reBoot: Manual

Jonny Proppe and Markus Reiher

Version: 2017-06-06

ETH Zurich, Laboratory of Physical Chemistry
Vladimir-Prelog-Weg 2, 8093 Zurich, Switzerland

https://github.com/reihergroup/reboot
http://www.reiher.ethz.ch/software/reboot.html
reboot@phys.chem.ethz.ch

1 Welcome!

reBoot is a toolbox for statistical calibration of property models. In this version, we support polynomial models based on a single scalar input variable, x,

$$f_M(x, \mathbf{w}) = \sum_{m=0}^{M} w_m x^m = \mathbf{x}^\top \mathbf{w}\,, \tag{1}$$

where M is the polynomial degree, $\mathbf{w} = (w_0, w_1, ..., w_M)^\top$ is the vector of parameters, and $\mathbf{x} = (1, x, x^2, ..., x^M)^\top$ is the polynomial vector of x.

As the model is linear in its parameters, all calibration procedures implemented in the reBoot program are currently based on different types of linear regression, i.e., ordinary least squares (OLS), weighted least squares (WLS), iteratively reweighted least squares (IRLS), and regularized least squares (regLS).

For the determination of model prediction uncertainty (MPU), we currently provide nonparametric bootstrapping, k-fold cross-validation, and the evidence approximation to Bayesian inference (based on the normal-population assumption).

Note that the statistical methods implemented in reBoot are not limited to single-variable polynomial models that are linear with respect to their parameters. For instance, implementation of non-polynomial models, many-variable models, or models being non-linear in their parameters is straightforward.

For a detailed discussion on the statistical calibration of property models employed in a scientific context, see our paper (Ref. 1). We also recommend excellent discussions on this topic by other groups (see Refs. 2, 3, 4, 5, 6).

2 Download and Installation

Access to the reBoot program is provided through our group webpage (`http://www.reiher.ethz.ch/software/reboot.html`) or our GitHub page (`https://github.com/reihergroup/reboot`). The entire program is based on scripts written in the GNU Octave[7] programming language that is mostly compatible with MATLAB. Additional packages for GNU Octave are not required. All that needs to be done is to clone the reBoot repository from our GitHub page to a local directory (say, `bootDir`), and to set a path to that directory by typing `addpath('absolute-path-to-bootDir')` in the GNU Octave shell (a permanent alternative is to append the `.octaverc` file in the home directory by the absolute path to `bootDir`).

3 A Short Dictionary

Table 1: Terms and definitions frequently used in this reBoot manual.

term	definition
target observable	the quantity to be predicted, y
target uncertainty	the uncertainty of the target-observable measurement / prediction
input variable	the quantity mapped to the target observable by a property model, x
(property) model	a parametric function, $f(x, \mathbf{w}) \approx y$, that maps x to y; it needs to be calibrated (estimation of model parameters, $\mathbf{w}$) against reference data
reference (data) set	the data set employed for the calibration of a property model, $\mathcal{D} \equiv \{(x_n, y_n)\}$ or $\mathcal{D} \equiv \{(x_n, y_n, u_n)\}$ with $n = 1, ..., N$
data pair	a pair of reference data, (x_n, y_n)
data triple	a triple of reference data, (x_n, y_n, u_n)
input-generating method	the computational method that generates the reference values of the input variable (e.g., a density functional[1,4])

4 Essential Variables

Table 2: Essential reBoot variables and their definitions.

variable	definition
`N`	number of data pairs (triples) in the reference set (default is `N = 100` if `inputOpt.random = true`)
`M`	polynomial degree (default is `M = 1`); `M + 1` is the number of model parameters

`dimInput`	number of input-generating methods (determined from the `data.txt` file; if `inputOpt.random` = `true` or `inputOpt.multipleInputs` = `false`, `dimInput` = `1`)
`x`	column vector or matrix of the input variable (dimension `N` × `dimInput`)
`X`	design matrix (dimension `N` × (`M` + `1`)); the n-th entry in the m-th column refers to x_n^{m-1}
`y`	column vector of the target observable (dimension `N` × `1`)
`u`	column vector of the target uncertainty (dimension `N` × `1`)
`B`	number of bootstrap samples (default is `B` = `1000`; minimum number is `B` = `100`)
`lsrType`	specifies the type of linear least-squares (LS) regression; the following values can be chosen: `@OLS` (default); `@WLS`; `@IRLS`; `@regLS`
`blrMode`	specifies the sophistication level of Bayesian linear regression; the following values can be chosen: `0`, equivalent to ordinary OLS regression plus parameter covariance; `1`, takes initial guess of evidence approximation; `2` (default), takes converged evidence approximation
`inputOpt`	a structure that holds the variables `multipleInputs`, `expUncertainty`, `allowOnlyEU`, `critical`, and `randomData`.
`inputOpt.multipleInputs`	if `false`, `dimInput` = `1` is assumed and the variable `inputID` (default is `1`) needs to be defined; if `true`, `dimInput` is determined from the `data.txt` file; ignored if `inputOpt.randomData` = `true`
`inputOpt.expUncertainty`	if `false`, the user confirms that the last column in the `data.txt` file contains values of the target observable; if `true`, the user confirm that the last column in the `data.txt` file contains values of the target uncertainty; ignored if `inputOpt.randomData` = `true`
`inputOpt.allowOnlyEU`	if `false`, all rows in the `data.txt` will be considered, and all zero target uncertainties will be transformed to the mean value of all non-zero target uncertainties; if `true`, only those rows in the `data.txt` will be considered that correspond to non-zero target uncertainties; ignored if `inputOpt.randomData` = `true` or `inputOpt.expUncertainty` = `false`

`inputOpt.critical`	if `false`, all rows in the `data.txt` will be considered; if `true`, only those rows in the `data.txt` file will be considered that are not listed in the `critical.txt` file (for details, see the section introducing the `critical.txt` file); ignored if `inputOpt.randomData` = `true`
`inputOpt.randomData`	if `false`, the reference data set will be read from the `data.txt` file and processed by the `userData.m` script; if `true`, random reference data are generated by the `randomData.m` script
`inputID`	if `dimInput` $>$ `1`, but `option.multipleInputs` = `false` is chosen, the user can select those values of the input variable that were obtained from a specific input-generating method (`1` $\leq$ `inputID` $\leq$ `dimInput`)
`calOpt`	a structure that holds the variables `xScale`, `resolution`, `increase`, `bootDetail`, `bayesConv`, `bayesMaxIter`, `irlsConv`, `irlsMaxIter`, and `reglsPenalty`
`calOpt.xScale`	transforms the input variable; the following values can be chosen: `0`, `x` = `x`; `1` (default), `x` = `x` − `mean(x)`; `2`, `x` = (`x` − `mean(x)`)/`std(x)`
`calOpt.resolution`	determines the smallest significant figure that can be resolved according to $10^{-\texttt{calOpt.resolution}}$ (default is `calOpt.resolution` = `[]`); example A: if `y` = `198.4378` and `calOpt.resolution` = `1`, then the user asserts that `y` can be resolved up to the first decimal place (i.e., `198.4`); example B: if `y` = `198.4378` and `calOpt.resolution` = -2, then the user asserts that `y` can be resolved only up to the third place before the decimal point (i.e., `200`); if `calOpt.resolution` = `[]` (empty), it will not affect `y`; `calOpt.resolution` is only activated through application of the `roundResult.m` script
`calOpt.increase`	`calOpt.resolution` is increased by this positive integer when applying the `rankEval.m` script (default is `calOpt.increase` = `0`)
`calOpt.bootDetail`	specifies the detail of bootstrapping; if `false`, a standard bootstrapping procedure will be performed; if `true` (default), jackknife-after-bootstrapping will be additionally performed (necessary to obtain the R632 estimate of MPU

`calOpt.bayesConv`	a positive real number that specifies convergence of the evidence approximation (default is $\texttt{calOpt.bayesConv} = 10^{-3}$); if the relative change of the model prediction variance is smaller than `calOpt.bayesConv` between two iterations, the algorithm stops
`calOpt.bayesMaxIter`	a positive integer that specifies the maximum number of iterations in the evidence approximation (default is $\texttt{calOpt.bayesMaxIter} = 100$)
`calOpt.irlsConv`	a positive real number that specifies convergence of IRLS regression (default is $\texttt{calOpt.irlsConv} = 10^{-3}$); if the relative change of the squared discrepancy is smaller than `calOpt.irlsConv` between two iterations, the algorithm stops
`calOpt.irlsMaxIter`	a positive integer that specifies the maximum number of iterations in IRLS regression (default is $\texttt{calOpt.irlsMaxIter} = 100$)
`calOpt.reglsPenalty`	a positive real number that specifies the penalty factor in regLS regression (default is $\texttt{calOpt.reglsPenalty} = 10^{-3}$)
`calPlot`	plots will (`true`) or will not (`false`, default) be generated when applying the `calibration.m` script or the `rankEval.m` script

5 Input Processing

The `init.m` script

The first script that needs to be executed in a `reBoot` session. It clears previous sessions, provides information on the `reBoot` toolbox, and starts the input processing.

The `control.m` script

The `control.m` script defines many essential variables (default values) if not already done by the user via the `setting.m` script, and partially checks whether definitions made by the user (`setting.m`) are erroneous.

The `setting.m` script

With the `setting.m` file, the user can change the default values of many essential variables. All essential scripts of the `reBoot` program will check whether definitions made

the user are erroneous. The `setting.m` file is not provided by us, it can be optionally created in the working directory.

The `userData.m` script

If `inputOpt.randomData` = `false`, reference data will be read from the `data.txt` file and processed by this script. It sorts the rows in the `data.txt` file according to the descending order of the target-observable values. If `inputOpt.critical` = `true`, those data pairs (triples) will be removed (after sorting) whose indices are listed in the `critical.txt` file.

The `randomData.m` script

If `inputOpt.randomData` = `true`, reference data will be randomly generated by this script.

The `data.txt` file

The `data.txt` file holds the data pairs (triples) of the reference set. We define the number of data pairs (triples) as `N` and the number of input-generating methods as `dimInput`. Given `dimInput` = `1`, each line in the `data.txt` file refers to a data pair (triple), with `N` lines in total.

$$\begin{array}{cc} \texttt{x(1)} & \texttt{y(1)} \\ \vdots & \vdots \\ \texttt{x(N)} & \texttt{y(N)} \end{array} \quad \text{or} \quad \begin{array}{ccc} \texttt{x(1)} & \texttt{y(1)} & \texttt{u(1)} \\ \vdots & \vdots & \vdots \\ \texttt{x(N)} & \texttt{y(N)} & \texttt{u(N)} \end{array}$$

Given `dimInput` $>$ `1`, all input values corresponding to a target value are listed first in a line, followed by the target value (and, optionally, the target uncertainty).

$$\begin{array}{ccccc} \texttt{x(1,1)} & \cdots & \texttt{x(1,dimInput)} & \texttt{y(1)} \\ \vdots & \ddots & \vdots & \vdots \\ \texttt{x(N)} & \cdots & \texttt{x(N,dimInput)} & \texttt{y(N)} \end{array} \quad \text{or} \quad \begin{array}{ccccc} \texttt{x(1,1)} & \cdots & \texttt{x(1,dimInput)} & \texttt{y(1)} & \texttt{u(1)} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ \texttt{x(N)} & \cdots & \texttt{x(N,dimInput)} & \texttt{y(N)} & \texttt{u(N)} \end{array}$$

Example: We choose `N` = `3` and `dimInput` = `2`. With the first input-generating method, we obtain the values `198, 124, 260`, and with the second input-generating method, we obtain the value `207, 125, 297`. The corresponding values of the target observable read `9.6, 7.9, 11.1`, along with their uncertainties `0.4, 0.1, 0.3`.

$$\begin{array}{ccc} 198 & 207 & 9.6 \\ 124 & 125 & 7.9 \\ 260 & 297 & 11.1 \end{array} \quad \text{or} \quad \begin{array}{cccc} 198 & 207 & 9.6 & 0.4 \\ 124 & 125 & 7.9 & 0.1 \\ 260 & 297 & 11.1 & 0.3 \end{array}$$

Note: The `data.txt` file must be placed in the working directory. The content of the `data.txt` file must be a table of integers and/or real numbers. We recommend that all target values (and the corresponding uncertainties) possess the same resolution. If the uncertainty is unknown for some of the target values, but it is still desired to work with data triples, the uncertainty value `0` should be assigned in such cases.

The `critical.txt` file

If `inputOpt.randomData` = `false` and `inputOpt.critical` = `true`, those data pairs (triples) will be removed whose indices are listed in this file. The indices in the `critical.txt` file must refer to the reference set where the target-observable values are sorted in descending order. Given the example above, the new reference set reads

```
260  297  11.1          260  297  11.1  0.3
198  207   9.6    or    198  207   9.6  0.4
124  125   7.9          124  125   7.9  0.1
```

We create a `critical.txt` file with the following content

```
3
```

The adjusted reference set now reads

```
260  297  11.1          260  297  11.1  0.3
198  207   9.6    or    198  207   9.6  0.4
```

In the `critical.txt` file, indices can be provided line by line or in one line with whitespace separation.

6 Simple Calibration / Linear LS Regression

In the following we introduce functions that yield point estimates of model parameters by means of different types of linear LS regression.

The `LSR.m` script

The function `LSR(x,y,u,M,lrsType,calOpt)` yields a point estimate of model parameters depending on the `lsrType` chosen (cf. the corresponding entry in Table 2). We recommend to use this script to select one of the four functions presented below.

The `OLS.m` script

The function `OLS(data)` yields a point estimate of model parameters by means of OLS regression. The structure `data` is automatically constructed by the `LSR` function, which is why we recommend to execute the latter with `lsrType` = @`OLS`.

The `WLS.m` script

The function `WLS(data)` yields a point estimate of model parameters by means of WLS regression. The structure `data` is automatically constructed by the `LSR` function, which is why we recommend to execute the latter with `lsrType` = @`WLS`.

The `IRLS.m` script

The function `IRLS(data,calOpt)` yields a point estimate of model parameters by means of IRLS regression. The structure `data` is automatically constructed by the `LSR` function, which is why we recommend to execute the latter with `lsrType` = @`IRLS`.

The `regLS.m` script

The function `regLS(data,calOpt)` yields a point estimate of model parameters by means of regLS regression. The structure `data` is automatically constructed by the `LSR` function, which is why we recommend to execute the latter with `lsrType` = @`regLS`.

7 Statistical Calibration / MPU Estimation

In the following we introduce functions that yield distributions of model parameters by means of different types of statistical calibration based on linear LS regression.

The `bootCal.m` script

The function `bootCal(x,y,u,M,B,calOpt)` samples model parameters by the bootstrapping method. It returns mean and covariance estimates of the model parameters as well as MPU estimates (REMSE and R632, the latter by choosing `calOpt.bootDetail` = `1`). If the number of ouput arguments equals `2`, the `bootCal` function provides all `B` bootstrapped parameter vectors (necessary input for the `bootHist` function). If additionally `calOpt.bootDetail` = `1`, it returns `N` mean and covariances estimates obtained from the jackknife-after-bootstrapping method. The `bootCal` function also measures the wall time of the bootstrap procedure. Currently, only bootstrapped OLS regression is available in `reBoot`.

The `bayesCal.m` script

The function `bayesCal(x,y,u,M,blrMode,calOpt)` calculates Gaussian distributions of model parameters by Bayesian linear regression. It returns mean and covariance estimates of the model parameters as well as an MPU estimate (RMPV). Based on the `blrMode` value, the sophistication level of Bayesian inference can be adjusted (cf. the corresponding entry in Table 2). Currently, weighted Bayesian regression is not available in `reBoot`.

The `LOOCV.m` script

The function `LOOCV(x,y,u,M,calOpt)` samples model parameters by the leave-one-out cross-validation (LOOCV) method. It returns mean and covariance estimates of the model parameters as well as an MPU estimate (RLOO). Note that the model-parameter estimates are generally poor compared to those obtained from bootstrapping and are provided only for the sake of completeness. Additionally, if the number of output arguments

equals 2, the LOOCV function returns N point estimates of model parameters obtained from the standard jackknife method based on OLS regression. Currently, the LOOCV method is only available in combination with OLS regression.

The calibration.m script

The function calibration(x,y,u,M,B,calOpt,calPlot) performs all statistical calibration procedures at once. For the sake of sophistication, it sets calOpt.bootDetail = 1 and blrMode = 2. The calibration function returns four MPU estimates (RMSE, R632, RMPV, RLOO), where the RMSE is obtained from OLS regression. If calPlot = 1, the calibration function returns plots that visualize the calibration results.

8 Prediction

Having completed all calibration procedures, it is time to actually make predictions of the target observable (and, optionally, the target uncertainty / locally resolved MPU).

The predict.m script

The function predict(x0,model,calOpt) takes one or multiple new input value(s), x0, and a model from one of the simple / statistical calibration procedures introduced above. It returns (a) prediction(s) for the target observable (and, optionally, the target uncertainty) at x0.

9 Performance Rankings

If dimInput > 1, i.e., if input values to more than one input-generating method are provided, it is possible to determine performance rankings. The reBoot toolbox allows to study the dependence of such performance rankings on the number and composition of reference data.

The rankEval.m script

The function rankEval(x,y,u,M,list,B,calOpt,calPlot) yields performance rankings for dimInput input-generating methods (determined by the number of columns in x). For this purpose, it generates B synthetic (bootstrap) samples for each of the data-set sizes provided in the scalar or vectorial variable list. The rankings are based on both RMSE and RMPV. The resolution of the rankings can be controlled by the variables calOpt.resolution and calOpt.increase (cf. the corresponding entries in Table 2). If calPlot = 1, the rankEval function returns plots that visualize the percentage of first places the different input-generating methods reached.

10 Little Helpers

The `add.m` script

The function `add(x,M)` transforms the input vector `x` into a design matrix `X` with polynomial degree `M`.

The `bootHist.m` script

The function `bootHist(w,bayes)` takes an ensemble of bootstrapped parameters, `w`, obtained from the `bootCal.m` script and the results of the `bayesCal.m` script, `bayes`, and returns a bootstrapped parameter histogram as well as a smooth Gaussian posterior parameter distribution.

The `bootMean.m` script

The function `bootMean(z,B)` estimates bias and variance of the mean of a vectorial quantity `z` by drawing `B` bootstrap samples from it.

The `jackBayes.m` script

The function `jackBayes(x,y,u,M,blrMode,calOpt)` provides MPU estimates (RMPV) for all `N` jackknife samples of the reference set provided.

The `remove.m` script

The function `remove(z,list)` returns `z` with the rows specified in `list` removed. This functionality is important for removing inconsistent data points (cf. the section introducing the `critical.txt` file) and for the standard jackknife procedures implemented in the `LOOCV.m` and `jackBayes.m` scripts.

The `roundResult.m` script

The function `roundResult(z,resolution)` returns `z` with all elements rounded according to $10^{-\texttt{resolution}}$ (cf. entry `calOpt.resolution` in Table 2).

References

[1] Proppe, J.; Reiher, M. Reliable Estimation of Prediction Uncertainty for Physicochemical Property Models, *J. Chem. Theory Comput.* **2017,** DOI: 10.1021/acs.jctc.7b00235, arXiv:1703.01685.

[2] Kennedy, M. C.; O'Hagan, A. Bayesian Calibration of Computer Models, *J. R. Stat. Soc. B* **2001,** *63,* 425–464.

[3] Sargsyan, K.; Najm, H. N.; Ghanem, R. On the Statistical Calibration of Physical Models, *Int. J. Chem. Kinet.* **2015,** *47,* 246–276.

[4] Pernot, P.; Civalleri, B.; Presti, D.; Savin, A. Prediction Uncertainty of Density Functional Approximations for Properties of Crystals with Cubic Symmetry, *J. Phys. Chem. A* **2015,** *119,* 5288–5304.

[5] Pernot, P.; Cailliez, F. A Critical Review of Statistical Calibration/Prediction Models Handling Data Inconsistency and Model Inadequacy, **2016,** arXiv:1611.04376.

[6] Pernot, P. The Parameters Uncertainty Inflation Fallacy, **2016,** arXiv:1611.04295.

[7] Eaton, J. W.; Bateman, D.; Hauberg, S. *GNU Octave Version 3.0.1 Manual: A High-Level Interactive Language for Numerical Computations;* CreateSpace Independent Publishing Platform: 2009.