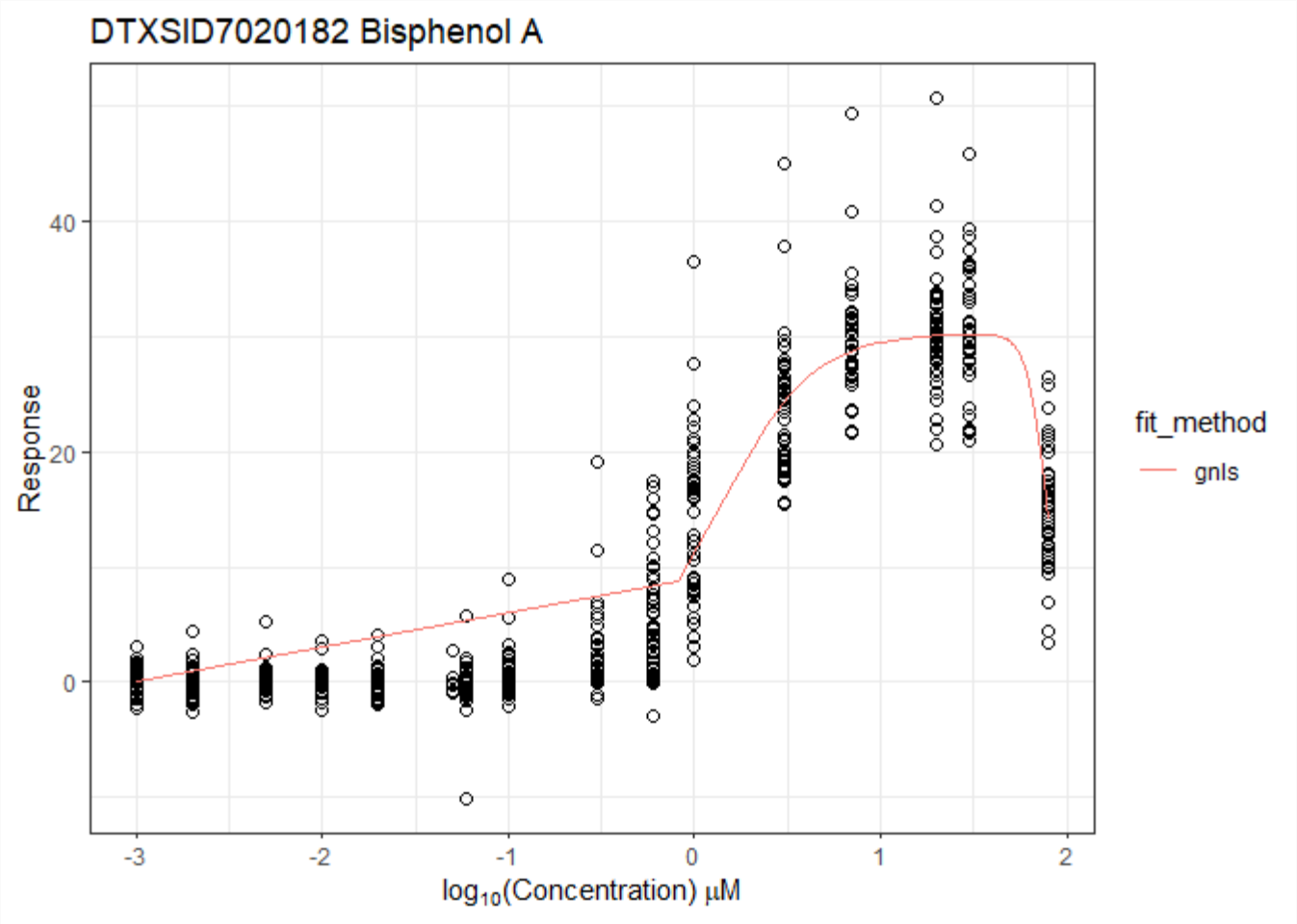


Figure 3: *Concentration-response data and the winning model fit for Bisphenol A using the* `concRespPlot2` *function. Concentrations (x-axis) are displayed in \(log_{10}\) units.*

Example 3: Concentration-response Modeling for `tcpl`-like data without a database connection

The `tcplLite` functionality was deprecated with the updates to `tcpl` and development of `tcplfit2`, since `tcplfit2` allows one to perform curve-fitting and hit-calling independent of a database connection. This example demonstrates how to perform an analysis analogous to `tcplLite` with `tcplfit2`. More information on the ToxCast program can be found at https://www.epa.gov/chemical-research/toxicity-forecasting. A detailed explanation of processing levels can be found within `tcpl`'s Data Processing vignette.

In this example, the example input data comes from the ACEA_AR assay. Data from the assay component ACEA_AR_agonist_80hr assumes the response changes in the positive direction relative to DMSO (neutral control & baseline activity) for this curve-fitting analysis. Using an 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 the heterogeneity in assay data reporting, source data often must go through pre-processing steps to transform into a uniform data format, namely Level 0 data.

Source Data Formatting

To run standalone `tcplfit2` fitting, without the need for a MySQL database connection like `invitrodb`, the user will need to step-through/replicate multiple levels of processing (i.e. through to Level 3). The below table is identical to the multi-concentration level 0 data (mc0) table one would see in `invitrodb` and is compatible with `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 response 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")
data.table::setDTthreads(2)
DT::datatable(head(dat[wllt=='t',]),rownames= FALSE, options = list(scrollX =
T))
```

Show 10 entries Search:

m0id	spid	acid	apid
519762672	TP0001364A01	1829	Experiment.ID:1502051323HT1_A113641_AP01_
519762768	TP0001364A02	1829	Experiment.ID:1502051323HT1_A113641_AP01_

519762864	TP0001364A03	1829	Experiment.ID:1502051323HT1_A113641_AP01_
519762960	TP0001364A04	1829	Experiment.ID:1502051323HT1_A113641_AP01_
519763056	TP0001364A05	1829	Experiment.ID:1502051323HT1_A113641_AP01_
519763152	TP0001364A06	1829	Experiment.ID:1502051323HT1_A113641_AP01_

Showing 1 to 6 of 6 entries Previous 1 Next

The first step is to establish the concentration index, and corresponds to Level 1 in `tcpl`. Concentration indices are integer values ranking N distinct concentrations from 1 to N, which correspond to the lowest and highest concentration groups, respectively. This index can be used to calculate the baseline median absolute deviation (BMAD) for an assay.

```
# Order by the following columns
setkeyv(dat, c('acid', 'srcf', 'apid', 'coli', 'rowi', 'spid', 'conc'))
```

Adjustments

The second step is perform any necessary data adjustments, and corresponds to Level 2 in `tcpl`. Generally, if the raw response values (`rval`) need to undergo logarithmic transformation or some other transformation, then those adjustment occur in this step. Transformed response values are referred to as corrected values and are stored in the `cval` field/variable. Here, the raw response values do not require transformation and are identical to the corrected values (`cval`). Samples with poor well quality (`wllq = 0`) and/or missing response values are removed from the overall dataset to consider in the concentration-response series.

```
# 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),]
```

Normalization

The third step normalizes and zero-centers before model fitting, and corresponds to Level 3 in `tcpl`. Our example dataset has both neutral and negative controls available, and the code below demonstrates how to normalize responses to a control in this scenario. However, given experimental designs vary from assay to assay, this process also varies across assays. Thus, the steps shown in this example may not apply to other assays and should only be considered applicable for this example data set. In other applications/scenarios, such as when neutral control or positive/negative controls are not available, the user should normalize responses that best accounts for their experimental design and data. Provided below is a list of normalizing methods used in `tcpl` for reference.

That being said, for this example, the normalized responses (`resp`) are calculated as a percent of control, i.e. the ratio of differences. The numerator is the difference between the corrected (`cval`) and baseline (`bval`) values and denominator is the difference between the positive/negative control (`pval`) and baseline (`bval`) values.

\[\% \space control = \frac{cval - bval}{pval - bval} \] The table blow provides methods for calculating `bval` and `pval`. For more on the data normalization step, refer to the Data Normalization section of `tcpl`'s Data Processing Vignette.

```
## Check out all Level 3 methods for normalization in tcpl
DT::datatable(tcpl::tcplMthdList(3),rownames= FALSE, options = list(scrollX = T))
```

Show 10 entries Search:

mc3_mthd_id	mc3_mthd	desc
1	none	Set the corrected response value (cval) as the normalized response value (resp); cval = resp. No

		additional mc3 methods needed for endpoint-specific normalization.
2	bval.apid.lowconc.med	Calculate the baseline value (bval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) for test compound wells (wllt = t) with a concentration index (cndx) of 1 or 2.
3	pval.apid.medpcbyconc.max	Calculate the positive control value (pval) as the plate-wise maximum, by assay plate ID (apid), of the medians of the corrected values (cval) for gain-of-signal single- or multiple-concentration negative control wells (wllt = m or o) by apid, well type, and concentration.
4	pval.apid.medpcbyconc.min	Calculate the positive control value (pval) as the plate-wise minimum, by assay plate ID (apid), of the medians of corrected value (cval) of gain-of-signal single- or multiple-concentration positive control wells (wllt = p or c) by apid, well type, and concentration.
5	resp.pc	Calculate the normalized response (resp) as a percent of control, i.e. the ratio of the difference between the corrected (cval) and baseline (bval) values divided the difference between the positive control (pval) and baseline (bval) values multiplied by 100; resp = (cval-bval)/(pval-bval)*100.

6	resp.multneg1	Multiply the normalized response value (resp) by -1; -1*resp.
7	resp.log2	Transform the response values to log-scale (base 2).
8	resp.mult25	Multiply the normalized response value (resp) by 25; 25*resp.
9	resp.fc	Calculate the normalized response (resp) as the fold change, i.e. the ratio of the corrected (cval) and baseline (bval) values; resp = cval/bal.
11	bval.apid.nwlls.med	Calculate the baseline value (bval) as the plate-wise median, by assay plate ID (apid), of the corrected values (cval) for neutral control wells (wllt = n).

Showing 1 to 10 of 47 entries Previous 1 2 3 4 5 Next

```
# calculate bval
dat[, bval := median(cval[wllt == "n"]), by = list(apid)]
# calculate pval
dat[, pval := median(cval[wllt %in% c("m","o")], na.rm = TRUE), by = list(apid,
wllt, conc)]
dat[, pval := min(pval, na.rm = TRUE), by = list(apid)]

# Calculate normalized responses
dat[, resp := ((cval - bval)/(pval - bval) * 100)]
```

Before model fitting, we need to determine the median absolute deviation around baseline (`BMAD`) and baseline variability (`onesd`), which are later used for cutoff and benchmark response (`BMR`) calculations, respectively. This is part of Level 4 processing in `tcpl`. `tcplfit2` assumes no effect occurs in the two lowest concentrations. Thus, we consider

test wells in the two lowest concentrations as our baseline to calculate `BMAD` and `onesd`.

`BMAD` can be calculated as the median absolute deviation of the data in control wells too. Check out other methods of determining `BMAD` and `onesd` used in `tcpl`.

```
# Check out all Level 4 methods for normalization in tcpl
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	Calculate the baseline median absolute value (bmad) as the median absolute deviation of normalized response values (rep) for test compound wells (wllt = t) with concentration index (cndx) equal to 1 or 2.
2	bmad.aeid.lowconc.nwells	Calculate the baseline median absolute value (bmad) as the median absolute deviation of normalized response values (resp) for neutral control wells (wllt = n).
3	onesd.aeid.lowconc.twells	Calculate one standard deviation of the normalized response for test compound wells (wllt = t) with a concentration index (cndx) of 1 or 2; onesd = sqrt(sum((resp - mean resp)^2)/sample size - 1). Used to establish BMR and therefore required for tcplfit2 processing.
4	bidirectional.false	Limits bidirectional fitting and processes data in positive analysis direction only. Use for gain-of-signal or

		inverted data.
5	bmad5.onesd16.static	Replace baseline median absolute deviation (bmad) with 5 and one standard deviation (osd) of the normalized response for test compound wells (wllt = t) with a concentration index (cndx) of 1 or 2 with 16. Typically used for binary data where values would otherwise be 0; non-zero values are required for tcplfit2 processing.
6	no.unbounded.models	Select only the models that are bounded

Showing 1 to 6 of 6 entries Previous 1 Next

If the user's dataset contains data from multiple assays (`aeid`), `BMAD` and `onesd` should be calculated per assay/ID. The example data set only contains data from one assay, so we can calculate `BMAD` and `onesd` on the whole dataset.

```
bmad <- mad(dat[cndx %in% c(1, 2) & wllt == "t", resp])
onesd <- sd(dat[cndx %in% c(1, 2) & wllt == "t", resp])
```

Dose-Response Curve Fitting

Once the data adjustments and normalization steps are complete, model fitting then hitcalling can be done, similar to what was shown in Example 2. Dose-Response Curve Fitting corresponds to Level 4 in `tcpl`. This is where `tcplfit2` is used to fit all available models within `tcpl`.

```
#do tcplfit2 fitting
myfun <- function(y) {
  res <- tcplfit2::tcplfit2_core(y$conc,
                                 y$resp,
```

```
                            cutoff = 3*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 on the full data set, `dat`, can take 7-10 minutes to run.** Hence the following code chunk provides an example subset of data to demonstrate curve fitting. The example 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 models are fit to the data, `tcplhit2_core` is used to perform hitcalling and corresponds to Level 5 in `tcpl`. The output of `tcplfit2_core`, i.e. Level 4 data, may be fed directly to the `tcplhit2_core` function. The results are then pivoted and the resulting data table is displayed below.

```
myfun2 <- function(y) {
  res <- tcplfit2::tcplhit2_core(params = y$params[[1]],
                                 conc = y$conc,
```

```
                                    resp = y$resp,
                                    cutoff = 3*bmad,
                                    onesd = onesd
                                    )
  list(list(res))
}

# continue 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_
1	TP0001366A03	0	49.28306384522267	gnls	0.281901000
2	TP0001366A04	0	49.28306384522267	poly1	0.285116813
3	TP0001366A05	0	49.28306384522267	gnls	0.230579663
4	TP0001366A06	0	49.28306384522267	poly1	0.130144380
5	TP0001366A07	0	49.28306384522267	gnls	0.254805475
6	TP0001366A08	0	49.28306384522267	gnls	0.440022924

Showing 1 to 6 of 6 entries Previous 1 Next

Please note, hitcalling can also be done with the full data set, `dat`, but here we only demonstrate hitcalling with the example dataset model fitting was performed on.

The resulting output from the previous code chunk is the same format as the `result_table`

table in Example 2. Thus, one can use the `concRespPlot2` function, as done previously in Example 2, to plot the results. The next code chunk demonstrates how to visualize the Example 3 fit results.

```
# allocate a place-holder object
  plt_list <- NULL
# plot results using `concRespPlot`
  for(i in 1:nrow(res_wide)){
    plt_list[[i]] <- concRespPlot2(res_wide[i,])
  }
# compile and display winning model plots for concentration-response series
  grid.arrange(grobs=plt_list,ncol=2)
```

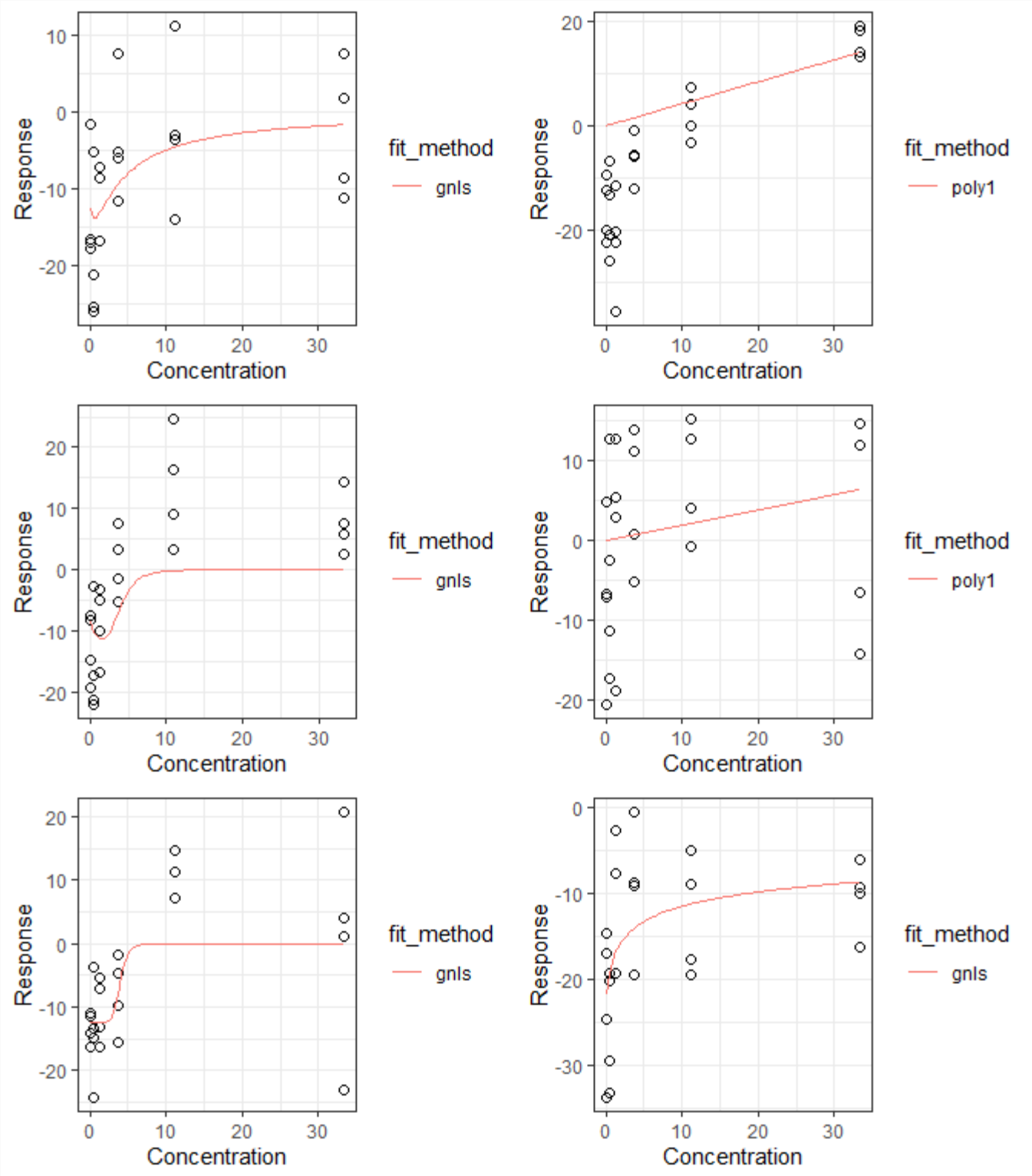


Figure 4: *Each sub-plot displays the winning curve for a given concentration-response series in the* `subdat` *dataset.*

Example 4: Bounding the Benchmark Dose (BMD)

Occasionally, the estimated benchmark dose (BMD) can occur outside the experimental tested concentration range, e.g. the BMD may be greater than the maximum tested

concentration in the data. In these cases, `tcplhit2_core` and `concRespCore` provide options for users to “censor” the estimated BMD. This can be done using the `bmd_low_bnd` and `bmd_up_bnd` arguments.

`bmd_low_bnd` and `bmd_up_bnd` are multipliers applied to the minimum or maximum tested concentrations (i.e. reference doses), respectively, to provide an lower and upper boundaries for BMD estimates (i.e. threshold doses). This section demonstrates how to “censor” BMD estimates using the provided arguments in the `concRespCore` and `tcplhit2_core` functions.

Imposing Lower BMD Bounds

First, consider a situation when the estimated BMD is less than the lowest tested concentration. This occurs when the experimental concentrations do not go low enough to capture the transition between the baseline response and the minimum response considered adverse occurring around the benchmark response (BMR). Failure to capture the response behavior in the low-dose region of the experimental design may indicate the data is not suitable for estimating a reliable point-of-departure, and should be flagged.

In the following code chunk, we use the `mc3` dataset with some minor modifications, to demonstrate this case. Here, we take one of the concentration-response series and remove dose groups less than \(0.41\). Removing the lower dose groups simulates the scenario where there is a lack of data in low-dose region and causes the BMD estimate to be less than the lowest concentration remaining in the data.

```
# We'll use data from mc3 in this section
data("mc3")
```

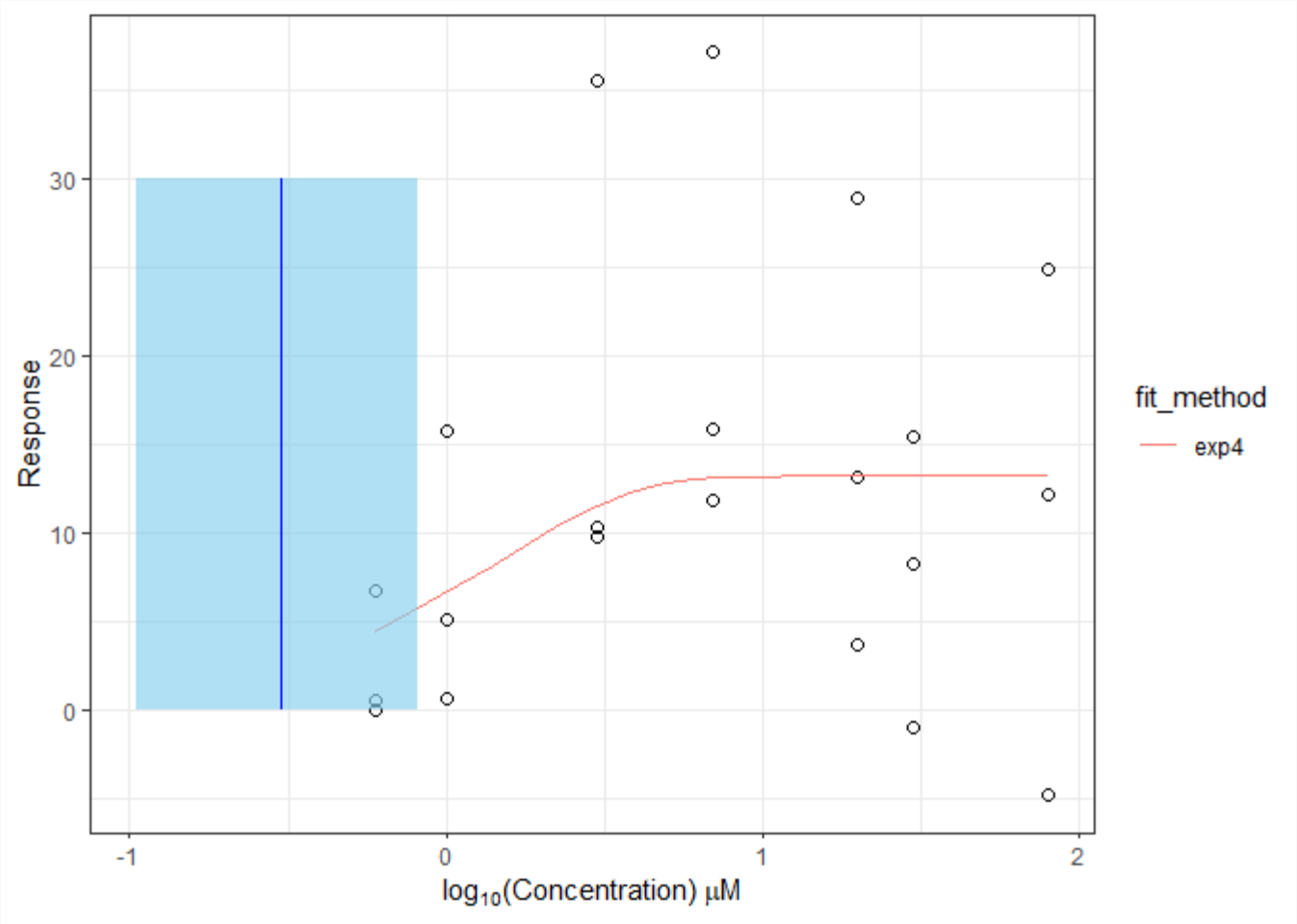


Figure 5: *This plot shows the winning curve, BMD estimation (represented by the solid blue line) and the estimated BMD confidence interval (represented by the light blue bar).*

```
# function results
res_low['Min. Conc.'] <- min(conc2)
res_low['Name'] <- name
res_low[1, c("Min. Conc.", "bmd", "bmdl", "bmdu")] <- round(res_low[1, c("Min.
Conc.", "bmd", "bmdl", "bmdu")], 3)
DT::datatable(res_low[1, c("Name","Min. Conc.", "bmd", "bmdl",
"bmdu")],rownames = FALSE)
```

Show 10 entries Search:

Name	Min. Conc.	bmd	bmdl	bmdu
Chlorothalonil	0.6	0.302	0.105	0.815

Showing 1 to 1 of 1 entries Previous 1 Next

The lowest concentration (i.e. reference dose) in the data is 0.6 but the estimated BMD from the hit-calling results is 0.302, which is lower. Users may allow the estimated BMD to be lower than the reference dose while restricting it to be no lower than a threshold dose set by using the argument `bmd_low_bnd`.

If BMD should be no lower than 80% of the lowest tested concentration, then `bmd_low_bnd = 0.8` can be used to set a threshold dose. Here, that results in a threshold dose of \(0.48\). If the estimated BMD is less than the threshold dose (like in this example), it will be "censored" to the threshold dose. Similarly, the confidence interval will also be shifted right by a distance equal to the difference between the estimated BMD and the threshold dose. Figure 7 provides a visual representation of the lower boundary censoring. The valid input range for `bmd_low_bnd` is between 0 and 1, excluding 0, (\(0 < \text{bmd_low_bnd} \leq 1\)).

```
# using the argument to set a lower bound for BMD
res_low2 <- concRespCore(row_low,fitmodels = c("cnst", "hill", "gnls", "poly1",
"poly2",
                                               "pow", "exp2", "exp3", "exp4",
"exp5"),
                         conthits = T, aicc = F, bidirectional=F, bmd_low_bnd =
0.8)
```

```
# print out the new results
# include previous results side by side for comparison
res_low2['Min. Conc.'] <- min(conc2)
res_low2['Name'] <- paste(name, "after `censoring`", sep = "-")
res_low['Name'] <- paste(name, "before `censoring`", sep = "-")
res_low2[1, c("Min. Conc.", "bmd", "bmdl", "bmdu")] <- round(res_low2[1,
c("Min. Conc.", "bmd", "bmdl", "bmdu")], 3)

output_low <- rbind(res_low[1, c('Name', "Min. Conc.", "bmd", "bmdl", "bmdu")],
                    res_low2[1, c('Name', "Min. Conc.", "bmd", "bmdl",
"bmdu")])
DT::datatable(output_low,rownames = FALSE)
```

Show 10 entries Search:

Name	Min. Conc.	bmd	bmdl	bmdu

Chlorothalonil-before `censoring`	0.6	0.302	0.105	0.815
Chlorothalonil-after `censoring`	0.6	0.48	0.283	0.993

Showing 1 to 2 of 2 entries Previous 1 Next

```
# generate some concentration for the fitted curve
logc_plot <- seq(from=-3,to=2,by=0.05)
```

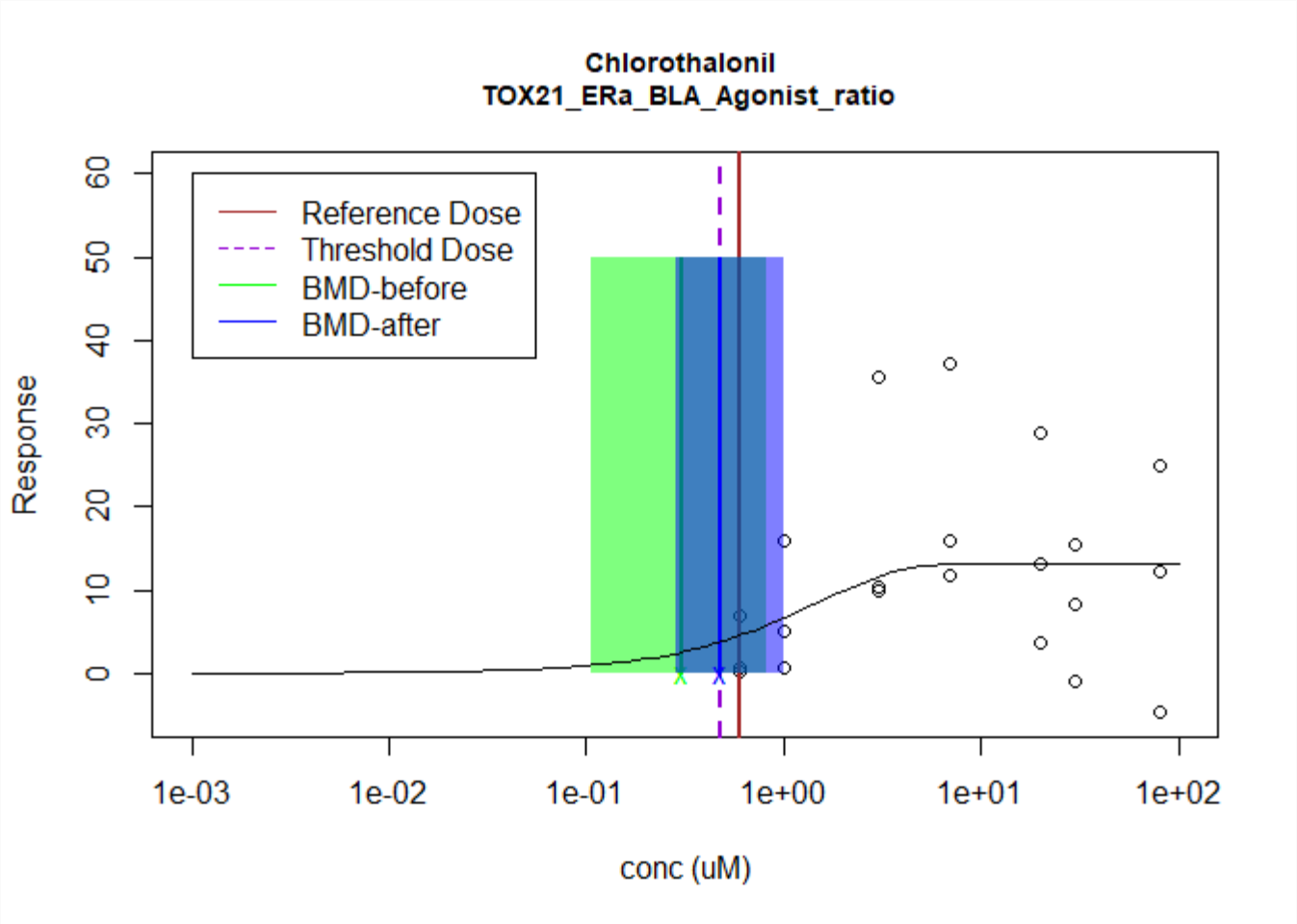


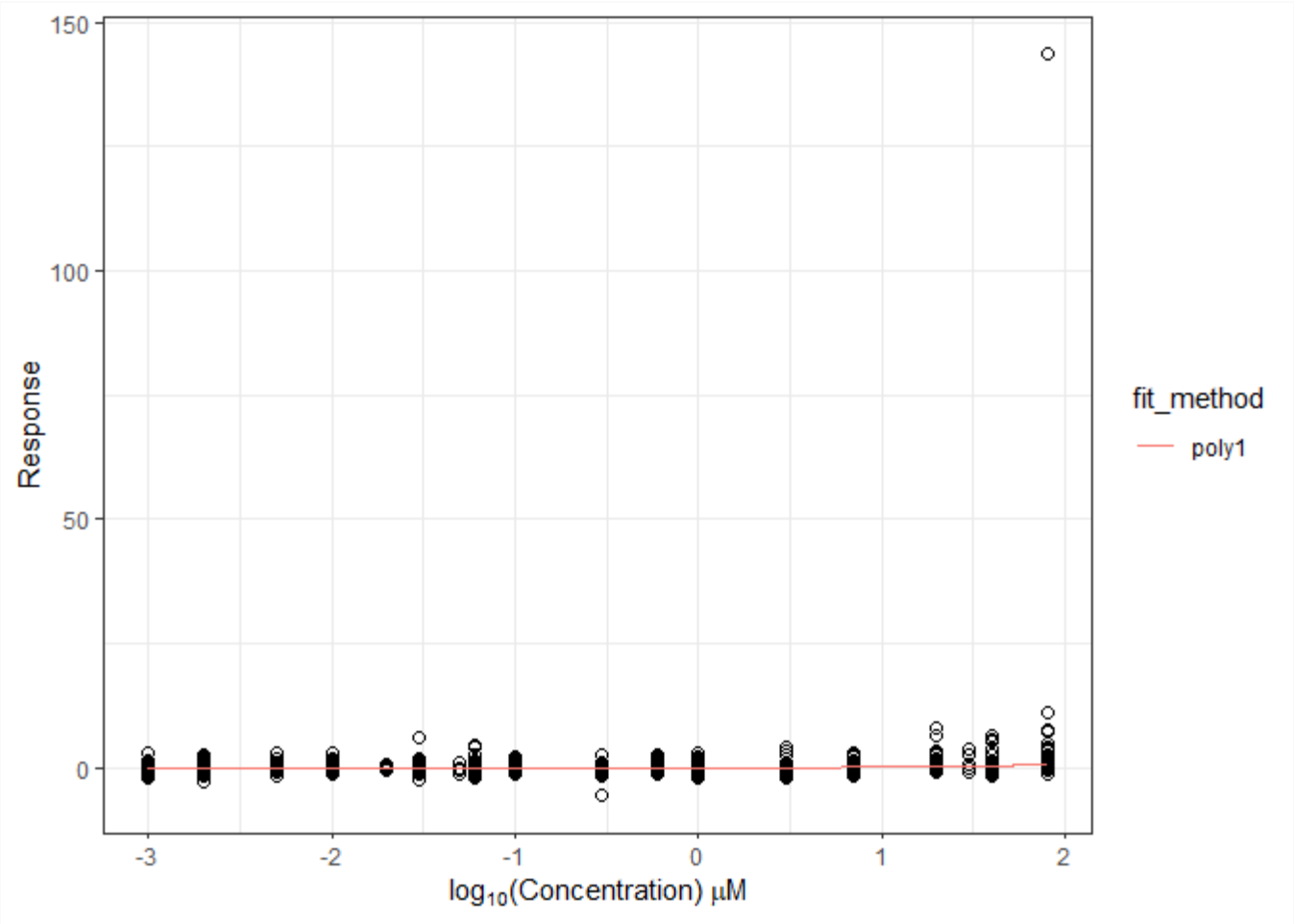
***Figure 6**: This plot shows the estimated BMD and confidence interval before and after "censoring". The solid green line and "X" mark the estimated BMD before "censoring", and the green shaded region represents the estimated confidence interval. The solid blue line and "X" mark the BMD after "censoring", and the blue shaded region represents the "censored" confidence interval. The solid brown line represents the minimum tested concentration, and the dashed dark violet line represents the threshold dose set by `bmd_low_bnd`. Here, the estimated BMD and the confidence interval were shifted right such*

that the BMD was “censored” to the threshold value represented by the overlap between the blue “X” and dashed dark violet line.

Imposing Upper BMD Bounds

In the next scenario, the estimated BMD is much larger than the maximum tested concentration. Here, `bmd_up_bnd` is used to set an upper bound on extremely large BMD estimates.

```
# load example data
spid <- unique(mc3$spid)[26]
ex_df <- mc3[is.element(mc3$spid,spid),]

# The data file has stored concentration in log10 form, so fix that
conc <- 10**ex_df$logc # back-transforming concentrations on log10 scale
resp <- ex_df$resp

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

# create the row object
row_up <- 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_up <- concRespCore(row_up,fitmodels = c("cnst", "hill", "gnls", "poly1",
"poly2",
                                            "pow", "exp2", "exp3", "exp4",
"exp5"),
                       conthits = T, aicc = F, bidirectional=F)

concRespPlot2(res_up, log_conc = T)
```




```
# max conc.
res_up['Max Conc.'] <- max(conc)
res_up['Name'] <- name
res_up[1, c("Max Conc.", "bmd", "bmdl", "bmdu")] <- round(res_up[1, c("Max
Conc.", "bmd", "bmdl", "bmdu")], 3)
# function results
DT::datatable(res_up[1, c('Name','Max Conc.', "bmd", "bmdl", "bmdu")],rownames
= FALSE)
```

Show 10 entries Search:

Name	Max Conc.	bmd	bmdl	bmdu
Pyrene	80	299.927	217.343	475.064

Showing 1 to 1 of 1 entries Previous 1 Next

The estimated BMD 299.927 is greater than the maximum tested concentration

(i.e. reference dose), which is 80. As with the `bmd_low_bnd`, users may allow the BMD to be greater than the maximum tested concentration but no greater than a threshold dose set using `bmd_up_bnd`.

Suppose, it is desired that the estimated BMD not be larger than 2 times the maximum tested concentration. Here, `bmd_up_bnd = 2` can set the upper threshold dose to \(160\). If the estimated BMD is greater than the upper threshold dose (like in this example), it will be "censored" to the threshold dose, and its confidence interval will be shifted left. Figure 9 provides a visual representation of upper boundary censoring. The valid input range for `bmd_up_bnd` is any value greater than or equal to 1 (\(\text{bmd_up_bnd} \geq 1\)).

```
# using bmd_up_bnd = 2
res_up2 <- concRespCore(row_up,fitmodels = c("cnst", "hill", "gnls", "poly1",
"poly2",
                                             "pow", "exp2", "exp3", "exp4",
"exp5"),
                        conthits = T, aicc = F, bidirectional=F, bmd_up_bnd =
2)
```

```
# print out the new results
# include previous results side by side for comparison
res_up2['Max Conc.'] <- max(conc)
res_up2['Name'] <- paste(name, "after `censoring`", sep = "-")
res_up['Name'] <- paste(name, "before `censoring`", sep = "-")
res_up2[1, c("Max Conc.", "bmd", "bmdl", "bmdu")] <- round(res_up2[1, c("Max
Conc.", "bmd", "bmdl", "bmdu")], 3)

output_up <- rbind(res_up[1, c('Name', "Max Conc.", "bmd", "bmdl", "bmdu")],
                   res_up2[1, c('Name', "Max Conc.", "bmd", "bmdl", "bmdu")])
DT::datatable(output_up,rownames = FALSE)
```

Show 10 entries Search:

Name	Max Conc.	bmd	bmdl	bmdu
Pyrene-before `censoring`	80	299.927	217.343	475.064
Pyrene-after `censoring`	80	160	77.416	335.136

Showing 1 to 2 of 2 entries Previous 1 Next

```
# generate some concentration for the fitting curve
logc_plot <- seq(from=-3,to=2,by=0.05)
conc_plot <- 10**logc_plot
```

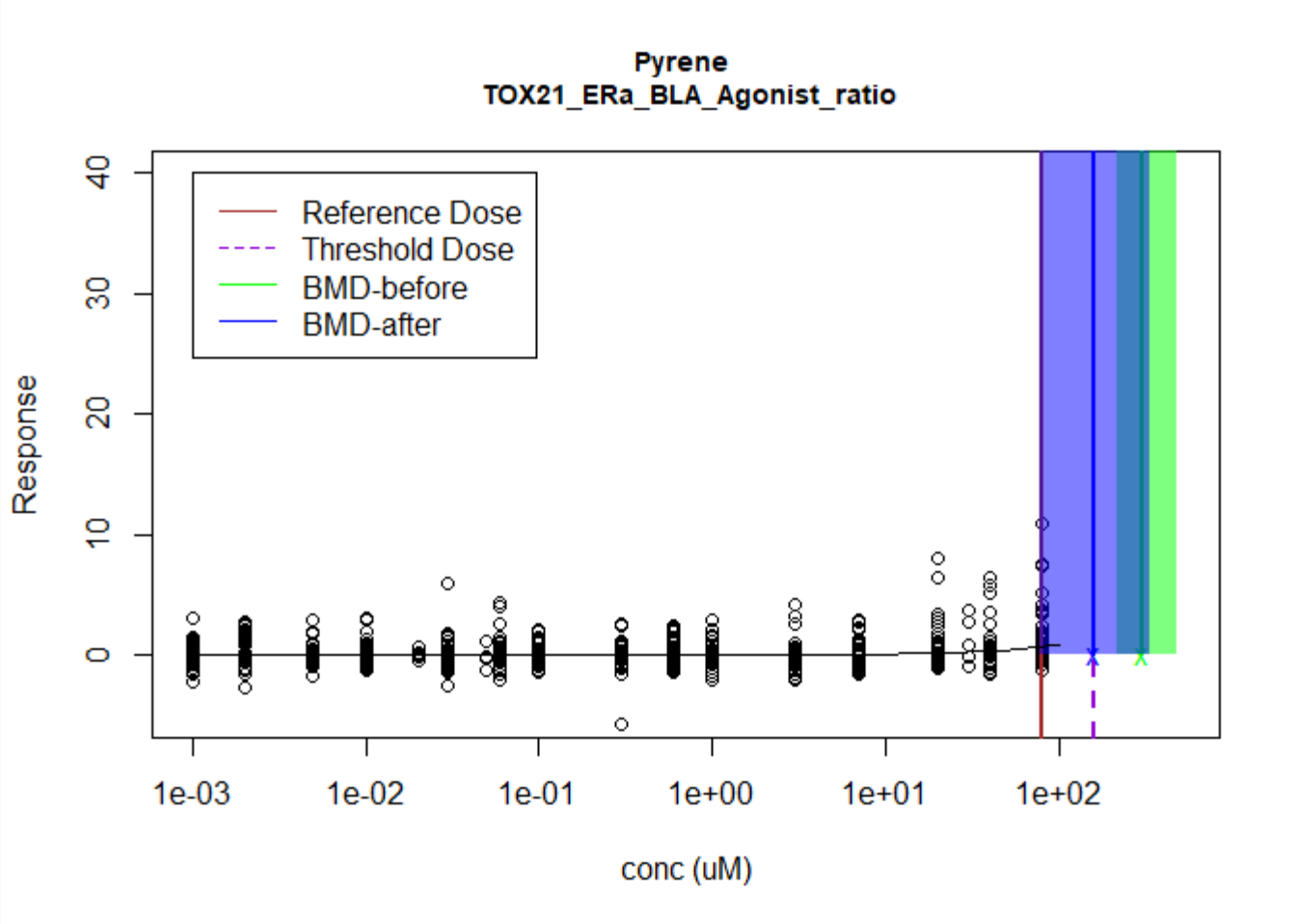


***Figure 7**: This plot shows the estimated BMD and confidence interval before and after "censoring". The green line and "X" mark the estimated BMD before "censoring" and the green shaded region represents the estimated confidence interval. The solid blue line and "X" mark the "censored" BMD, and the blue shaded region represents the "censored" confidence interval. The solid brown line represents the maximum tested concentration, and the dashed dark violet line represents the threshold dose set by `bmd_up_bnd`. Here, the estimated BMD and the confidence interval were shifted left such that the BMD was "censored" to the threshold value represented by the overlap between the blue "X" and dashed dark violet line.*

Bounding BMDs with `tcplhit2_core`

The previous two examples provided for BMD censoring use the `concRespCore` function. However, the `bmd_low_bnd` and `bmd_low_bnd` are arguments originating from `tcplhit2_core` function, which is utilized within the `concRespCore` function. Thus, for users that perform dose-response modeling and hit-calling utilizing the `tcplfit2_core` and `tcplhit2_core` separately can do the same BMD "censoring". Regardless of whether a user utilizes the `bmd_low_bnd` and `bmd_low_bnd` arguments in the `concRespCore` or `tcplhit2_core` function the results should be identical. The code provided below shows how to replicate the results from the lower bound example example using `tcplhit2_core` as an alternative.

```
# using the same data, fit curves
param <- tcplfit2_core(conc2, resp2, cutoff = cutoff)
hit_res <- tcplhit2_core(param, conc2, resp2, cutoff = cutoff, onesd = onesd,
                         bmd_low_bnd = 0.8)
```

```
# adding the result from tcplhit2_core to the output table for comparison
hit_res["Name"]<-  paste("Chlorothalonil", "tcplhit2_core", sep = "-")
hit_res['Min. Conc.'] <- min(conc2)
hit_res[1, c("Min. Conc.", "bmd", "bmdl", "bmdu")] <- round(hit_res[1, c("Min.
Conc.", "bmd", "bmdl", "bmdu")], 3)

output_low <- rbind(output_low,
                    hit_res[1, c('Name', "Min. Conc.", "bmd", "bmdl", "bmdu")])
DT::datatable(output_low,rownames = FALSE)
```

Show 10 entries Search:

Name	Min. Conc.	bmd	bmdl	bmdu
Chlorothalonil-before `censoring`	0.6	0.302	0.105	0.815
Chlorothalonil-after `censoring`	0.6	0.48	0.283	0.993
Chlorothalonil-tcplhit2_core	0.6	0.48	0.283	0.993

Showing 1 to 3 of 3 entries Previous 1 Next

Impacts if BMD is between the “Threshold Value” and “Reference Dose”

If the estimated BMD falls between the reference dose and the threshold dose, e.g. lowest tested dose and lower threshold, the estimated BMD will remain unchanged. For demonstration purposes, the lower bound example is used, but the same principle applies to the upper bound case.

The same data from the lower bound example is used along with a smaller `bmd_low_bnd` value to obtain a lower threshold dose. Here, the estimated BMD is acceptable as long as it is no less than 40% (two-fifths) of the minimum tested concentration (i.e. reference dose). The estimated BMD is 0.302, which is between the lowest tested dose, 0.6, and the new threshold dose \(0.24\). Thus, the BMD estimate and its confidence interval will remain unchanged.

```
res_low3 <- concRespCore(row_low,fitmodels = c("cnst", "hill", "gnls", "poly1",
"poly2",
                                               "pow", "exp2", "exp3", "exp4",
"exp5"),
                         conthits = T, aicc = F, bidirectional=F, bmd_low_bnd =
0.4)
```

```
# print out the new results
# add to previous results for comparison
res_low3['Min. Conc.'] <- min(conc2)
res_low3['Name'] <- paste("Chlorothalonil", "after `censoring` (two fifths)",
sep = "-")
res_low3[1, c("Min. Conc.", "bmd", "bmdl", "bmdu")] <- round(res_low3[1,
c("Min. Conc.", "bmd", "bmdl", "bmdu")], 3)

output_low <- rbind(output_low[-3, ],
                    res_low3[1, c('Name', "Min. Conc.", "bmd", "bmdl",
"bmdu")])
DT::datatable(output_low,rownames = FALSE)
```

Show 10 entries Search:

Name	Min. Conc.	bmd	bmdl	bmdu
Chlorothalonil-before `censoring`	0.6	0.302	0.105	0.815
Chlorothalonil-after `censoring`	0.6	0.48	0.283	0.993
Chlorothalonil-after `censoring` (two fifths)	0.6	0.302	0.105	0.815

Showing 1 to 3 of 3 entries Previous 1 Next

```
# initiate the plot
plot(conc2,resp2,xlab="conc
(uM)",ylab="Response",xlim=c(0.001,100),ylim=c(-5,60)
```

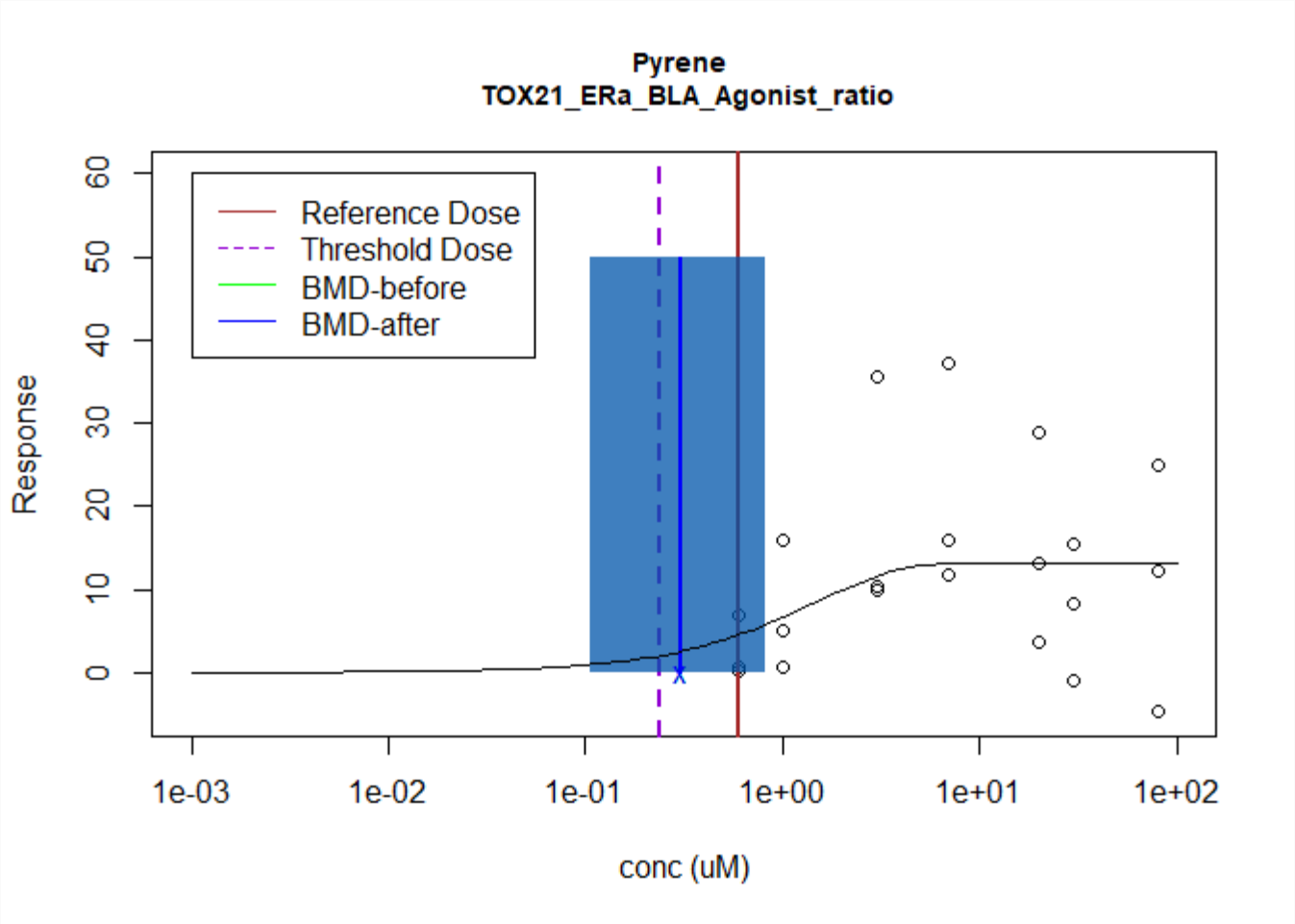


***Figure 8**: This plot shows the estimated BMD and the confidence interval before and after "censoring". The dashed dark violet line represents the threshold dose and the solid brown line represents the minimum tested concentration, which are at 0.24 and 0.6, respectively. The estimated BMD of 0.302 falls between the threshold and reference doses, which leaves*

the BMD and confidence intervals unchanged. Here, the estimated BMD and "censored" BMD are the same. Thus, the green and blue lines and "X"s representing the estimated BMD before and after "censoring", respectively, as well as their confidence intervals indicated by the shaded regions completely overlap.

Appendix

Additonal Plotting Options

Example 1 and 2 illustrated two plotting functions available in `tcplfit2` based on `ggplot2` plotting grammar. This appendix will show two other plotting options available in `tcplfit2`, which use base R plotting, namely the `do.plot` argument in `concRespCore` and the `concRespPlot` function.

The `concRespPlot` function and the `do.plot` argument in `concRespCore` provide plots similar to Figure 1 and 2, respectively. The `do.plot` argument returns a plot of all curve fits of a chemical, and `concRespCore` returns a plot of the winning curve with the hitcall results.

The input data used for the demonstration contains 6 signatures for one chemical in a transcriptomics data set. Each signature is a different assay endpoint, thus one row in the data represents a given chemical and signature pair (assay endpoint). This data set is a sample from the signature scoring method that provides the cutoff, one standard deviation, and the concentration-response data.

```
# 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)){
```

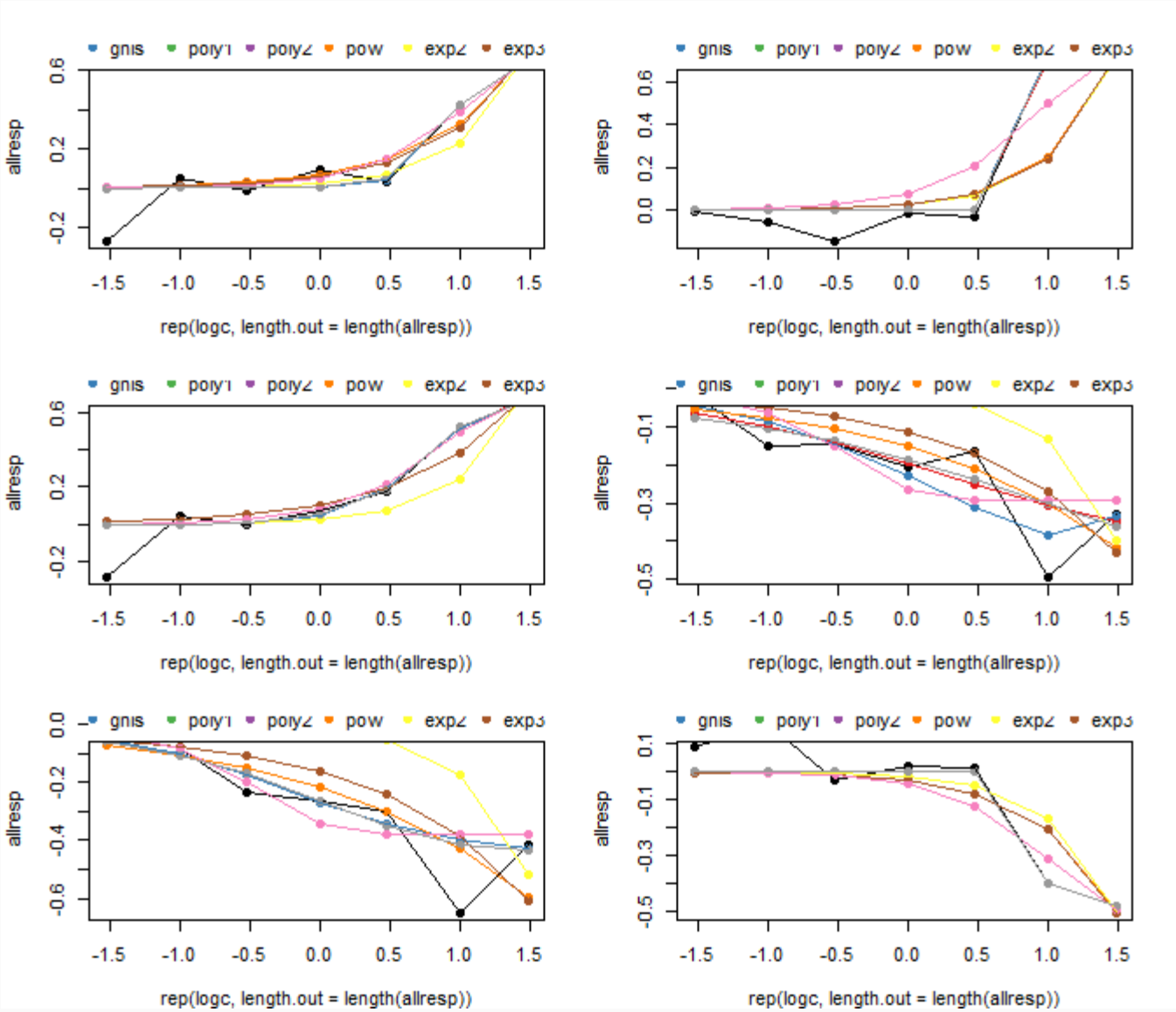


Figure 9: *This figure provides several example plots generated using the argument* `do.plot=TRUE` *in the* `concRespCore` *function. Each plot displays data for a single row of data in the* `signatures` *dataset, and like Figure 1 provides all model fits for a given response.*

```
# 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`
for(i in 1:nrow(res)){
  concRespPlot(res[i,],ymin=-1,ymax=1)
}
```

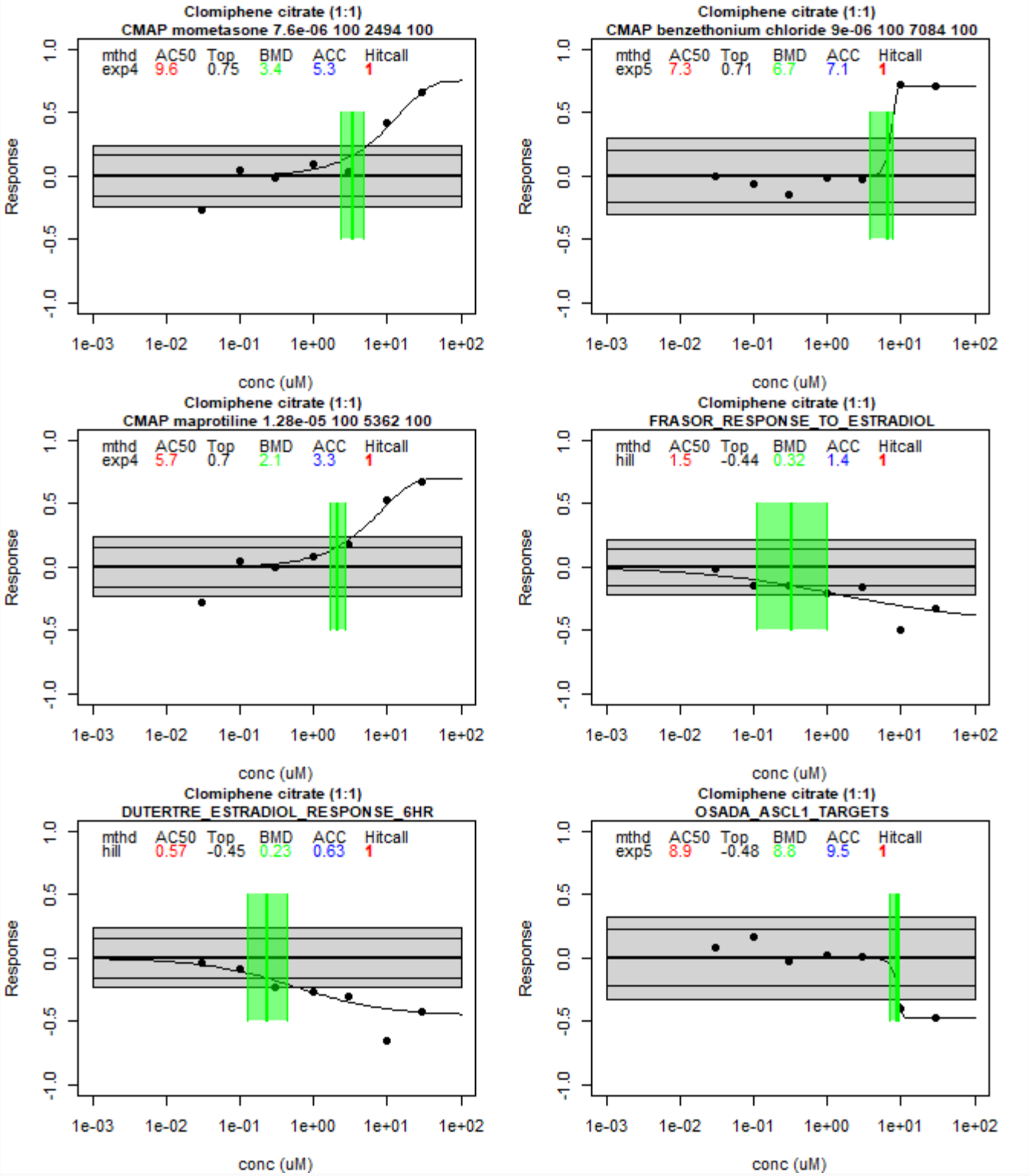


Figure 10: *Each figure shows curve-fit results for a randomly selected set of responses in the* `mc3` *data. For each plot, the title contains the chemical name and assay ID. Summary statistics from the curve-fit results – including the winning model, AC50, top, BMD, ACC, and hitcall – are displayed at the top of the plot. Black dots represent observed responses, and the winning model fit is displayed as a solid black curve. The estimated BMD is displayed with a solid green vertical line, and confidence interval for the BMD are represented with the*

solid green lines bounding the green shaded region (i.e., lower and upper BMD confidence limits - BMDL and BMDU, respectively). The black horizontal lines bounding the grey shaded region indicates the estimated baseline noise and is centered around the x-axis (i.e. y = 0).

Plotting with tcplfit2

Center for Computational Toxicology and Exposure, US EPA

Introduction

The `tcplfit2` package performs concentration-response modeling and includes several functionalities for visualizing the results. These `ggplot`-based visualization functions are `plot_allcurves` and `concRespPlot2`. The following sections will demonstrate how to utilize the `tcplfit2`'s `ggplot`-based functions as well as further customize plots with additional `ggplot2`-layers.

For this vignette, the `signature` dataset available in the `tcplfit2` package will be used to demonstrate utility of the plotting functions. The `signatures` dataset contains 6 transcriptional signatures (assay endpoints) for one chemical. Each row in the data represents a chemical-assay endpoint pair with a cutoff, baseline standard deviation, and experimental concentration-response data. For demonstration purposes, only the first row will be used.

```
# Load the example data set
data("signatures")
# using the first row of signatures data as an example
signatures[1,]
```

```
##      sample_id        dtxsid                      name
## 1 TP0001651A05 DTXSID8020337 Clomiphene citrate (1:1)
##                             signature    cutoff     onesd
## 1 CMAP mometasone 7.6e-06 100 2494 100 0.2393037 0.1196519
##                      conc
## 1 0.03|0.1|0.3|1|10|3|30
##
resp
## 1
-0.267223789187984|0.0446549190032419|-0.0150009098668537|0.097550989137631|0.4201
```

Plotting All Models From `tcplfit2_core`

Users performing concentration-response modeling may want to compare the resulting fits from all models. The `plot_allcurves` function enables users to automatically generate this visualization with the output from the `tcplfit2_core` function. Note, to utilize `plot_allcurves`, `tcplfit2_core` must be run separately to obtain the necessary input. The resulting figure allows one to evaluate general behaviors and qualities of the resulting curve fits. Furthermore, some curves may fail to fit the observed data. In these cases, failed models are excluded from the plot and a warning message is provided, such that the user will know which models are capable of describing the data. Lastly, if a user wants to visualize their data with the concentrations on the `log-10` scale they can set the `log_conc` argument to `TRUE`.

The following code demonstrates how to obtain the curve-fitting results with `tcplfi2_core` and generate a visualization with `plot_allcurves`:

```
# using the first row of signature as an example
conc=as.numeric(str_split(signatures[1,"conc"],"\\|")[[1]])
resp=as.numeric(str_split(signatures[1,"resp"],"\\|")[[1]])
cutoff=signatures[1,"cutoff"]

# run curve fitting
output <- tcplfit2_core(conc, resp, cutoff)
```

```
# show the structure of the output
summary(output)
```

```
##            Length Class  Mode
## cnst        5     -none- list
## hill       17     -none- list
## gnls       22     -none- list
## poly1      13     -none- list
## poly2      15     -none- list
## pow        15     -none- list
## exp2       15     -none- list
## exp3       17     -none- list
## exp4       15     -none- list
## exp5       17     -none- list
## modelnames 10     -none- character
## errfun      1     -none- character
```

```
# get plots in normal and in log-10 concentration scale
basic <- plot_allcurves(output, conc, resp)
basic_log <- plot_allcurves(output, conc, resp, log_conc = T)
grid.arrange(basic, basic_log)
```

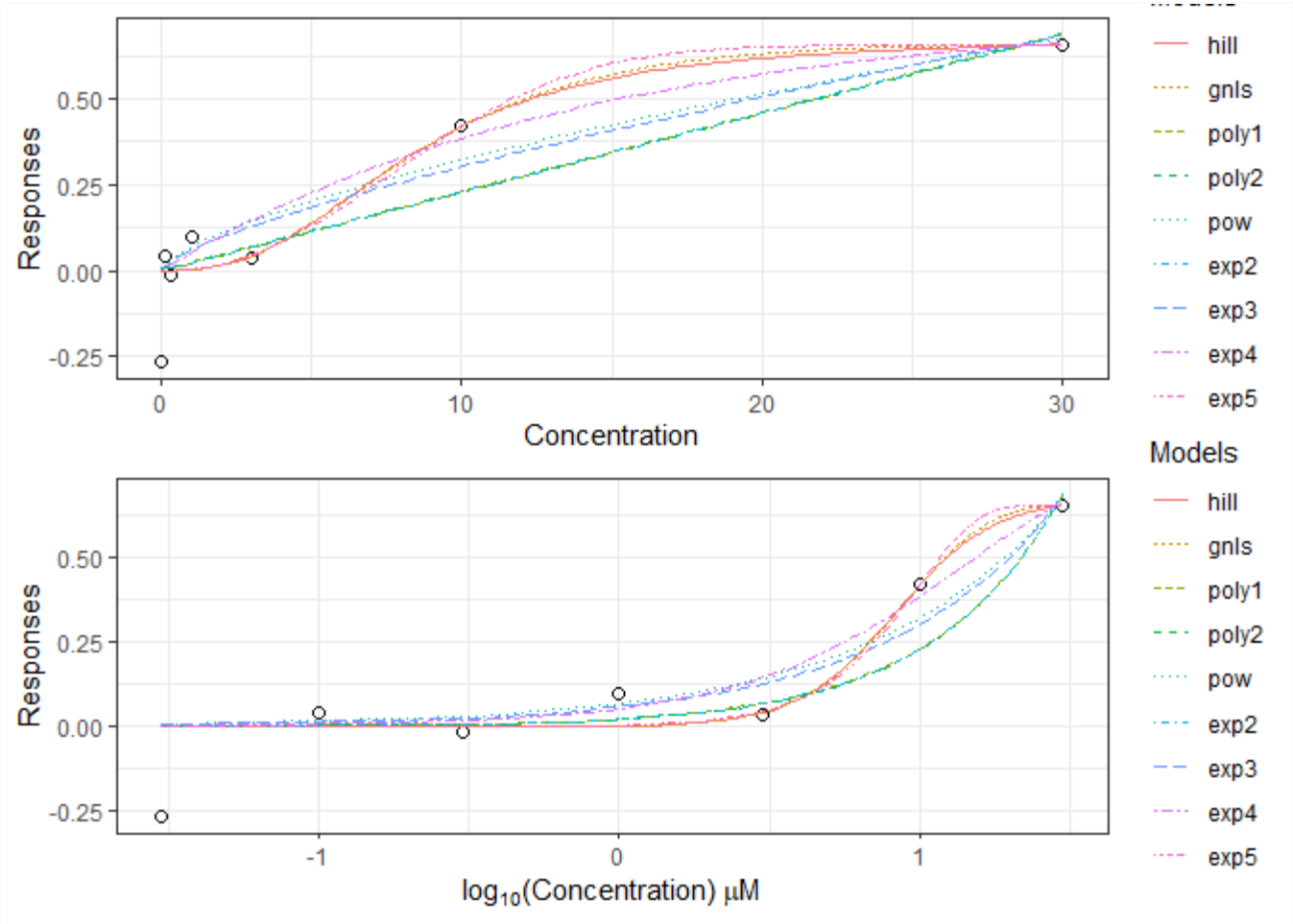


***Figure 1**: Example plots generated by `plot_allcurves`. The two plots display the experimental data (open circles) with all successful curve fits, concentrations are in the original and `log-10` scale (top and bottom plots, respectively).*

Plotting the Winning Model

Most users utilizing the `tcplfit2` package are only interested in generating a plot displaying the observed concentration-response data with the winning curve. This can be achieved with the `concRespPlot2` function, which generates a basic plot with minimal information. Minimalism in the resulting plot gives users the flexibility to include additional details they consider informative, while maintaining a clean visualization. More details on this is found in the Customization section. As with the `plot_allcurves` function, the `log_conc` argument is available to return a plot with concentrations on the `log-10` scale.

```
# prepare the 'row' object for concRespCore
row = list(conc=conc,
           resp=resp,
```

```
            bmed=0,
            cutoff=cutoff,
            onesd=signatures[1,"onesd"],
            name=signatures[1,"name"],
            assay=signatures[1,"signature"])

# run concentration-response modeling
out = concRespCore(row,conthits=F)
# show the output
out
```

```
##                       name                                  assay n_gt_cutoff
## 1 Clomiphene citrate (1:1) CMAP mometasone 7.6e-06 100 2494 100           3
##      cutoff fit_method top_over_cutoff      rmse  a  b        tp  p  q
ga
## 1 0.2393037       exp4        3.130382 0.1135895 NA NA 0.7491121 NA NA
9.591142
##   la      er       bmr     bmdl     bmdu caikwt mll hitcall     ac50
ac50_loss
## 1 NA -2.6244 0.1614104 2.380694 4.848624     NA  NA       1 9.591142
NA
##         top       ac5     ac10     ac20      acc    ac1sd      bmd
## 1 0.7491121 0.7097501 1.457883 3.087658 5.325258 2.408014 3.357835
##                     conc
## 1 0.03|0.1|0.3|1|10|3|30
##
resp
## 1
-0.267223789187984|0.0446549190032419|-0.0150009098668537|0.097550989137631|0.4201

##   errfun      AUC
## 1    dt4 13.29358
```

```
# pass the output to the plotting function
basic_plot <- concRespPlot2(out)
basic_log <- concRespPlot2(out, log_conc = TRUE)
```

```
grid.arrange(basic_plot, basic_log)
```

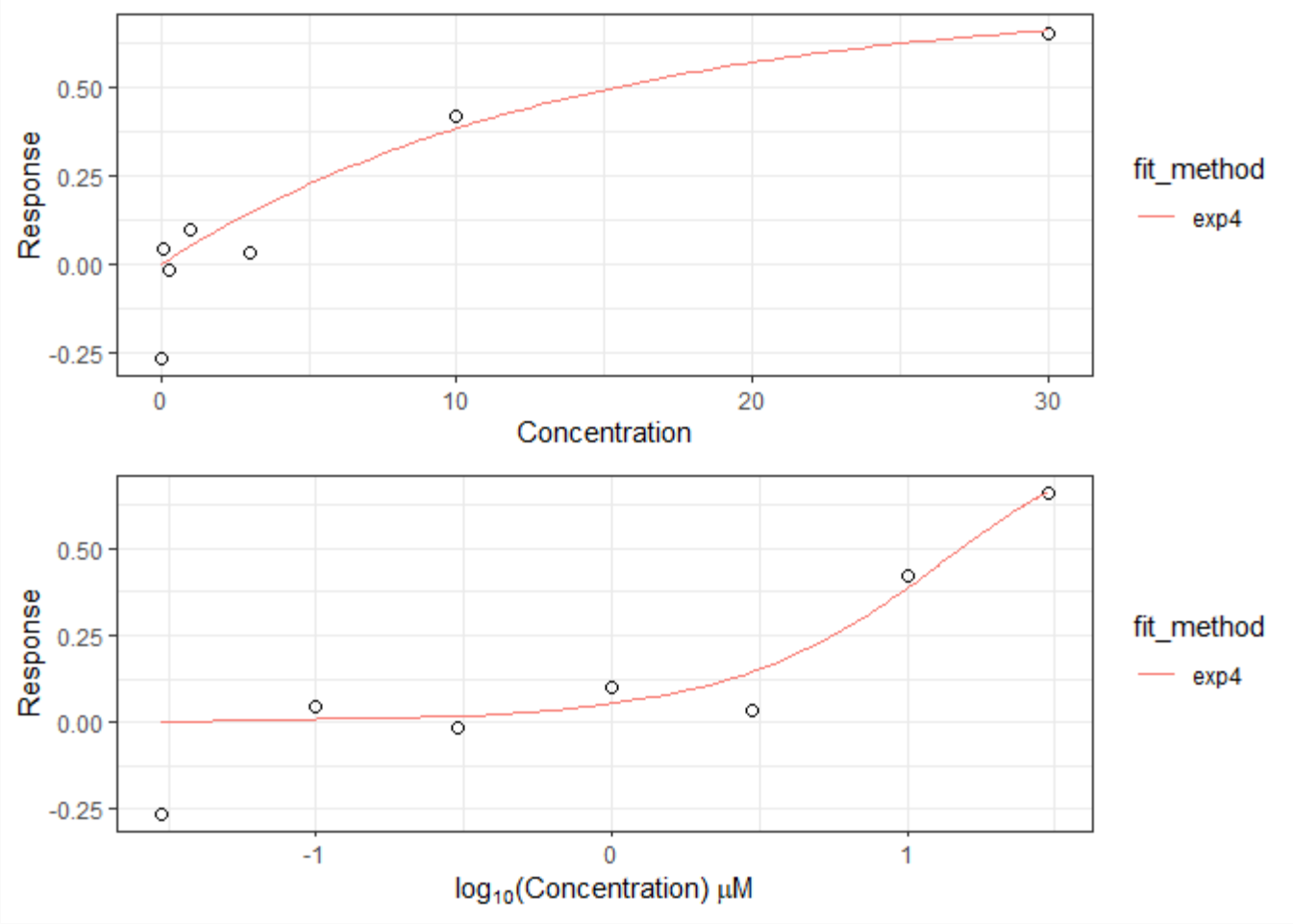


***Figure 2**: Example plots generated by `concRespPlot2`. The two plot display the experimental data (open circles) and the best curve fit (red curve), concentrations are in the original and `log-10` scale (top and bottom plots, respectively).*

Plotting Customizations

This section provides some examples on customizing a basic plot returned by `concRespPlot2` with additional information. Since `concRespPlot2` returns a `ggplot` object, additional details can be included in `ggplot2` layers. `ggplot2` layers can be added directly to the base plot with a `+` operator.

Some customizations may include, but are not limited to:

- Addition of titles displaying the evaluated compound and assay endpoint
- Visualization of the user-specified the cutoff band to evaluate response efficacy
- Points and lines to label potency estimates and relevant responses - e.g. the benchmark

dose (BMD) and benchmark response (BMR) to evaluate the estimates relative to the experimental data
- Addition of comparable data and winning curves for evaluating different experimental scenarios (e.g. multiple compounds, technologies, endpoints, etc.)

The following sub-sections explore a few customization possibilities:

Add Plot Title, Shade Cutoff Band, and Label Potency Estimates

Users may want to generate a polished figure to include in a report or publication. In this case, the basic plot may not include enough context. Thus, this section introduces simple modifications one can make to the basic plot to provide additional information. The code below adds a plot title, shades a region signifying the cutoff band, and highlights potency estimations (BMR and BMD).

```
# Using the fitted result and plot from the example in the last section
# get the cutoff from the output
cutoff <- out[, "cutoff"]

basic_plot +
  # Cutoff Band - a transparent rectangle
  geom_rect(aes(xmin = 0,xmax = 30,ymin = -cutoff,ymax = cutoff),
            alpha = 0.1,fill = "skyblue") +
  # Titles
  ggtitle(
    label = paste("Best Model Fit",
                  out[, "name"],
                  sep = "\n"),
    subtitle = paste("Assay Endpoint: ",
                     out[, "assay"])) +
  ## Add BMD and BMR labels
  geom_hline(
    aes(yintercept = out[, "bmr"]),
    col = "blue") +
  geom_segment(
    aes(x = out[, "bmd"], xend = out[, "bmd"], y = -0.5, yend = out[, "bmr"]),
    col = "blue"
```

```
  ) + geom_point(aes(x = out[, "bmd"], y = out[, "bmr"], fill = "BMD"), shape =
21, cex = 2.5)
```

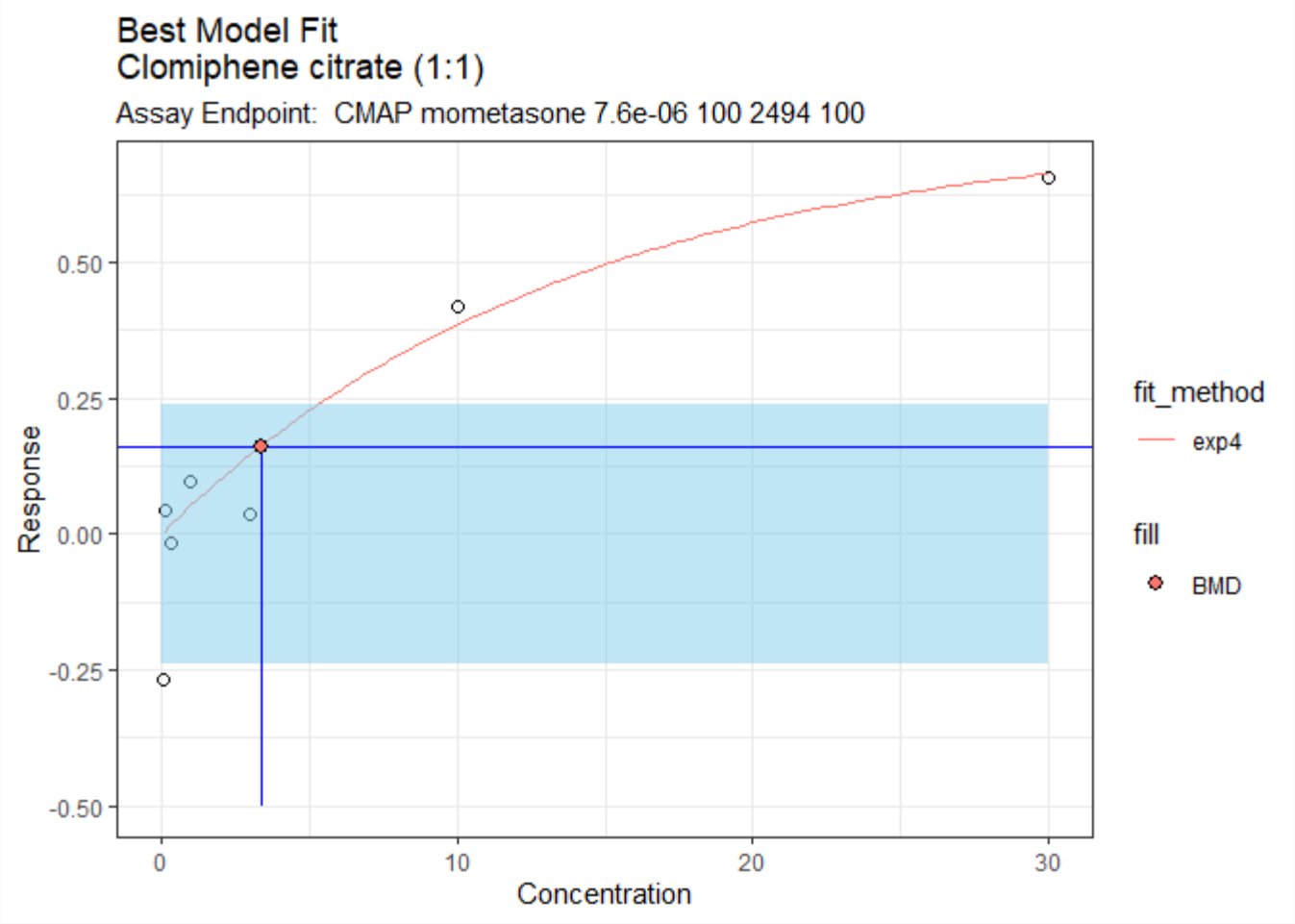


***Figure 3**: Basic plot generated with `concRespPlot2` with updated titles to provide additional details about the observed data. Experimental data is shown with the open circles and the red curve represents the best fit model. The title and subtitle display the compound name and assay endpoint, respectively. The light blue band represents responses within the cutoff threshold(s) – i.e. cutoff band. The red point represents the BMD estimated from the winning model, given the BMR. The blue segments display the BMR and the estimated BMD (horizontal and vertical segments, respectively).*

Label All Potency Estimates

`concRespCore`, and `tcplfit2_core` return several potency estimates in addition to the BMD (displayed in Figure 3) e.g. AC50, ACC, etc. Users may want to compare several potency estimates on the plot. The code chunk below demonstrates how to add all available potency estimates to the base plot. Note, when labeling potency estimates on the plot where

`log_conc = TRUE`, the potency values also need to be log-transformed to be displayed in the correct positions.

```
# Get all potency estimates and the corresponding y value on the curve
estimate_points <- out %>%
  select(bmd, acc, ac50, ac10, ac5) %>%
  tidyr::pivot_longer(everything(), names_to = "Potency Estimates") %>%
  mutate(`Potency Estimates` = toupper(`Potency Estimates`))

y = c(out[, "bmr"], out[, "cutoff"], rep(out[, "top"], 3))
y = y * c(1, 1, .5, .1, .05)
estimate_points <- cbind(estimate_points, y = y)

# add Potency Estimate Points and set colors
basic_plot + geom_point(
  data = estimate_points,
  aes(x = value, y = y, fill = `Potency Estimates`), shape = 21, cex = 2.5
)
```

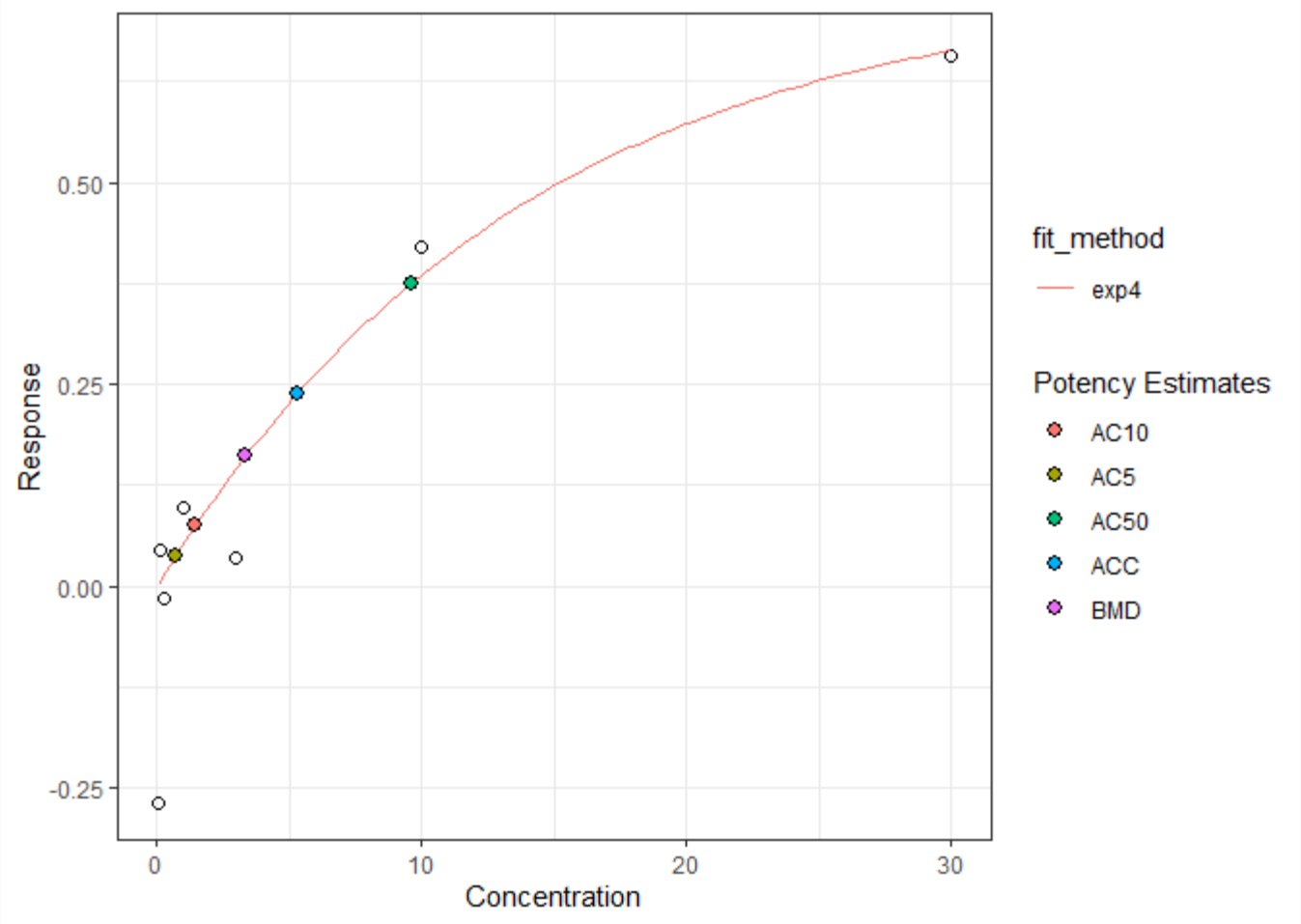


***Figure 4**: Basic plot generated by `concRespPlot2` with potency estimates highlighted. Experimental data is shown with the open circles and the red curve represents the best fit model. Five colored points represent the various potency estimates from `concRespCore`. These include the activity concentrations at 5, 10, and 50 percent of the maximal response (AC5 = gold, AC10 = red, and AC50 = green, respectively), as well as the activity concentration at the user-specified threshold (cutoff) and BMD (ACC = blue and BMD = purple, respectively).*

```
# add Potency Estimate Points and set colors - with plot in log-10
concentration
basic_log + geom_point(
  data = estimate_points,
  aes(x = log10(value), y = y, fill = `Potency Estimates`), shape = 21, cex =
2.5
)
```

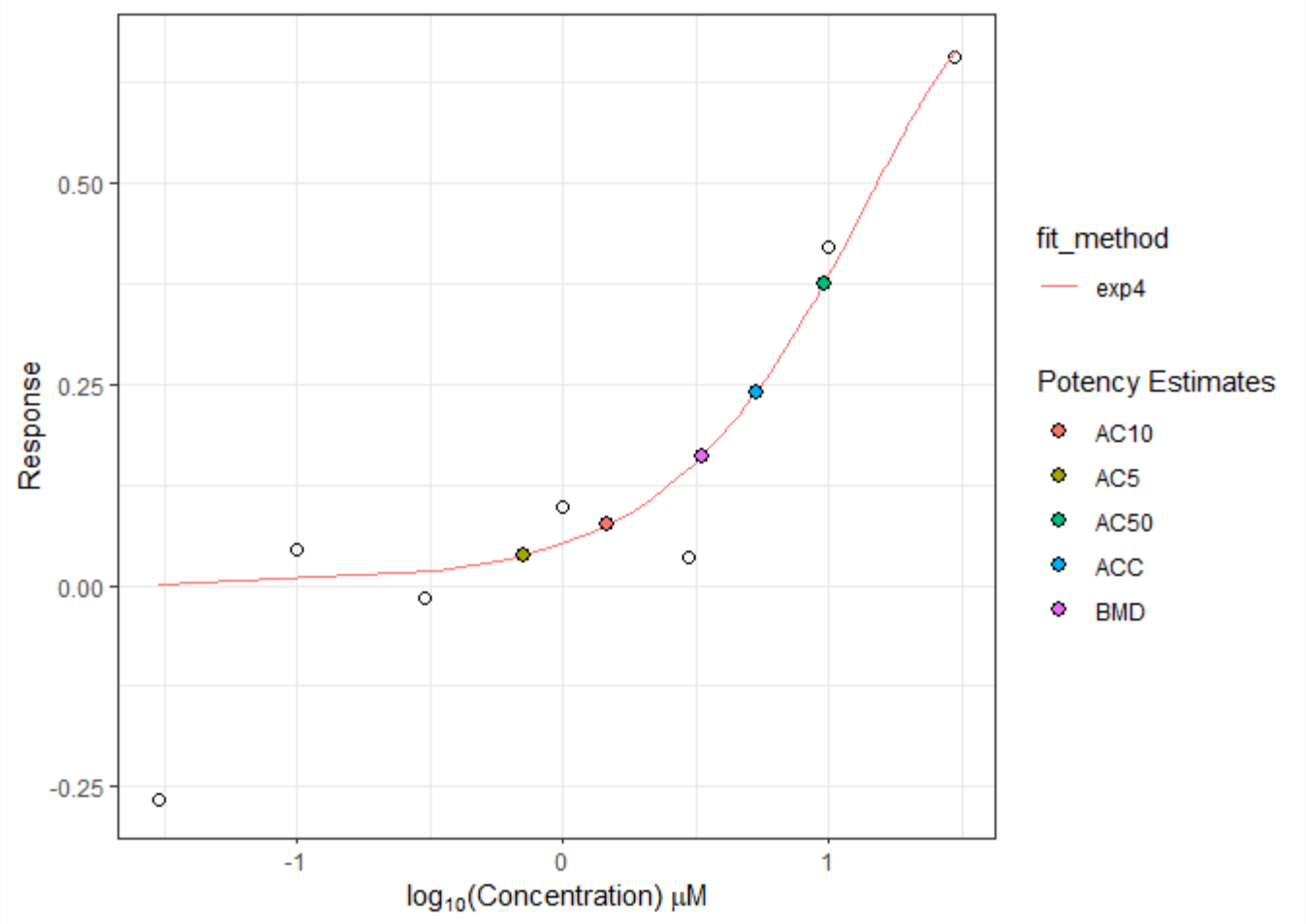


***Figure 5**: Basic plot generated by `concRespPlot2`, where `log_conc = TRUE`, with*

potency estimates highlighted. Experimental data is shown with the open circles and the red curve represents the best fit model. Five colored points represent the various potency estimates from `concRespCore`. These include the activity concentrations at 5, 10, and 50 percent of the maximal response (AC5 = gold, AC10 = red, and AC50 = green, respectively), as well as the activity concentration at the user-specified threshold (cutoff) and BMD (ACC = blue and BMD = purple, respectively).

Add Additional Curves

Working with `ggplot2` based functions can flexibly accommodate users' unique plotting needs. For example, a user might want to add one or more additional curves fits to the basic plot for comparing either various compounds, experimental scenarios, technologies, etc. To accomplish this, a user first needs to know the model to be displayed on the plot and the corresponding parameter estimates. Next, a user can generate smooth curve by predicting response for a series of 100 points within the concentration range, then add this curve to the basic pot. This section provides example code a user may modify to add another curve, and may be generalized to add more than one curve.

```
# maybe want to extract and use the same x's in the base plot
# to calculate predicted responses
conc_plot <- basic_plot[["layers"]][[2]][["data"]][["conc_plot"]]

basic_plot +
  # fitted parameter values of another curve you want to add
  geom_line(data=data.frame(x=conc_plot, y=tcplfit2::exp5(c(0.5, 10, 1.2),
conc_plot)), aes(x,y,color = "exp5"))+
  # add different colors for comparisons
  scale_colour_manual(values=c("#CC6666", "#9999CC"),
                      labels = c("Curve 1-exp4", "Curve 2-exp5")) +
  labs(title = "Curve 1 v.s. Curve 2")
```

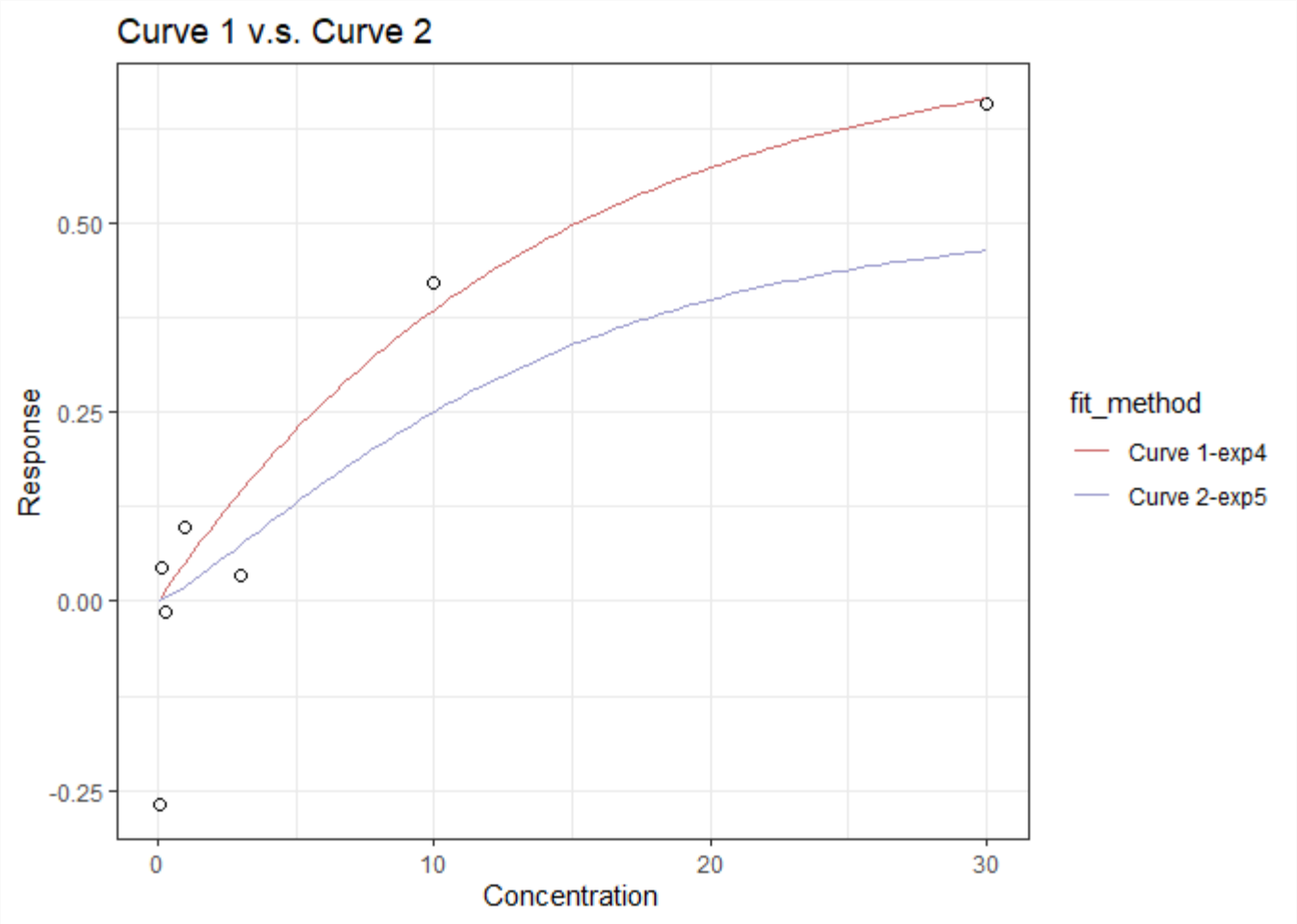


***Figure 6**: Basic plot generated by `concRespPlot2` with an additional curve for comparison. Experimental data is shown with the open circles, the red curve represents the best fit model for the baseline model, and the blue curve represents the additional curve of interest.*

Plots like Figure 6 typically have similar concentrations and response ranges. If one is comparing curves that do not have similar concentration and/or response ranges, additional alterations may be necessary.

```
  # 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)
  }
}
```

Area Under the Curve (AUC) Calculation with tcplfit2

Center for Computational Toxicology and Exposure, US EPA

Introduction

This vignette explores how to estimate the area under the curve (AUC) for concentration-response curves with `tcplfit2`, with various applications. The AUC can be interpreted as a measure of overall efficacy and potency, which users may want to include as part of their analyses.

Area Under the Curve (AUC) with `concRespCore`

The `concRespCore` function has a logical argument `AUC` controlling whether the area under the curve (AUC) is calculated for the winning model and returned alongside the other modeling results (e.g. model parameters and hit-call details). This argument defaults to `TRUE`, such that the AUC is always included in the output unless otherwise specified (i.e. `AUC=FALSE`).

```
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)

# AUC is included in the output
concRespCore(row, conthits = TRUE)
#>      n_gt_cutoff cutoff fit_method top_over_cutoff      rmse  a  b       tp
#> cnst           1      1       hill        1.225599 0.1750312 NA NA 1.225599
#>              p  q       ga la        er    bmr     bmdl     bmdu
caikwt
#> cnst 0.7752844 NA 2.554272 NA -2.467853 0.6745 2.341811 4.822553 6.064845e-
05
#>           mll   hitcall     ac50 ac50_loss      top        ac5      ac10
#> cnst 4.492301 0.9650614 2.554272        NA 1.225599 0.05726215 0.1501198
#>           ac20      acc    ac1sd      bmd                        conc
#> cnst 0.4272681 17.43267 1.580024 3.314775 0.03|0.1|0.3|1|3|10|30|100
#>                              resp errfun      AUC
#> cnst 0|0.2|0.1|0.4|0.7|0.9|0.6|1.2    dt4 1.969363
```

The following sections demonstrate how to estimate the AUC when curve fitting is performed with `concRespCore` as well as via separate calls using `tcplfit2_core` and `tcplhit2_core`. Additionally, several types of potential curve fits with the resulting AUC are highlighted with context to help with interpretation.

Positive Responses

This section provides an example of how to use `get_AUC` function in `tcplfit2` to calculate area under the curves (AUC) for a given concentration-response curve. First, example data is obtained and curve-fit.

```
# This is taken from the example under tcplfit2_core
conc_ex2 <- c(.03, .1, .3, 1, 3, 10, 30, 100)
resp_ex2 <- c(0, .1, 0, .2, .6, .9, 1.1, 1)

# fit all available models in the package
```

```
# use do.plot = TRUE to show all fitted curves
oldpar <- par(no.readonly = TRUE)
on.exit(par(oldpar))
par(xpd = TRUE)
output_ex2 <- tcplfit2_core(conc_ex2, resp_ex2, .8, do.plot = TRUE)
```

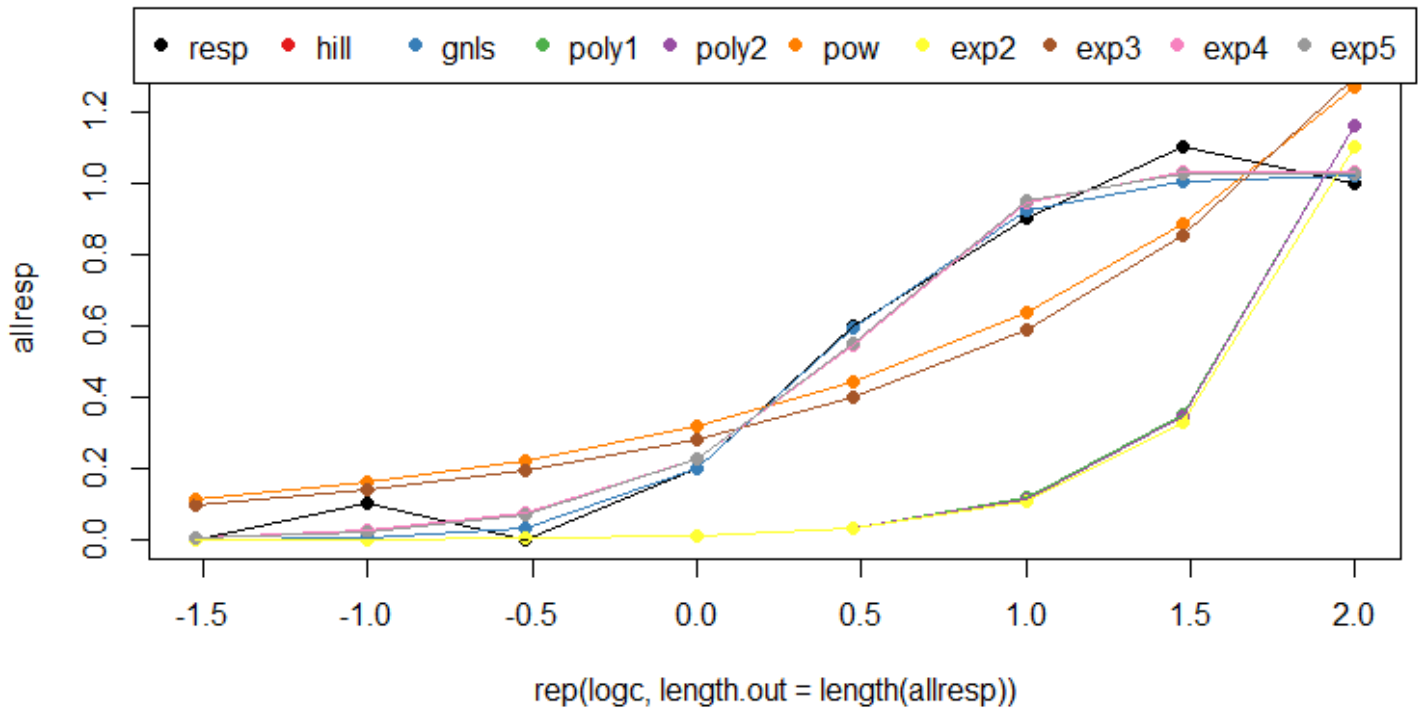


Figure 1: *This figure depicts all fit concentration-response curves. The models are polynomial 1 and 2, power, hill, gain-loss, and exponential 2 to exponential 5.*

The `get_AUC` function can be used to calculate the AUC for a single model. Inputs to this function are: the name of the model, lower and upper concentration bounds (usually the lowest and the highest concentrations in the data, respectively), and the estimated model parameters. The code chunk below demonstrates how to calculate AUC for the hill model, starting by extracting information from the `tcplfit2_core` output then inputting this information into the `get_AUC` function. After estimating the AUC, the hill curve is plot and corresponding region under the curve is shaded.

```
fit_method <- "hill"
# extract the parameters
modpars <- output_ex2[[fit_method]][output_ex2[[fit_method]]$pars]
```

```r
# plug into get_AUC function
# for hill and gnls, no need to convert concentration used for bounds to log-scale
# they will be converted inside the function
estimated_auc1 <- get_AUC(fit_method, min(conc_ex2), max(conc_ex2), modpars)
estimated_auc1
#> [1] 1.64823

# extract the predicted responses from the model
pred_resp <- output_ex2[[fit_method]][["modl"]]

# plot to see if the result make sense
# the shaded area is what the function tries to find
plot(log10(conc_ex2), pred_resp)
lines(log10(conc_ex2), pred_resp)
polygon(c(log10(conc_ex2), max(log10(conc_ex2))), c(pred_resp, min(pred_resp)), col=rgb(1, 0, 0,0.5))
```

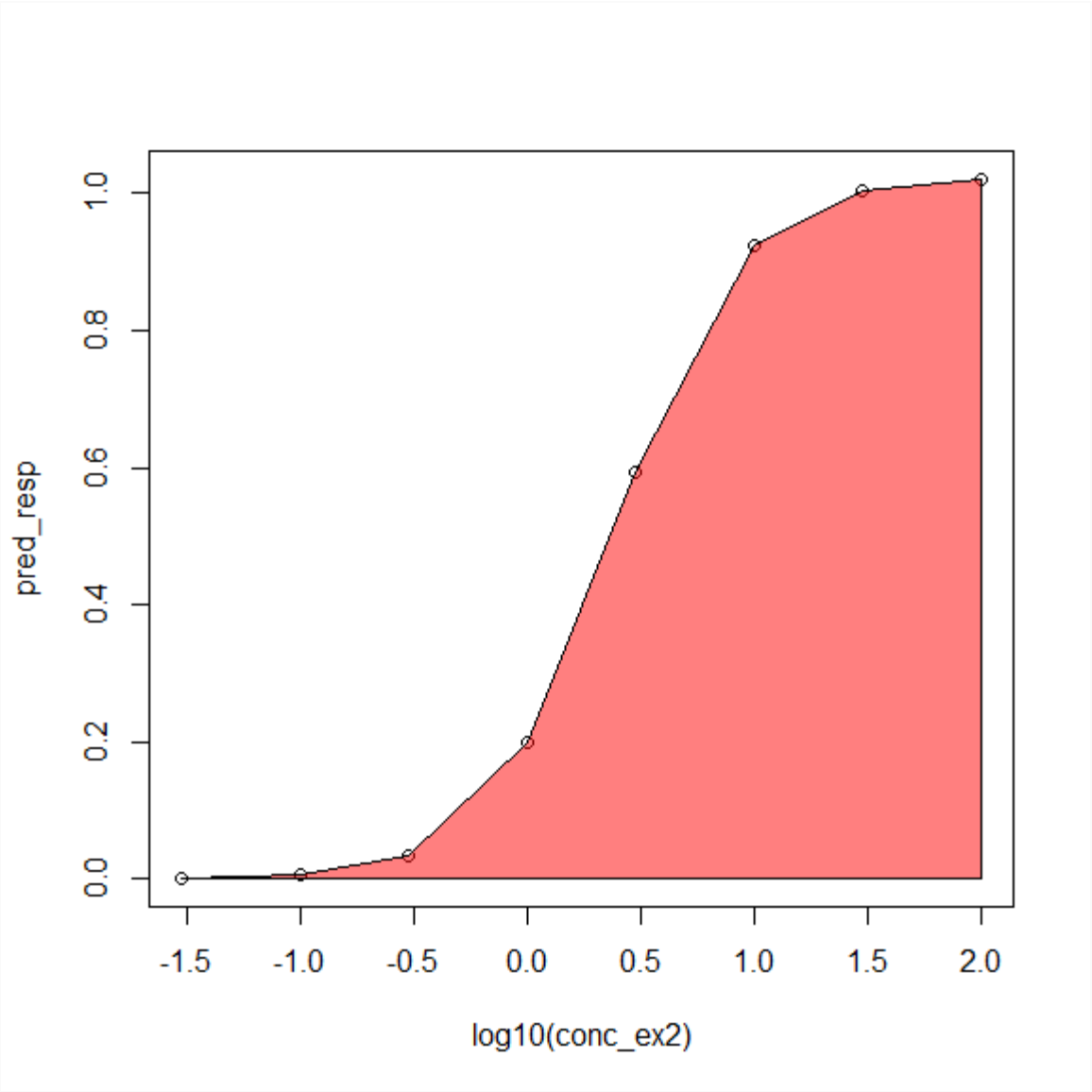


Figure 2: *The red shaded region is the area under the fitted hill curve. The AUC estimated with* `get_AUC` *is 1.64823. This estimate seems to align with the area of the shaded region.*

AUC can be calculated for other models with exception of the constant model.

```
# hill and gnls will return a much smaller number because they are in log10-
scale
fitmodels = c("gnls", "poly1", "poly2", "pow", "exp2", "exp3", "exp4", "exp5")
mylist <- list()
for (model in fitmodels){

  fit_method <- model
  # extract corresponding model parameters
  modpars <- output_ex2[[fit_method]][output_ex2[[fit_method]]$pars]
```

```
  # get AUC
  mylist[[fit_method]] <- get_AUC(fit_method, min(conc_ex2), max(conc_ex2),
modpars)

}
# print AUC's for other models
data.frame(mylist,row.names = "AUC")
#>        gnls    poly1    poly2      pow     exp2     exp3     exp4     exp5
#> AUC 1.64823 58.09263 57.89675 97.43487 55.01408 96.18956 98.80625 98.65734
```

Negative Responses

This section demonstrates the behavior of the `get_AUC` function with negative response curves. Here, example data is pulled from example 3 in the tcplfit2 Introduction Vignette.

```
# Taking the code from example 3 in the vignette
library(stringr)  # string management package
data("signatures")

# use row 5 in the data
conc=as.numeric(str_split(signatures[5,"conc"],"\\|")[[1]])
resp=as.numeric(str_split(signatures[5,"resp"],"\\|")[[1]])
cutoff=signatures[5,"cutoff"]

# plot all models, this is an example of negative curves
oldpar <- par(no.readonly = TRUE)
on.exit(par(oldpar))
par(xpd = TRUE)
output_negative <- tcplfit2_core(conc, resp, cutoff, do.plot = TRUE)
```

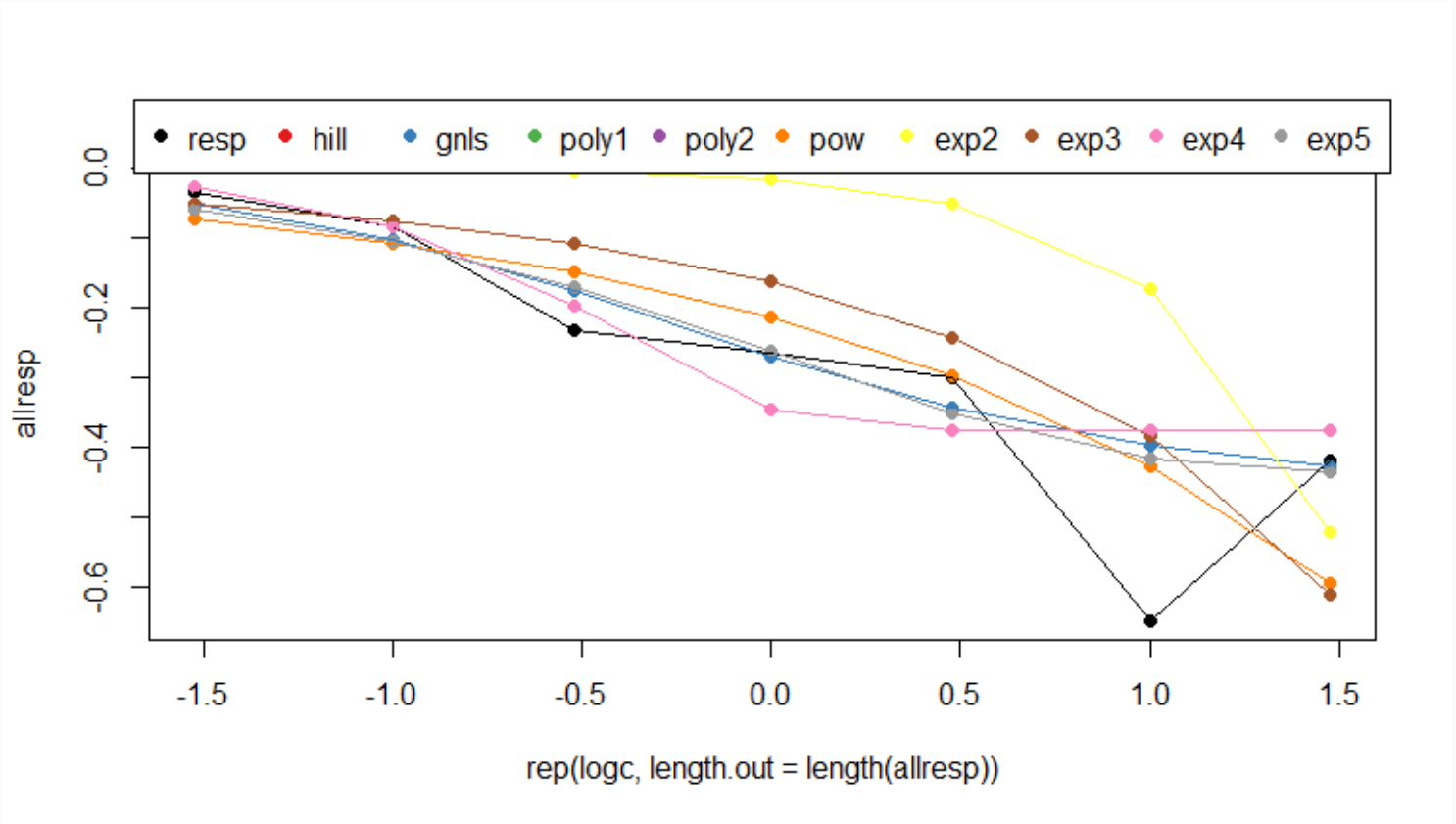


Figure 3: *This plot depicts all fitted concentration-response curves. All curves show decreasing responses starting from 0 and below x-axis.*

The code chunk below calculates the AUC for exponential 3 model with the `get_AUC` function.

```
fit_method <- "exp3"

# extract corresponding model parameters and predicted response
modpars <- output_negative[[fit_method]][output_negative[[fit_method]]$pars]
pred_resp <- output_negative[[fit_method]][["modl"]]

estimated_auc2 <- get_AUC(fit_method, min(conc), max(conc), modpars)
estimated_auc2
#> [1] -12.92738

# plot this curve
pred_resp <- pred_resp[order(conc)]
plot(conc[order(conc)], pred_resp)
lines(conc[order(conc)], pred_resp)
polygon(c(conc[order(conc)], max(conc)), c(pred_resp, max(pred_resp)),
```

```
col=rgb(1, 0, 0,0.5))
```

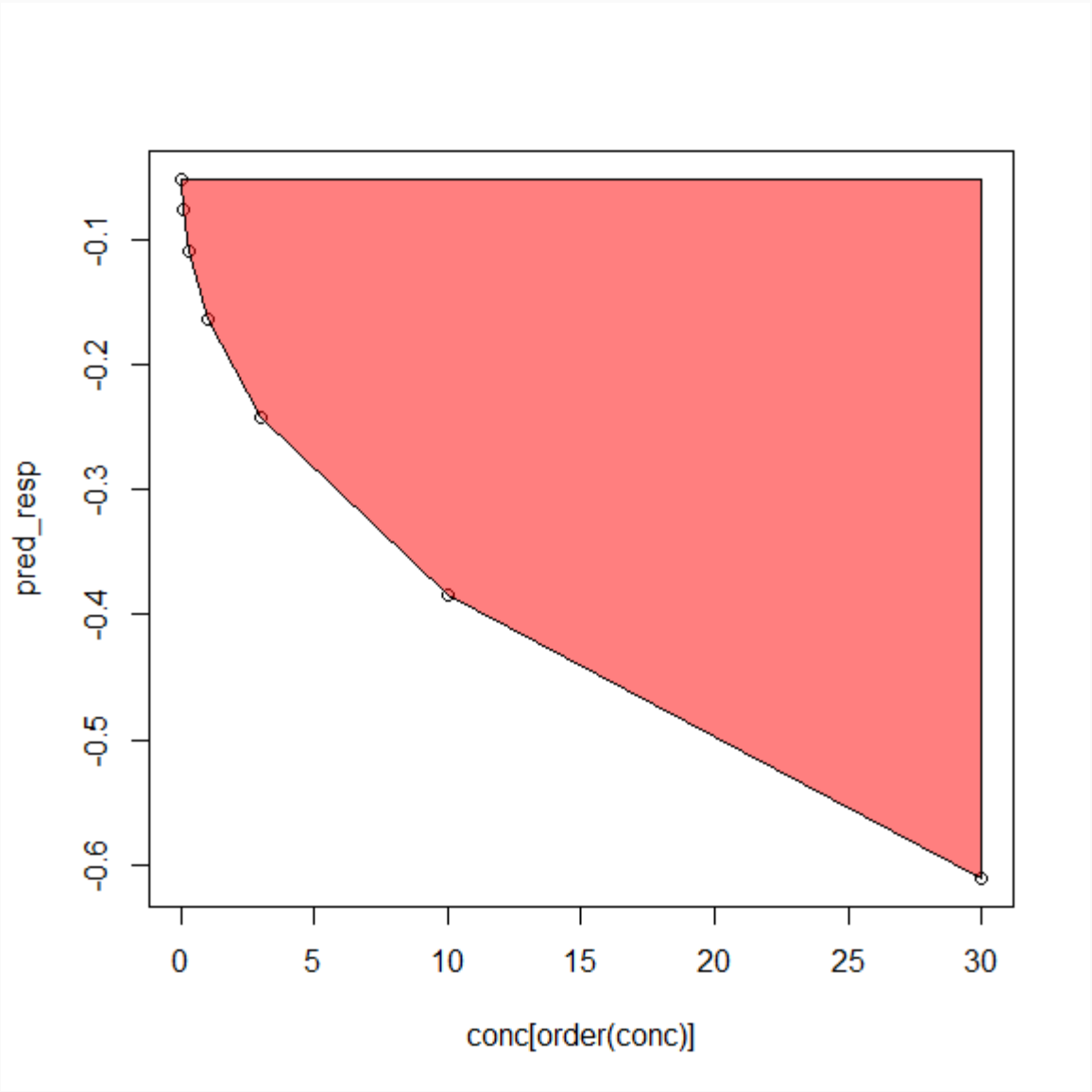


Figure 4: *Notice the function returns a negative AUC value, -12.92738. The absolute value, 12.92738, seems to align with the area between the curve and x-axis. Note: The x-axis in this plot is in the original units, so it looks different from the curve in figure 3, which has the x-axis in* log_{10} *units.*

As demonstrated, when integrating over a curve in the negative direction, the function will return a negative AUC value. However, some users may want to consider all “areas” as positive values. For this reason, the `return.abs = TRUE` argument in `get_AUC` converts negative AUC values to positive values when returned. This argument is by default `FALSE`.

```
get_AUC(fit_method, min(conc), max(conc), modpars, return.abs = TRUE)
```

```
#> [1] 12.92738
```

Bi-phasic Responses

Currently, none of the models in `tcplfit2` are capable of fitting bi-phasic curves. However, this section demonstrates what happens if a user did want to estimate the AUC for a bi-phasic curve.

The following chunk simulates a polynomial 2 curve that has area below and above the x-axis. Please note, the polynomial 2 model implemented in this package is re-parameterized such that the baseline response is always assumed to be 0.

The Polynomial 2 model in `tcplfit2` is implemented as $a * (\frac{x}{b} + \frac{x^2}{b^2})$. Here, $a = 1$ and $b = (-2)$ is used to simulate a bi-phasic curve, which can be represented in the typical form as $\frac{1}{4}x^2 - \frac{1}{2}x$.

```
# simulate a poly2 curve
ps <- unlist(list(a = 1, b = -2))
conc_sim <- c(0, 0.03, 0.10, 0.30, 1.00, 2.00, 2.50, 3.00, 3.20)
resp_sim <- poly2(ps, conc_sim)

# plot the curve
plot(conc_sim, resp_sim)
lines(conc_sim, resp_sim)
abline(h = 0)
polygon(conc_sim[1:6], resp_sim[1:6], col=rgb(1, 0, 0,0.5))
polygon(c(conc_sim[6:9], 3.20), c(resp_sim[6:9], 0), col=rgb(0, 0, 1,0.5))
```

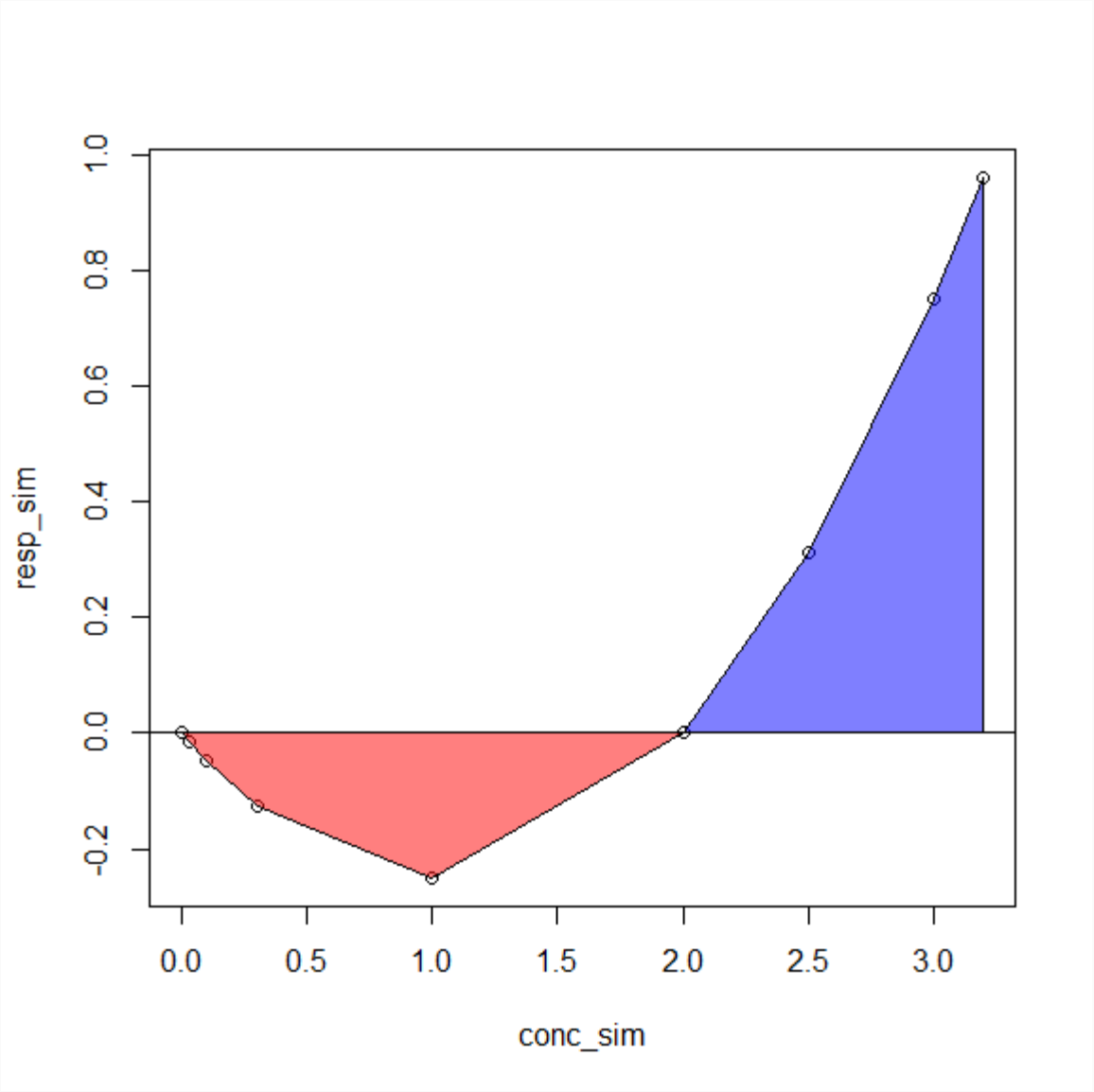


Figure 5: *This plot illustrates the simulated bi-phasic polynomial 2 curve. The curve initially decreases, then increases and crosses the x-axis at* `x = 2` *(x-intercept).*

```
# get AUC for the simulated Polynomial 2 curve
get_AUC("poly2", min(conc_sim), max(conc_sim), ps)
#> [1] 0.1706667
```

Integrating over a curve with area above and below the x-axis, the current function returns the difference between the total area above the x-axis and the total area below the x-axis (i.e. blue region minus red region). In this example, area above the x-axis is slightly larger than the area below the x-axis so the function returned a positive difference.

Since the AUC is used as an indicator of overall effect size in a toxicological context, then it

makes more sense to report the total area of the above and below x-axis (i.e., blue region plus red region) for bi-phasic curves. To do that, we need to calculate each area separately and take the sum of those areas. The code below demonstrate how this is done for the previously simulated polynomial 2 curve. It should be noted, one needs to identify where the x-intercept(s) occur in order to calculate each area. In this example, we know the x-intercept is at `x = 2`.

```
# calculate area of each section separately and then sum them together
x_intercept <- 2
total_auc <- abs(get_AUC("poly2", min(conc_sim), x_intercept, ps)) +
get_AUC("poly2", x_intercept, max(conc_sim), ps)
total_auc
#> [1] 0.8373333
```

The total area of red region plus blue region is 0.8373333.

AUC with `tcplfit2_core` and `tcplhit2_core`

In some cases, users may want to run the `tcplfit2_core` and `tcplhit2_core` functions separately, and only obtain the AUC for the winning model from `tcplhit2_core`. Thus, `tcplfit2` also includes a wrapper function for `get_AUC`, called `post_hit_AUC`, which allows users to estimate the AUC for the winning model only.

`tcplhit2_core` provides output in a data frame format with a single row containing the concentration-response data, the winning model name along with the fitted parameter values, and hit-calling results. The code chunk below provides an example demonstrating how to use the wrapper function `post_hit_AUC`. Internally, the wrapper function extracts information from the one-row data frame output and pass it to `get_AUC`, which calculates the AUC. Thus, manual entry of the model name, parameters values, etc. into `get_AUC` is not necessary with `post_hit_AUC`.

The winning model from the Positive Responses example is the Hill model. Comparing the AUC from the previous example and the AUC returned from the `post_hit_AUC` here should be identical, i.e. 1.64823.

```
out <- tcplhit2_core(output_ex2, conc_ex2, resp_ex2, 0.8, onesd = 0.4)
```

```
out
#>      n_gt_cutoff cutoff fit_method top_over_cutoff       rmse  a  b       tp
#> cnst           3    0.8       hill        1.279049 0.05022924 NA NA 1.023239
#>             p  q       ga la        er    bmr    bmdl     bmdu       caikwt
#> cnst 1.592714 NA 2.453014 NA -3.295307 0.5396 2.33877 2.979982 1.446965e-08
#>           mll   hitcall     ac50 ac50_loss      top       ac5      ac10
#> cnst 12.71495 0.9999999 2.453014        NA 1.023239 0.3862094 0.6174049
#>          ac20      acc    ac1sd      bmd                       conc
#> cnst 1.027285 5.466819 1.856853 2.627574 0.03|0.1|0.3|1|3|10|30|100
#>                          resp errfun
#> cnst 0|0.1|0|0.2|0.6|0.9|1.1|1    dt4
post_hit_AUC(out)
#> [1] 1.64823
```