R package *nplr*
n-parameter logistic regressions

Frederic Commo[1,2] and Briant M. Bot[1]

[1]*Sage Bionetworks, Fred Hutchinson Cancer Research Center, Seattle, Washington*
[2]*INSERM U981, Institut Gustave Roussy, 114 rue Edouard Vaillant, 94805 Villejuif, France*

July 11, 2015

1 Introduction

1.1 Overview

In in-vitro experiments, the aim of drug response analyses is usually to estimate the drug concentration required to reach a given cell line growth inhibition rate - typically the 50% inhibitory concentration (IC_{50}), which inhibits 50% of the proliferation, compared with an untreated control. This estimation can be achieved by modeling the inhibition rate observed under a range of drug concentrations. Once the model is fitted, the x values (drug concentrations) can be estimated from the y values (inhibition rates) by simply inverting the function.

The most commonly used model for drug response analysis is the Richards' equation [1], also refered to as a 5-parameter logistic regression [2]:

$$y = B + \frac{T - B}{\left[1 + 10^{b(x_{mid} - x)}\right]^s}$$

where B and T are the bottom and top asymptotes, and b, x_{mid} and s are the Hill slope, the x-coordinate at the inflexion point and an asymetric coefficient, respectively.

The `nplr` package is based on the full 5-parameter model, where all of the parameters are optimized, simultaneously, using a Newton-Raphson method (`nlm`, R package `stats`). The objective function to minimize is a weighted sum of squared errors:

$$sse(Y) = \Sigma_i w_i.(\hat{y}_i - y_i)^2, i = 1, ..., n$$

The weights, wi, used in the objective function can be computed using 3 possible methods, as follows:

- residuals weights: $w_i = \left(\frac{1}{res_i}\right)^p, i = 1, ..., n\ values$
- standard weights: $w_{ir} = \frac{1}{Var(y_r)}, r = 1, ..., r\ replicated\ conditions$
- general weights: $w_i = \left(\frac{1}{\hat{y}_i}\right)^p, i = 1, ..., n\ values$

where p is a tuning parameter. The `standard weights` and the `general weights` methods are described in [3, 4].

`nplr` provides several options in order to compute flexible weighted n-parameter logistic regression: `npars="all"` can be explicitly specified, from 2 to 5, or set to `all`. In that case, all the possible models are evaluated, and the optimal one is retunred, with respect to the minimal error returned by `nlm`.

The final model performance is estimated by a weighted standard error, and a weighted goodness-of-fit [5]:

- weighted standard error: $StdErr = \frac{n}{(n-1).\Sigma_i w_i}.\Sigma_i w_i.(\hat{y}_i - y_i)^2, i = 1, ..., n$
- weighted goodness-of-fit: $GOF = 1 - \frac{\Sigma_i w_i}{(n-1).S_y^2}, i = 1, ..., n$

1.2 Functions in `nplr`

The main function is simply `nplr()`. It requires 2 main arguments: a vector of `x` and a vector of `y`. Several other arguments have default values.

The *npars* argument allows a user to run specific n-parameter models, n from 2 to 5, while the default value, `npars="all"`, asks the function to test which model fits the best the data, according to a weighted Goodness-of-Fit estimator.
In some situations, the x values may need to be log-transformed, e.g. x is provided as original drug concentrations. In such case, setting `useLog=TRUE` in `nplr()` will apply a Log_{10} transformation on the x values.

The `nplr()` function has been optimized for fitting curves on y-values passed as proportions of control, between 0 to 1. If data are supplied as original response values, e.g. optic density measurements, the `convertToProp()` function may be helpful. In drug-response curve fitting, a good practice consists in adjusting the signals on a T_0 and a *control* (Ctrl) values. Providing this values, the proportion values, y_p, are computed as:

$$y_p = \frac{y - T_0}{Ctrl - T_0}$$

where y, T_0 and $Ctrl$ are the observed values, the 'time zero' and the 'untreated control', respectively.

Note that if neither T_0 nor $Ctrl$ are provided, `convertToProp()` will compute the proportions with respect to the *min* and *max* of y. In that case, the user should be aware that $y = 0.5$ may not correspond to a IC_{50}, but rather to a EC_{50} (the half-effect between the maximum and the minimum of the observed effects).

In a drug-response (or progression) curve fitting context, typical needs are to invert the function in order to estimate the x value, e.g. the IC_50, given a y

value, e.g. the 0.5 survival rate. To do so, the implemented `getEstimates()` method takes 2 arguments: the model (an instance of the class nplr), and one (or a vector of) target(s). `getEstimates()` returns the corresponding x values and their estimated confidence intervals, as specified by `conf.level`.

2 Examples

The examples below use some samples of the NCI-60 Growth Inhibition Data. The full data can be downloaded at [6]. For the purpose of the demonstration, the supplied drug concentrations have been re-exponentiated.

2.1 Example 1

2.1.1 Fitting a model

```
> require(nplr)
```

The first example fits a simple drug-response curve: the PC-3 cell line treated with Thioguanine, 19 points without replicates.

```
> path <- system.file("extdata", "pc3.txt", package="nplr")
> pc3 <- read.delim(path)
> np1 <- nplr(x=pc3$CONC, y=pc3$GIPROP)

Testing pars...
The 5-parameters model showed better performance
```

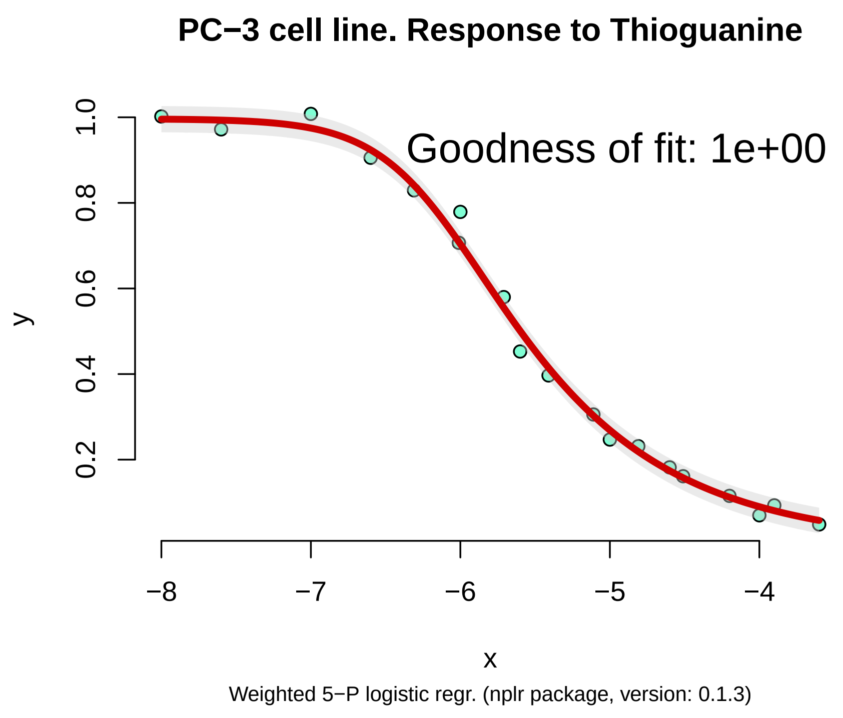
Calling the object returns the fitting summary for the model.

```
> np1
Instance of class nplr

Call:
nplr(x = pc3$CONC, y = pc3$GIPROP)

5-P logistic model
Bottom asymptote: 0.0001829361
Top asymptote: 0.9964906
Inflexion point at (x, y): -5.850105 0.627084
Goodness of fit: 0.9999821
Standard error: 0.05828571
```

2.1.2 Visualizing the model

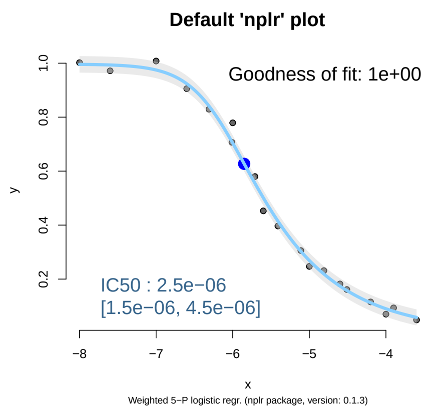
A specific `plot()` function has been implemented in order to visualize the results.

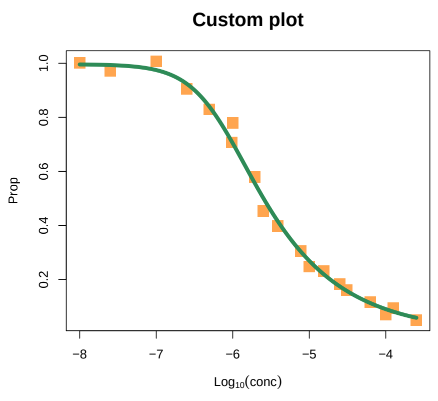
```
> plot(np1, cex.main = 1.2,
+      main="PC-3 cell line. Response to Thioguanine")
```



This function has several predefined graphical parameters, and some of them can be overwritten. However, a convenient way to draw simplest or customized plots is shown in the example below:

```
> plot(np1, pcol="grey40", lcol="skyblue1", showEstim=.5, showInfl=TRUE,
+      main="Default 'nplr' plot", cex.main=1.5)
> x1 <- getX(np1); y1 <- getY(np1)
> x2 <- getXcurve(np1); y2 <- getYcurve(np1)
> plot(x1, y1, pch=15, cex=2, col="tan1", xlab=expression(Log[10](conc)),
+      ylab="Prop", main="Custom plot", cex.main=1.5)
> lines(x2, y2, lwd=5, col="seagreen4")
```





2.1.3 Accessing the performances

Once the model is built, several accessor functions allow to get access to the performances of the model, and its parameters.

```
> getGoodness(np1)
[1] 0.9999821
```

```
> getStdErr(np1)
[1] 0.05828571
```

```
> getPar(np1)
$npar
[1] 5

$params
        bottom       top      xmid      scal         s
1 0.0001829361 0.9964906 -6.182545 -1.428008 0.3351753
```

Here, the 5-parameter model have been chosen as it showed better performances, according to the goodness-of-fit (`npar=5`). The optimal values for the parameters are reported in `params`.

2.1.4 Estimating the drug concentrations

The purpose of such fitting is to estimate the response to the drug. To do so, `nplr` provides 2 estimates: the area under the curve (AUC), and the drug concentration for a given response to reach.

The `getAUC()` function returns the area under the curve (AUC) estimated by the trapezoid rule and the Simpson's rule, while `getEstimates()` invert the function and returns the estimated concentration for a given response. If no target is specified, the default output is a table of the x values corresponding to responses from 0.9 to 0.1.

```
> getAUC(np1)
  trapezoid  Simpson
1  2.507094 2.527395
```

```
> getEstimates(np1)
    y        x.025            x        x.975
1 0.9 5.954381e-08 3.181424e-07 6.642209e-07
2 0.8 2.765216e-07 6.223618e-07 1.083694e-06
3 0.7 5.827073e-07 1.017702e-06 1.709314e-06
4 0.6 9.532372e-07 1.593305e-06 2.679280e-06
5 0.5 1.492151e-06 2.522415e-06 4.598222e-06
6 0.4 2.371194e-06 4.219737e-06 8.598157e-06
7 0.3 3.900765e-06 7.916256e-06 2.180856e-05
8 0.2 7.328584e-06 1.873892e-05 1.056177e-04
9 0.1 1.623945e-05 8.030163e-05 6.587494e-03
```

A single value (a target), or a vector of values, can be passed to `getEstimates()`, and a confidence level can be specified (by default, conf.level is set to .95).

```
> getEstimates(np1, .5)
    y        x.025            x        x.975
1 0.5 1.509178e-06 2.522415e-06 4.585057e-06
```

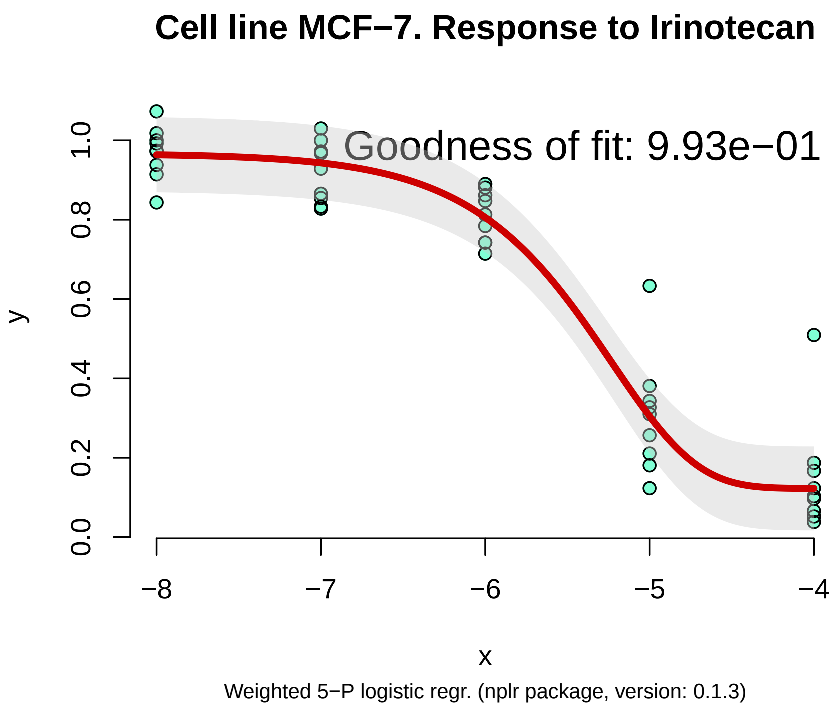
```
> getEstimates(np1, c(.25, .5, .75), conf.level=.90)
     y         x.05            x         x.95
1 0.25 5.790128e-06 1.168667e-05 3.247022e-05
2 0.50 1.615236e-06 2.522415e-06 4.148069e-06
3 0.75 4.694523e-07 8.041128e-07 1.248234e-06
```

2.2 Example 2

The next example analyses a drug-response experiment with replicated drug concentrations: the MCF-7 cell line treated with Irinotecan.

```
> path <- system.file("extdata", "mcf7.txt", package="nplr")
> mcf7 <- read.delim(path)
> np2 <- nplr(x=mcf7$CONC, y=mcf7$GIPROP)

Testing pars...
The 5-parameters model showed better performance
```

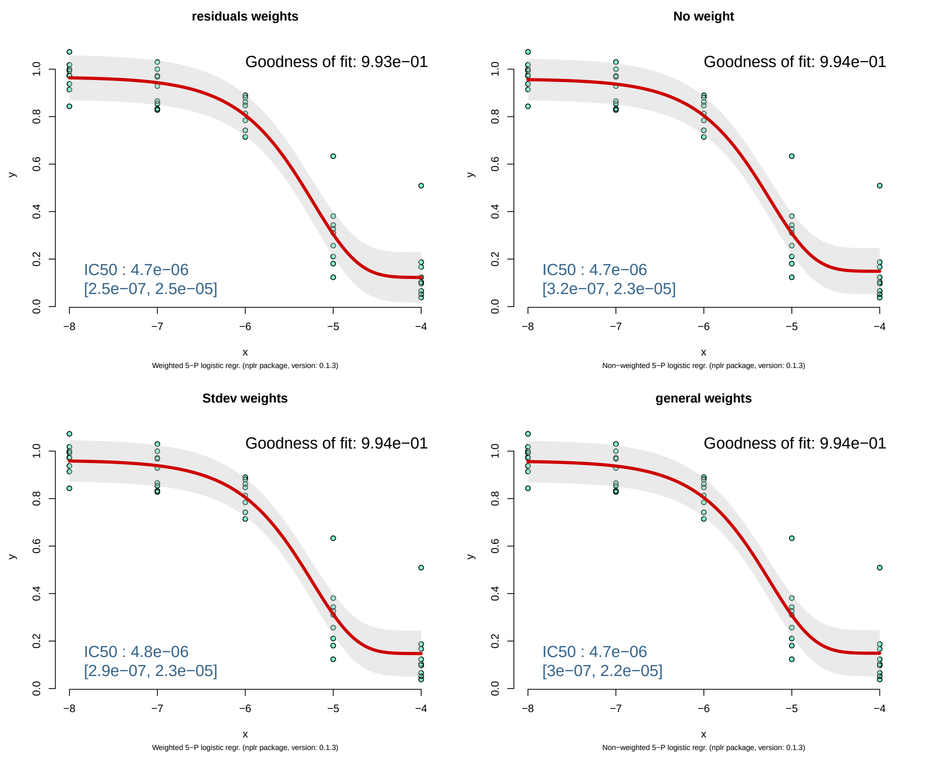
```
> plot(np2 , cex.main=1.25, main="Cell line MCF-7. Response to Irinotecan")
```



As there are replicates, we can compare the effect of the different weighted methods: the default method is `residuals weights, "res"`. A `no-weight` condition can be tested by setting the `LPweight` argument to 0: The vector of weights is then just a vector of 1's.

```
> x <- mcf7$CONC
> y <- mcf7$GIPROP
> noweight <- nplr(x, y, LPweight=0, silent=TRUE)
> sdw <- nplr(x, y, method="sdw", silent=TRUE)
> gw <-  nplr(x, y, method="sdw", LPweight=1.5, silent=TRUE)
```

```
> plot(np2, showEstim=.5, main="residuals weights")
> plot(noweight, showEstim=.5, main="No weight")
> plot(sdw, showEstim=.5, main="Stdev weights")
> plot(noweight, showEstim=.5, main="general weights")
```



Note that the curves do not seem to change dramatically. However, the different weights can give different performances.

2.3 Example 3

2.3.1 Fitting a Progression/Time model

This last example illustrates a Progression/Time experiment: these are simulated data.

```
> path <- system.file("extdata", "prog.txt", package="nplr")
> prog <- read.delim(path)
```

Here, the progression values are given in some unknown unit, and the x values are `Time` in hours. So we don't need to use a Log_{10} transformation. Let us assume that we have access to the T_0 and the *control* values. We can use `convertToProp()` in order to convert the y values to proportions.

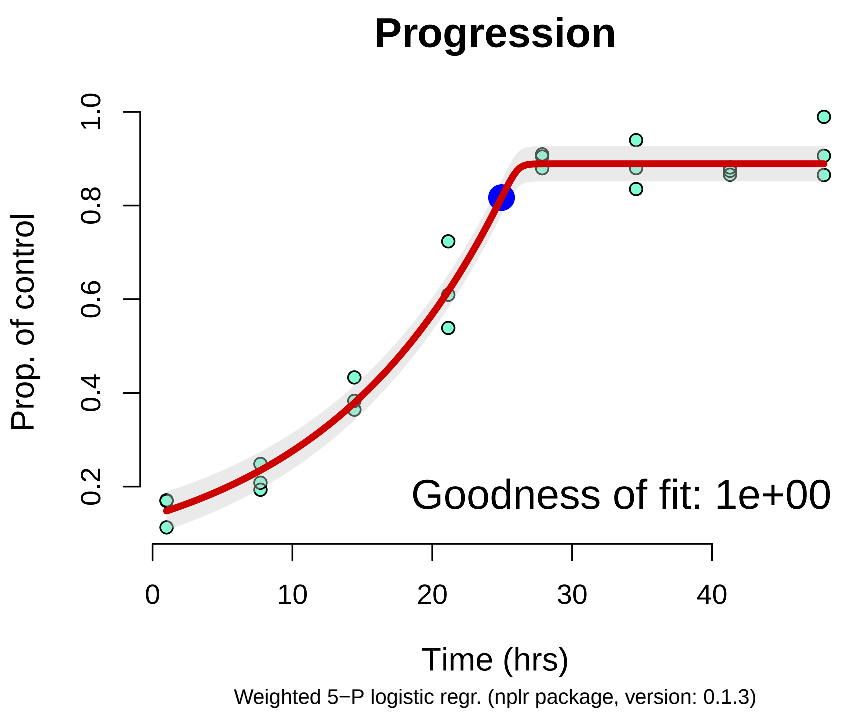
```
> x <- prog$time
> yp <- convertToProp(prog$prog, T0 = 5, Ctrl = 102)
> np3 <- nplr(x, yp, useLog=FALSE)
```

```
Testing pars...
The 5-parameters model showed better performance
```

When progression is at stake, it may be interesting to get the coordinates of the inflexion point, as it corresponds to the point where the slope (the progression) is maximal.

```
> getInflexion(np3)
         x        y
1 24.95111 0.817017
```

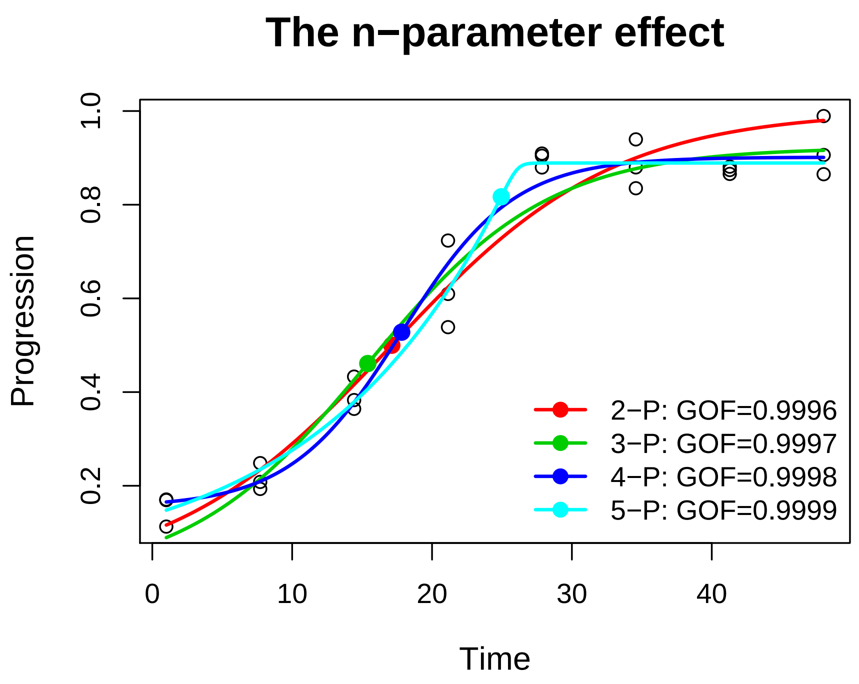
```
> plot(np3, showInfl=TRUE, xlab="Time (hrs)", cex.main=1.5, cex.lab=1.2,
+      ylab="Prop. of control", main="Progression")
```



2.3.2 Evaluating the number of parameters

When a 5-p logistic regression is used, and because of the asymetric parameter, the curve is no longer symetrical around its inflexion point. Here is an illustration of the impact of the number of parameters on the fitting.

```
> plot(x, yp , cex.main=1.5, cex.lab=1.2,
+      main="The n-parameter effect", xlab="Time", ylab="Progression")
> le <- c()
> for(i in 2:5){
+   test <- nplr(x, yp, npars=i, useLog=FALSE)
+   lines(getXcurve(test), getYcurve(test), lwd=2, col=i)
+   points(getInflexion(test), pch=19, cex=1.25, col=i)
+   gof <- getGoodness(test)
+   le <- c(le, sprintf("%s-P: GOF=%s", i, round(gof, 4)))
+ }
> legend("bottomright", legend=le, lwd=2, pch=19, col=2:5, bty="n")
```

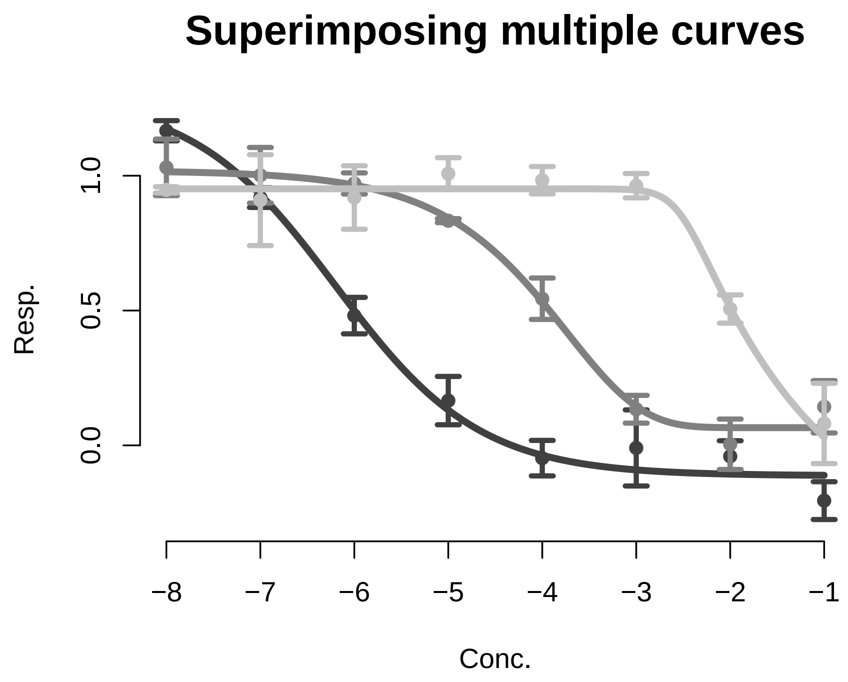


Note that even if it is the case here, the 5-P model may not be always the best choice.

2.4 Superimposing multiple curves

When multiple cell lines, or multiple compounds are compared, it can be useful to superimpose the response curves on a single plot.

```
> # Simulating responses
> set.seed(123)
> drug <- rep(seq(-8, -1), 3)
> Resp <- lapply(c(-6, -4, -2), function(xmid){
+     scal <- rnorm(1, -1, .1)
+     s <- rnorm(1, 1, .1)
+     err <- rnorm(length(drug), 0, .15)
+     nplr:::.nPL5(0, 1, xmid, scal, s, drug) + err
+     }
+ )
>
> # Computing models (to store in a list)
> Models <- lapply(Resp, function(resp){
+   nplr(10^drug, resp, silent = TRUE)
+   })
>
> # Visualizing
> overlay(Models, xlab = "Conc.", ylab = "Resp.",
+   main="Superimposing multiple curves", cex.main=1.5)
```



3 Accessing R code

The R code for `nplr` is available on github: `https://github.com/fredcommo/nplr`

References

[1] Richards FJ. "A flexible growth function for empirical use." In: *J Exp Bot.* 10 (1959), pp. 290–300.

[2] Giraldo J et al. “Assessing the (a)symmetry of concentration-effect curves: empirical versus mechanistic models.” In: *Pharmacol Ther.* 95.1 (2002), pp. 21–45.

[3] Motulsky HJ and Brown RE. “Assessing the (a)symmetry of concentration-effect curves: empirical versus mechanistic models.” In: *BMC Bioinformatics* 9 (2006), pp. 7–123.

[4] Laurence M. Levasseur et al. “Implications for Clinical Pharmacodynamic Studies of the Statistical Characterization of an In Vitro Antiproliferation Assay.” In: *Journal of Pharmacokinetics and Biopharmaceutics* 26.6 (Dec. 1998), pp. 717–733.

[5] John B Willett and Judith D Singer. “Another Cautionary Note about R2: Its Use in Weighted Least-Squares Regression Analysis.” In: *The American Statistician* 42.3 (Dec. 1988), pp. 236–238.

[6] URL: `https://wiki.nci.nih.gov/display/NCIDTPdata/NCI-60+Growth+Inhibition+Data`.