# Package: ospsuite.parameteridentification (via r-universe)

July 15, 2026

**Type** Package

**Title** Open Systems Pharmacology Parameter Identification package

**Version** 2.2.0

**Maintainer** Pavel Balazki <pavel.balazki@esqlabs.com>

**Description** R package for performing parameter identification for the Open Systems Pharmacology models.

**License** GPL-2 | file LICENSE

**URL** https://github.com/Open-Systems-Pharmacology/OSPSuite.ParameterIdentification, https://www.open-systems-pharmacology.org/OSPSuite.ParameterIdentification

**BugReports** https://github.com/Open-Systems-Pharmacology/OSPSuite.ParameterIdentification/issues

**Depends** R (>= 4.4), ospsuite (>= 12.4.0)

**Imports** DEoptim, DiceKriging, dfoptim, ggplot2, ggtext, nloptr, numDeriv, ospsuite.utils (>= 1.9.0), patchwork, purrr, R6, stats, tibble, utils

**Suggests** devtools, knitr, mockthat, readr, rmarkdown, testthat (>= 3.0.0), vdiffr

**Roxygen** list(markdown = TRUE)

**VignetteBuilder** knitr

**Config/roxygen2/version** 8.0.0

**Config/testthat/edition** 3

**Encoding** UTF-8

**Remotes** ospsuite=Open-Systems-Pharmacology/OSPSuite-R@*release

**Config/pak/sysreqs** cmake dotnet-runtime-8.0 make libicu-dev libjpeg-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libx11-dev

**Repository** https://open-systems-pharmacology.r-universe.dev

**Date/Publication** 2026-06-03 13:16:30 UTC

**RemoteUrl** https://github.com/Open-Systems-Pharmacology/OSPSuite.ParameterIdentification

**RemoteRef** v2.2.0

**RemoteSha** cb9e9e5859dc31732a31130f8f46307118420a65

# Contents



---

`AlgorithmOptions` *Algorithm Options for Optimization Algorithms*

---

## Description

Default options for various optimization algorithms used within the package. These algorithms are configured via the `PIConfiguration` class.

## Usage

```
AlgorithmOptions_HJKB

AlgorithmOptions_BOBYQA

AlgorithmOptions_DEoptim

AlgorithmDefaults
```

## AlgorithmOptions_HJKB

Default options for the `HJKB` algorithm. See `dfoptim::hjkb()` for more details on the options.

### AlgorithmOptions_BOBYQA

Default options for the BOBYQA algorithm. See nloptr::nl.opts() for more details on the options.

### AlgorithmOptions_DEoptim

Default options for the DEoptim algorithm. See DEoptim::DEoptim.control() for more details on the options.

---

## Algorithms *Optimization Algorithms*

---

### Description

Optimization algorithms supported by optimization routines, available for use in the ParameterIdentification class. These algorithms are configured via the PIConfiguration class.

### Usage

```
Algorithms
```

### Details

Supported algorithms include:

- HJKB – Hooke-Jeeves algorithm from the **dfoptim** package.
- BOBYQA – BOBYQA algorithm from the **nloptr** package.
- DEoptim – Differential evolution algorithm from the **DEoptim** package, suitable for stochastic global optimization.

  These algorithms can be specified and configured within the PIConfiguration class to tailor the parameter identification process to specific needs.

---

## CIMethods *Confidence Interval Estimation Methods*

---

### Description

Confidence interval estimation methods supported in the ParameterIdentification class. These methods are configured via the PIConfiguration class.

### Usage

```
CIMethods
```

## Details

Supported methods:

- hessian - Hessian-based approximation using the Fisher Information Matrix.
- PL – Profile likelihood estimation, iterating over each parameter.
- bootstrap – Bootstrap resampling to estimate parameter uncertainty.
  These methods can be specified and configured within the PIConfiguration class to customize confidence interval estimation.

---

CIOptions *Confidence Interval Estimation Options*

---

## Description

Default options for various confidence interval estimation methods used within the package. These methods are configured via the PIConfiguration class.

## Usage

```
CIOptions_hessian

CIOptions_PL

CIOptions_bootstrap

CIDefaults
```

## Format

A list containing default settings for confidence interval estimation methods.

## CIOptions_hessian

Default options for the hessian method. The Hessian matrix is computed via numDeriv::hessian() using the Richardson method (the only supported method). The r and d options map directly to method.args of that function.

- epsilon: Numerical step size for numerical differentiation. Default is NULL, which applies an adaptive step size based on the parameter values.
- confLevel: Confidence level for interval estimation. Default is 0.95 (95% confidence intervals).
- r: Number of Richardson iterations. Controls the trade-off between accuracy and computation time: approximately (N^2 + N)*r objective function evaluations are performed, where N is the number of free parameters. Must be at least 2. Default is NULL, which uses numDeriv's default of 4.
- d: Fractional step size. Smaller values reduce the step size relative to the parameter value. Default is NULL, which uses numDeriv's default of 0.1.

### CIOptions_PL

Default options for the `PL` (Profile Likelihood) method.

- `epsilon`: Numerical step size for profile likelihood adjustments. Default is `NULL`, which applies an adaptive step size.
- `confLevel`: Confidence level for interval estimation. Default is `0.95` (95% confidence intervals).
- `maxIter`: Maximum number of iterations for likelihood profiling. Default is `100`.

### CIOptions_bootstrap

Default options for the `bootstrap` method.

- `nSamples`: Number of bootstrap resampling iterations. Default is `1000`.
- `confLevel`: Confidence level for interval estimation. Default is `0.95` (95% confidence intervals).
- `seed`: Random seed for reproducibility. Default is `NULL`.

---

## `getOSPSuitePISetting` *Get the value of a global ospsuite.parameteridentification setting.*

---

### Description

Get the value of a global ospsuite.parameteridentification setting.

### Usage

```
getOSPSuitePISetting(settingName)
```

### Arguments

| | |
|---|---|
| `settingName` | String name of the setting |

### Value

Value of the setting stored in `piEnv`. If the setting does not exist, an error is thrown.

### Examples

```
getOSPSuitePISetting("packageVersion")
```

## ObjectiveFunctionOptions

*Objective Function Options for Model Fit Assessment*

### Description

Default settings for objective function options in model fit analysis, pivotal for calculating error and fit. These configurations enable tailored analysis approaches, pivotal in the `ParameterIdentification` process for optimizing parameter values against observed data. These options are configured via the `PIConfiguration` class.

### Usage

```
ObjectiveFunctionOptions
```

### Details

Settings include:

- `objectiveFunctionType` – Type of objective function used. Default is `lsq` (least squares), influencing error calculation.
- `residualWeightingMethod` – Method for residual weighting. Default is `none`.
- `robustMethod` – Method for robust outlier handling. Default is `none` (standard analysis).
- `scaleVar` – Whether residual scaling is applied. Default is `FALSE`.
- `linScaleCV` – Coefficient of variation for linear scaling. Default is `0.2`.
- `logScaleSD` – Standard deviation for log scaling. Default is `NULL`.

  These options are configurable in `PIConfiguration`, directly influencing the `calculateCostMetrics` functionality for detailed model fit assessment.

## ospsuitePIRSettingNames

*Names of the settings stored in piEnv Can be used with* `getOSPSuitePISetting()`

### Description

Names of the settings stored in piEnv Can be used with `getOSPSuitePISetting()`

### Usage

```
ospsuitePIRSettingNames
```

ParameterIdentification

*ParameterIdentification*

## Description

Performs parameter estimation by fitting model simulations to observed data. Supports customizable optimization and confidence interval methods.

## Active bindings

`simulations` A named list of `Simulation` objects, keyed by the IDs of their root containers.

`parameters` A list of `PIParameters`, each representing a grouped set of model parameters to be optimized (read-only).

`configuration` A `PIConfiguration` object controlling algorithm, CI estimation, and objective function options.

`outputMappings` A list of `PIOutputMapping` objects mapping observed datasets to simulated outputs.

`pkOutputMappings` A list of `PKOutputMapping` objects for PK metric optimization. `NULL` in standard PI mode. Read-only.

## Methods

### Public methods:

- `ParameterIdentification$new()`
- `ParameterIdentification$run()`
- `ParameterIdentification$estimateCI()`
- `ParameterIdentification$plotResults()`
- `ParameterIdentification$gridSearch()`
- `ParameterIdentification$calculateOFVProfiles()`
- `ParameterIdentification$print()`

`ParameterIdentification$new()`**:** Initializes a `ParameterIdentification` instance.

*Usage:*

```
ParameterIdentification$new(
  simulations,
  parameters,
  outputMappings = NULL,
  pkOutputMappings = NULL,
  configuration = NULL
)
```

*Arguments:*

`simulations` A `Simulation` or list of `Simulation` objects to be used for parameter estimation. Each simulation must contain the model parameters specified in `parameters`. Use `ospsuite::loadSimulation()` to load simulation files.

`parameters` A `PIParameters` or list of `PIParameters` objects specifying the model parameters to optimize. Each `PIParameters` object may group one or more underlying model parameters. See `PIParameters` for details.

`outputMappings` (Optional) A `PIOutputMapping` or list of `PIOutputMapping` objects mapping model outputs to observed data. Mutually exclusive with `pkOutputMappings`.

`pkOutputMappings` (Optional) A `PKOutputMapping` or list of `PKOutputMapping` objects for PK metric optimization. Mutually exclusive with `outputMappings`.

`configuration` (Optional) A `PIConfiguration` object specifying algorithm, CI method, and objective function settings. Defaults to a new configuration if omitted. See `PIConfiguration` for configuration options.

*Returns:* A `ParameterIdentification` object ready to run parameter estimation. Executes Parameter Identification

`ParameterIdentification$run()`**:** Runs parameter identification using the configured optimization algorithm. Returns a structured `piResults`object containing estimated parameters, diagnostics, and (optionally) confidence intervals.

*Usage:*

```
ParameterIdentification$run()
```

*Returns:* A `PIResult` object in standard mode, or a `PKResult` object (internal) when `pkOutputMappings` was provided. Estimate Confidence Intervals

`ParameterIdentification$estimateCI()`**:** Computes confidence intervals for the optimized parameters using the method defined in the associated `PIConfiguration`. Intended for advanced use when `autoEstimateCI` was set to `FALSE` during the initial run.

*Usage:*

```
ParameterIdentification$estimateCI()
```

*Returns:* The same `PIResult` object returned by the `run()` method, updated to include confidence interval estimates. Plot Parameter Estimation Results

`ParameterIdentification$plotResults()`**:** Re-runs model simulations using the current or specified parameter values and generates plots comparing predictions to observed data.

*Usage:*

```
ParameterIdentification$plotResults(par = NULL)
```

*Arguments:*

`par` Optional parameter values for simulations, in the order of `ParameterIdentification$parameters`. Use current values if `NULL`.

*Returns:* A list of `patchwork` objects (one per output mapping), showing:

- Individual time profiles
- Predicted vs. observed values
- Residuals vs. time Perform a Parameter Grid Search

Generates a grid of parameter combinations, computes the OFV for each, and optionally sets the best result as the starting point for s subsequent optimization.
Note: The resulting grid can be used to explore the parameter space or initialize better starting values.

`ParameterIdentification$gridSearch()`:

*Usage:*

```
ParameterIdentification$gridSearch(
  lower = NULL,
  upper = NULL,
  logScaleFlag = FALSE,
  totalEvaluations = 50,
  setStartValue = FALSE
)
```

*Arguments:*

`lower` Numeric vector of parameter lower bounds, defaulting to `PIParameter` minimum values.

`upper` Numeric vector of parameter upper bounds, defaulting to `PIParameter` maximum values.

`logScaleFlag` Logical scalar or vector; determines if grid points are spaced logarithmically. Default is `FALSE`.

`totalEvaluations` Integer specifying the total grid points. Default is 50.

`setStartValue` Logical. If `TRUE`, updates `PIParameter` starting values to the best grid point. Default is `FALSE`.

*Returns:* A tibble where each row is a parameter combination and the corresponding objective function value (`ofv`). Calculate Objective Function Value (OFV) Profiles

`ParameterIdentification$calculateOFVProfiles()`: Generates OFV profiles by varying each `PIParameter` independently while holding the others fixed at `par`. Useful as a post-optimization diagnostic: around a (local) minimum the OFV is expected to be roughly convex along each axis.

*Usage:*

```
ParameterIdentification$calculateOFVProfiles(
  par = NULL,
  boundFactor = 0.1,
  totalEvaluations = 20
)
```

*Arguments:*

`par` Numeric vector of parameter values, one for each `PIParameter`. Defaults to current parameter values if `NULL`, not numeric, or of mismatched length.

`boundFactor` Numeric scalar. A value of `0.1` (default) means bounds extend ±10% around `par` for each parameter.

`totalEvaluations` Integer specifying the number of grid points per parameter profile. Default is `20`.

*Details:* For each parameter `i` a one-dimensional grid of `totalEvaluations` equally spaced points is built between `lower[i]` and `upper[i]`, with the bounds derived from `par[i]` and `boundFactor`:

- par[i] >= 0: lower[i] = (1 - boundFactor) * par[i], upper[i] = (1 + boundFactor) * par[i].
- par[i] < 0: bounds are mirrored so that lower < upper is preserved.

The objective function is evaluated along each axis with all other parameters held at their par values. Failed simulations contribute Inf to the corresponding ofv cell.

*Returns:* A named list of tibbles, one element per PIParameter. List names are the parameter paths (taken from parameters[[1]]$path). Each tibble has two columns:

- a column named after the parameter path, holding the grid values;
- ofv, holding the matching objective function values.

Pass the returned list to plotOFVProfiles() to visualize the profiles.

*Examples:*

```
# piTask is a configured ParameterIdentification instance
# Default: +/-10% around current values, 20 grid points per parameter
# ofvProfiles <- piTask$calculateOFVProfiles()

# Wider neighborhood, finer grid
# ofvProfiles <- piTask$calculateOFVProfiles(
#   boundFactor = 0.5,
#   totalEvaluations = 50
# )

# plotOFVProfiles(ofvProfiles)[[1]]
```

**ParameterIdentification$print():** Prints a summary of ParameterIdentification instance.

*Usage:*

```
ParameterIdentification$print()
```

## Examples

```
## ------------------------------------------------
## Method `ParameterIdentification$calculateOFVProfiles()`
## ------------------------------------------------

# piTask is a configured ParameterIdentification instance
# Default: +/-10% around current values, 20 grid points per parameter
# ofvProfiles <- piTask$calculateOFVProfiles()

# Wider neighborhood, finer grid
# ofvProfiles <- piTask$calculateOFVProfiles(
#   boundFactor = 0.5,
#   totalEvaluations = 50
# )

# plotOFVProfiles(ofvProfiles)[[1]]
```

PIConfiguration *PIConfiguration*

## Description

Encapsulates configurations such as optimization algorithm choice, and evaluation settings for parameter identification.

## Active bindings

`printEvaluationFeedback` Boolean. If `TRUE`, prints objective function value after each evaluation. Default is `FALSE`.

`simulationRunOptions` Object of `SimulationRunOptions` for simulation runs. If `NULL`, default options are used.

`objectiveFunctionOptions` Settings for model fit evaluation, affecting error metrics and cost calculation. Defaults in `ObjectiveFunctionOptions`. Partial lists are merged with the current settings; only the provided keys are updated and validated.

`algorithm` Optimization algorithm name. See `Algorithms` for a list of supported algorithms. Defaults to `BOBYQA`. Changing the algorithm resets `algorithmOptions` to the new algorithm's defaults.

`ciMethod` Confidence interval estimation method. See `CIMethods` for available options. Defaults to `hessian`. Changing the method resets `ciOptions` to the new method's defaults.

`algorithmOptions` Named list of user-defined overrides for the selected algorithm's settings. Returns `NULL` if no overrides are set; in that case algorithm defaults (`AlgorithmOptions_XYZ`) are used automatically. Partial lists are merged with previously stored overrides. Unknown keys produce a warning and are ignored. Set to `NULL` to clear all overrides.

`ciOptions` Named list of settings for the selected CI method. Refer to `CIOptions` for default settings per method (e.g., `CIOptions_XYZ` where `XYZ` corresponds to the method name). If `NULL`, CI method's default settings are returned. Partial lists are merged with the current settings; only the provided keys are validated. Unknown keys produce a warning and are ignored. Set to `NULL` to reset to defaults.

`autoEstimateCI` Logical. If `TRUE`, confidence intervals are automatically estimated after optimization. If `FALSE`, the step is skipped and can be triggered manually by calling the `estimateCI()` method on the `ParameterIdentification` object.

`modelCostField` Read-only field name in the cost object used as the optimization target. Currently, only `modelCost` is supported.

## Methods

### Public methods:

- `PIConfiguration$new()`
- `PIConfiguration$print()`
- `PIConfiguration$clone()`

`PIConfiguration$new()`: Initialize a new instance of the class.

*Usage:*

```
PIConfiguration$new()
```

*Returns:* A new `PIConfiguration` object.

`PIConfiguration$print()`: Prints a summary of the `PIConfiguration`.

*Usage:*

```
PIConfiguration$print()
```

`PIConfiguration$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
PIConfiguration$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`PIOutputMapping` *PIOutputMapping*

---

## Description

Establishes connections between simulated quantities and corresponding observed data sets. Utilized within `ParameterIdentification` instances to align and compare simulation outputs with empirical data.

## Active bindings

`observedDataSets` A named list containing `DataSet` objects for comparison with simulation outcomes.

`dataTransformations` A named list of factors and offsets.

`dataWeights` A named list of y-value weights.

`quantity` Simulation quantities to be aligned with observed data values.

`simId` Identifier of the simulation associated with the mapped quantity.

`scaling` Specifies scaling for output mapping: linear (default) or logarithmic.

`transformResultsFunction` A function to preprocess simulated results (time and observation values) before residual calculation. It takes numeric vectors `xVals` and `yVals`, and returns a named list with keys `xVals` and `yVals`.

## Methods

### Public methods:

- PIOutputMapping$new()
- PIOutputMapping$addObservedDataSets()
- PIOutputMapping$removeObservedDataSet()
- PIOutputMapping$setDataTransformations()
- PIOutputMapping$setDataWeights()
- PIOutputMapping$print()
- PIOutputMapping$clone()

`PIOutputMapping$new()`: Initialize a new instance of the class.

*Usage:*

```
PIOutputMapping$new(quantity)
```

*Arguments:*

`quantity` An object of the type `Quantity`.

*Returns:* A new `PIOutputMapping` object.

`PIOutputMapping$addObservedDataSets()`: Adds or updates observed data using `DataSet` objects.

*Usage:*

```
PIOutputMapping$addObservedDataSets(data, weights = NULL)
```

*Arguments:*

`data` A `DataSet` object or a list thereof, matching the simulation quantity dimensions.

`weights` A named list of numeric values or numeric vectors. The names must match the names of the observed datasets.

*Details:* Replaces any existing dataset with the same label.

`PIOutputMapping$removeObservedDataSet()`: Removes specified observed data series.

*Usage:*

```
PIOutputMapping$removeObservedDataSet(label)
```

*Arguments:*

`label` The label of the observed data series to remove.

`PIOutputMapping$setDataTransformations()`: Configures transformations for datasets.

*Usage:*

```
PIOutputMapping$setDataTransformations(
  labels = NULL,
  xOffsets = 0,
  yOffsets = 0,
  xFactors = 1,
  yFactors = 1
)
```

*Arguments:*

`labels` List of dataset labels for targeted transformations. Absence of labels applies transformations globally.

`xOffsets` Numeric list/value for X-offset adjustments.

`yOffsets` Numeric list/value for Y-offset adjustments.

`xFactors` Numeric list/value for X-scaling factors.

`yFactors` Numeric list/value for Y-scaling factors.

`PIOutputMapping$setDataWeights()`: Assigns weights to observed data sets for residual weighting during parameter identification.

*Usage:*

```
PIOutputMapping$setDataWeights(weights)
```

*Arguments:*

`weights` A named list of numeric values or numeric vectors. The names must match the names of the observed datasets.
Each element in the list can be:

- a scalar, which will be broadcast to all y-values of the corresponding dataset,
- or a numeric vector matching the number of y-values for that dataset.

To apply both dataset-level and point-level weights, multiply them beforehand and provide the combined result as a single numeric vector per dataset.

`PIOutputMapping$print()`: Prints a summary of the PIOutputMapping.

*Usage:*

```
PIOutputMapping$print()
```

`PIOutputMapping$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
PIOutputMapping$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

---

`PIParameters` *PIParameters*

---

## Description

A parameter to be optimized in a parameter identification routine

## Active bindings

`parameters` A list of parameter objects. Adding or removing parameters is not supported.

`currValue` The current parameter values for simulations, in units specified by `$unit`.

`startValue` Initial value used for optimization.

`minValue` Lowest permissible parameter value.

`maxValue` Highest permissible parameter value.

`unit` Parameter value units. Changing the unit does NOT automatically adjust min/max/start values.

## Methods

### Public methods:

- PIParameters$new()
- PIParameters$setValue()
- PIParameters$toDataFrame()
- PIParameters$print()
- PIParameters$clone()

`PIParameters$new()`: Initialize a new instance of the class. The initial start value is derived from the first parameter upon creation, modifiable via `PIParameters$startValue`. All parameters are optimized using this unified value.

*Usage:*

```
PIParameters$new(parameters)
```

*Arguments:*

`parameters` List of `Parameter` class objects to be optimized.

*Returns:* A new `PIParameters` object.

`PIParameters$setValue()`: Updates parameter(s) value. Value is specified in units of `$unit`.

*Usage:*

```
PIParameters$setValue(value)
```

*Arguments:*

`value` Numeric value to set.

`PIParameters$toDataFrame()`: Export parameter metadata and configuration to a data frame. Returns one row per internal model parameter wrapped by the `PIParameters` object.

*Usage:*

```
PIParameters$toDataFrame(group = NULL)
```

*Arguments:*

`group` Optional character label identifying the optimization group.

*Returns:* A `data.frame` with one row per internal parameter.

`PIParameters$print()`: Prints a summary of the PIParameters.

*Usage:*

```
PIParameters$print()
```

`PIParameters$clone()`: The objects of this class are cloneable with this method.

*Usage:*

```
PIParameters$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

PKOutputMapping *PKOutputMapping*

## Description

Maps a simulation output quantity to a target PK parameter for use in ParameterIdentification optimization.

## Active bindings

`quantity` The Quantity object from the simulation. Read-only.

`simId` ID of the root simulation container. Read-only.

`pkParameter` Name of the PK parameter to target. Must be one of ospsuite::StandardPKParameter.

`targetValueInBaseUnit` Target value in base units. Computed from `targetValue` and `targetUnit`. Read-only.

`targetValue` Numeric target value in units of `targetUnit`. Recomputes `targetValueInBaseUnit` on assignment.

`targetUnit` Unit string for `targetValue`. Must be compatible with the PK parameter's dimension. Recomputes `targetValueInBaseUnit` on assignment.

## Methods

### Public methods:

- PKOutputMapping$new()
- PKOutputMapping$print()
- PKOutputMapping$clone()

`PKOutputMapping$new()`: Initialize a new PKOutputMapping.

*Usage:*

```
PKOutputMapping$new(quantity, pkParameter, targetValue, targetUnit)
```

*Arguments:*

`quantity` An ospsuite Quantity object from the simulation whose PK profile will be analyzed.

`pkParameter` Character string specifying the PK parameter to target. Must be one of ospsuite::StandardPKParameter (e.g., "C_max", "AUC_tEnd").

`targetValue` Numeric scalar target value in units of `targetUnit`.

`targetUnit` Character string specifying the unit of `targetValue`. Must be dimensionally compatible with `pkParameter`.

*Returns:* A new PKOutputMapping object.

*Examples:*

```
sim <- ospsuite::loadSimulation("model.pkml")
quantity <- ospsuite::getQuantity(
  "Organism|PeripheralVenousBlood|Drug|Plasma (Peripheral Venous Blood)",
  container = sim
)
mapping <- PKOutputMapping$new(
  quantity = quantity,
  pkParameter = "C_max",
  targetValue = 500,
  targetUnit = "nmol/l"
)
```

`PKOutputMapping$print()`**:** Print a summary of the `PKOutputMapping`.

*Usage:*

```
PKOutputMapping$print()
```

*Returns:* Called for its side effect of printing. Returns `invisible(self)`.

`PKOutputMapping$clone()`**:** The objects of this class are cloneable with this method.

*Usage:*

```
PKOutputMapping$clone(deep = FALSE)
```

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
## ------------------------------------------------
## Method `PKOutputMapping$new()`
## ------------------------------------------------

## Not run:
sim <- ospsuite::loadSimulation("model.pkml")
quantity <- ospsuite::getQuantity(
  "Organism|PeripheralVenousBlood|Drug|Plasma (Peripheral Venous Blood)",
  container = sim
)
mapping <- PKOutputMapping$new(
  quantity = quantity,
  pkParameter = "C_max",
  targetValue = 500,
  targetUnit = "nmol/l"
)

## End(Not run)
```

## plot.modelCost *Plot Model Cost Residuals*

### Description

Plots raw residuals and, if different, weighted and robust weighted residuals from a `modelCost` object.

### Usage

```
## S3 method for class 'modelCost'
plot(x, legpos = "topright", ...)
```

### Arguments

| | |
|---|---|
| `x` | A `modelCost` object containing residuals to plot. |
| `legpos` | Position of the legend; default is "topright". Use NA to omit the legend. |
| `...` | Additional arguments passed to the plot function. |

### Value

Generates a plot.

### Examples

```
# Assuming modelCostObj is a valid `modelCost` object
## Not run:
plot.modelCost(modelCostObj)

## End(Not run)
```

## plotOFVProfiles *Plot Objective Function Value (OFV) Profiles*

### Description

Visualizes the OFV profiles produced by `ParameterIdentification$calculateOFVProfiles()`. Each profile is rendered as a scatter plot of the OFV against the corresponding parameter value, with point color encoding OFV (lower = brighter, reversed Viridis).

### Usage

```
plotOFVProfiles(profiles)
```

### Arguments

| | |
|---|---|
| `profiles` | A named list of tibbles as returned by `ParameterIdentification$calculateOFVProfiles()`. Each tibble must contain exactly two columns: the parameter values (named after the parameter's path) and the matching objective function values (named `ofv`). The list names must match the parameter-value column names. |

### Details

Around a (local) minimum the objective function is expected to be convex and roughly parabolic. Inspecting the OFV profile around a tentative optimum is therefore a quick visual check that the optimization has converged to a sensible point in parameter space.

Because parameter paths in OSP simulations can be long (e.g. `Neighborhoods|Kidney_pls_Kidney_ur|Aciclovir|Renal`) the x-axis label uses only the last `|`-separated segment, while the full path is shown as the plot title.

### Value

A list of `ggplot` objects, one per parameter profile, in the same order as `profiles`.

### See Also

ParameterIdentification for `calculateOFVProfiles()`.

### Examples

```
## Not run:
# piTask is a configured ParameterIdentification instance
ofvProfiles <- piTask$calculateOFVProfiles()
plots <- plotOFVProfiles(ofvProfiles)
plots[[1]]

## End(Not run)
```

---

## `residualWeightingOptions` *Residual Weighting Methods for Cost Function*

---

### Description

Methods for weighting residuals in the model cost function, affecting how discrepancies between simulations and observed data are evaluated.

### Usage

```
residualWeightingOptions
```

### Details

The methods include:

- `none` – No weighting applied, treating all residuals equally.
- `error` – Weights based on error estimates for the dependent variable in observed data.

---

## `robustMethodOptions` *Robust Weighting Methods for Cost Function*

---

### Description

Robust weighting methods to address outliers in the cost function calculation during parameter optimization.

### Usage

```
robustMethodOptions
```

### Details

The available methods are:

- `none` – No robust weighting applied.
- `huber` – Huber weighting for moderate outliers.
- `bisquare` – Bisquare (Tukey's biweight) weighting for severe outliers.

---

## `ScalingOptions` *Scaling Options for Output Mapping*

---

### Description

Scaling options applicable to output mappings, impacting how simulation results are compared to observed data.

### Usage

```
ScalingOptions
```

### Details

Available scaling options:

- `lin` – Linear scaling (default).
- `log` – Logarithmic scaling, used when data spans several orders of magnitude.

# Index