Imrob Documentation

Release 20180828

Jan 07, 2019

CONTENTS:

The lmrob package is a translation of the lmrob functions of the robustbase package of R code. The included functions are the following:

- lmrob
- lmrob_control
- lmrob_fit
- lmrob_S
- lmrob_D_fit
- lmrob_M_fit
- lmrob_M_S
- lmrob_lar

The nlrob package is a translation of the nlrob functions of the robustbase package of R code. The included functions are the following:

- nlrob
- nlrob_MM
- nlrob_tau
- nlrob_CM
- nlrob_mtl
- nls

CHAPTER

ONE

SYSTEM REQUIREMENTS

Additional to the repository the following packages must be in the system before installing the package.

- `lapack`
- `blas`
- `R`

CHAPTER

TWO

INSTALLATION

Installing the required packages

```
$ sudo apt install r-base -y
```

Install a virtualenv

```
sudo apt install virtualenv
```

Create virtualev

```
$ virtualenv -p python3 /path/to/env
```

Activate the virtualenv

```
$ source /path/to/env/bin/activate
```

Installing lmrob amd nlrob

```
(env) $ python setup.py install_lib
```

2.1 Issues

The following compilers must be installed in the system:

- gcc
- gfortran

If the `Python.h` is not found install python-devel to the system

CHAPTER

THREE

TESTING THE CODE

There are two kinds of test, unitary an advanced tests. To try the unitary test do the following steps

3.1 lmrob Unitary Test

1. **Move to tests directory**

```
(env) $ cd tests/unittest
```

2. **Try all the tests**

```
(env) $ python -m unittest test_lmrob -v
```

and you should get a the following

```
test_lm_fit (test_lmrob.LmrobTestCase) ... ok
test_lmrob (test_lmrob.LmrobTestCase) ... ok
test_lmrob_D_fit (test_lmrob.LmrobTestCase) ... ok
test_lmrob_M_S (test_lmrob.LmrobTestCase) ... ok
test_lmrob_M_fit (test_lmrob.LmrobTestCase) ... ok
test_lmrob_S (test_lmrob.LmrobTestCase) ... ok
test_lmrob_categorical (test_lmrob.LmrobTestCase) ... ok
test_lmrob_categorical_offset (test_lmrob.LmrobTestCase) ... ok
test_lmrob_categorical_offset_weights (test_lmrob.LmrobTestCase) ... ok
test_lmrob_categorical_weights (test_lmrob.LmrobTestCase) ... ok
test_lmrob_fit (test_lmrob.LmrobTestCase) ... ok
test_lmrob_lar (test_lmrob.LmrobTestCase) ... ok
test_lmrob_offset (test_lmrob.LmrobTestCase) ... ok
test_lmrob_offset_weights (test_lmrob.LmrobTestCase) ... ok
test_lmrob_tau (test_lmrob.LmrobTestCase) ... ok
test_lmrob_weights (test_lmrob.LmrobTestCase) ... ok
test_ghq (test_lmrob.LmrobUtilities) ... ok
test_lmrob_hat (test_lmrob.LmrobUtilities) ... ok
test_lmrob_kappa (test_lmrob.LmrobUtilities) ... ok
test_lmrob_rweights (test_lmrob.LmrobUtilities) ... ok
test_mchi (test_lmrob.LmrobUtilities) ... ok
test_mpsi (test_lmrob.LmrobUtilities) ... ok
test_mwgt (test_lmrob.LmrobUtilities) ... ok
```

3.2 lmrob Advanced Test

1. **Move to tests directory**

```
(env) $ cd tests/advancedtest/
```

2. Try all the tests

```
(env) $ cd tests/advancedtest/
```

3. Run the following command

```
(env) $ ./test_lmrob.sh
```

4. Getting something like that

```
METHOD: S
+----------------+---------------------+-------------+
| R Coefficients | Python Coefficients |    Diff     |
+----------------+---------------------+-------------+
|   -36.92541589 |        -36.92541690 | 1.00995e-06 |
|     0.84957481 |          0.84957481 | 1.51956e-10 |
|     0.43047400 |          0.43047399 | 1.38134e-08 |
|    -0.07353895 |         -0.07353894 | 1.51891e-08 |
+----------------+---------------------+-------------+
|  R residuals   |  Python residuals   |    Diff     |
+----------------+---------------------+-------------+
|     5.88160019 |          5.88160023 | 3.48700e-08 |
|     0.80806118 |          0.80806125 | 6.27125e-08 |
|     6.06396134 |          6.06396129 | 4.22755e-08 |
|     8.31829084 |          8.31829086 | 1.93947e-08 |
|    -0.82076106 |         -0.82076109 | 3.12389e-08 |
|    -1.25123511 |         -1.25123512 | 5.92210e-09 |
|    -0.24047511 |         -0.24047525 | 1.47660e-07 |
|     0.75952489 |          0.75952475 | 1.47660e-07 |
|    -0.85293589 |         -0.85293589 | 4.85767e-09 |
|    -0.21533869 |         -0.21533863 | 6.34561e-08 |
|     0.44651239 |          0.44651220 | 1.87127e-07 |
|    -0.19655257 |         -0.19655275 | 1.84601e-07 |
|    -3.06826068 |         -3.06826067 | 7.77104e-09 |
|    -1.68980562 |         -1.68980590 | 2.73180e-07 |
|     1.24311084 |          1.24311065 | 1.84998e-07 |
|     0.02249381 |          0.02249371 | 1.01470e-07 |
|    -0.43752638 |         -0.43752606 | 3.13642e-07 |
|     0.07724669 |          0.07724681 | 1.18744e-07 |
|     0.72031165 |          0.72031176 | 1.16218e-07 |
|     1.76994083 |          1.76994089 | 5.89368e-08 |
|    -9.46225536 |         -9.46225556 | 1.95371e-07 |
+----------------+---------------------+-------------+

METHOD: MM
+----------------+---------------------+-------------+
| R Coefficients | Python Coefficients |    Diff     |
+----------------+---------------------+-------------+
|   -41.52461665 |        -41.52461530 | 1.35669e-06 |
|     0.93884534 |          0.93884535 | 9.05454e-09 |
|     0.57955324 |          0.57955314 | 1.02435e-07 |
```

(continues on next page)

(continued from previous page)

```
|    -0.11292183 |         -0.11292182 | 7.40542e-10 |
+----------------+---------------------+-------------+
|  R residuals   |  Python residuals   |    Diff     |
+----------------+---------------------+-------------+
|     2.81909429 |          2.81909491 | 6.18791e-07 |
|    -2.29382753 |         -2.29382691 | 6.19531e-07 |
|     3.78534931 |          3.78534976 | 4.58453e-07 |
|     7.23112652 |          7.23112700 | 4.75948e-07 |
|    -1.60976700 |         -1.60976673 | 2.71078e-07 |
|    -2.18932024 |         -2.18931987 | 3.73513e-07 |
|    -1.09134253 |         -1.09134206 | 4.71505e-07 |
|    -0.09134253 |         -0.09134206 | 4.71505e-07 |
|    -1.43393887 |         -1.43393846 | 4.09731e-07 |
|    -0.32662546 |         -0.32662556 | 9.72605e-08 |
|     0.68967097 |          0.68967087 | 1.03925e-07 |
|     0.15630238 |          0.15630218 | 2.05620e-07 |
|    -3.10078181 |         -3.10078191 | 9.87416e-08 |
|    -1.43819497 |         -1.43819497 | 4.45245e-09 |
|     2.20043371 |          2.20043368 | 3.14890e-08 |
|     0.86166823 |          0.86166820 | 2.92674e-08 |
|    -0.29879056 |         -0.29879048 | 8.35353e-08 |
|     0.49166222 |          0.49166230 | 7.83515e-08 |
|     1.02503080 |          1.02503098 | 1.80046e-07 |
|     1.61780240 |          1.61780252 | 1.24238e-07 |
|   -10.50973597 |        -10.50973598 | 9.19078e-09 |
+----------------+---------------------+-------------+

METHOD: SMD
+----------------+---------------------+-------------+
| R Coefficients | Python Coefficients |    Diff     |
+----------------+---------------------+-------------+
|   -41.76557868 |        -41.76557868 | 1.20792e-13 |
|     0.91122641 |          0.91122641 | 7.77156e-16 |
|     0.66967312 |          0.66967312 | 2.22045e-16 |
|    -0.11296643 |         -0.11296643 | 8.32667e-17 |
+----------------+---------------------+-------------+
|  R residuals   |  Python residuals   |    Diff     |
+----------------+---------------------+-------------+
|     2.84030349 |          2.84030349 | 3.64153e-14 |
|    -2.27266294 |         -2.27266294 | 3.95239e-14 |
|     3.84874822 |          3.84874822 | 3.86358e-14 |
|     7.02546544 |          7.02546544 | 1.86517e-14 |
|    -1.63518832 |         -1.63518832 | 1.28786e-14 |
|    -2.30486144 |         -2.30486144 | 1.64313e-14 |
|    -1.29673599 |         -1.29673599 | 1.64313e-14 |
|    -0.29673599 |         -0.29673599 | 1.74305e-14 |
|    -1.65995579 |         -1.65995579 | 6.88338e-15 |
|    -0.10235519 |         -0.10235519 | 5.25968e-15 |
|     0.91434267 |          0.91434267 | 6.88338e-15 |
|     0.47104936 |          0.47104936 | 8.77076e-15 |
|    -2.87642233 |         -2.87642233 | 9.32587e-15 |
|    -1.30346474 |         -1.30346474 | 1.15463e-14 |
|     2.20415398 |          2.20415398 | 8.88178e-16 |
|     0.86525469 |          0.86525469 | 2.10942e-15 |
|    -0.38594843 |         -0.38594843 | 3.66374e-15 |
|     0.40481657 |          0.40481657 | 1.99840e-15 |
|     0.84810988 |          0.84810988 | 3.33067e-16 |
```

(continues on next page)

(continued from previous page)

```
|     1.60668426 |          1.60668426 | 1.04361e-14 |
|   -10.13378768 |        -10.13378768 | 1.77636e-15 |
+----------------+---------------------+-------------+

METHOD: SMDM
+----------------+---------------------+-------------+
| R Coefficients | Python Coefficients |    Diff     |
+----------------+---------------------+-------------+
|   -41.68557185 |        -41.68557188 | 3.71171e-08 |
|     0.83626010 |          0.83626010 | 3.70974e-09 |
|     0.93445082 |          0.93445081 | 1.16940e-08 |
|    -0.12585903 |         -0.12585903 | 7.18257e-10 |
+----------------+---------------------+-------------+
|  R residuals   |  Python residuals   |    Diff     |
+----------------+---------------------+-------------+
|     2.75604537 |          2.75604536 | 7.84833e-09 |
|    -2.36981366 |         -2.36981367 | 7.13008e-09 |
|     3.93210653 |          3.93210652 | 1.34059e-08 |
|     6.36036156 |          6.36036158 | 2.52814e-08 |
|    -1.77073681 |         -1.77073681 | 1.89339e-09 |
|    -2.70518763 |         -2.70518761 | 1.35874e-08 |
|    -1.88448427 |         -1.88448425 | 2.09719e-08 |
|    -0.88448427 |         -0.88448425 | 2.09719e-08 |
|    -2.36014723 |         -2.36014720 | 2.84264e-08 |
|     0.43109366 |          0.43109363 | 2.50160e-08 |
|     1.56382492 |          1.56382489 | 3.14803e-08 |
|     1.37241671 |          1.37241667 | 4.24561e-08 |
|    -2.31718828 |         -2.31718831 | 2.64525e-08 |
|    -0.86718978 |         -0.86718981 | 2.26593e-08 |
|     2.25390572 |          2.25390571 | 1.80236e-09 |
|     0.87632863 |          0.87632863 | 3.52415e-10 |
|    -0.82014859 |         -0.82014857 | 2.21021e-08 |
|     0.06086461 |          0.06086463 | 1.70742e-08 |
|     0.25227282 |          0.25227285 | 2.80500e-08 |
|     1.48643028 |          1.48643029 | 4.35506e-09 |
|    -9.08847985 |         -9.08847991 | 5.40456e-08 |
+----------------+---------------------+-------------+
```

3.3 nlrob Unitary Test

1. **Move to tests directory**

```
(env) $ cd tests/unittest
```

2. **Try all the tests**

```
(env) $ python -m unittest test_nlrob -v
```

and you should get a the following

```
test_nlrob_cm (test_nlrob.NlrobTestCase) ... ok
test_nlrob_mm (test_nlrob.NlrobTestCase) ... ok
test_nlrob_mtl (test_nlrob.NlrobTestCase) ... ok
```

(continues on next page)

(continued from previous page)

```
test_nlrob_tau (test_nlrob.NlrobTestCase) ... ok
test_nls (test_nlrob.NlrobTestCase) ... ok
```

CHAPTER

FOUR

COMPONET DESCRIPTION

4.1 lmrob

4.1.1 lmrob

lmrob.**lmrob**(*formula=''*, *data={}*, *subset=None*, *weights=array([]*, *dtype=float64)*, *na_action='drop'*, *method=''*, *model=True*, *return_x=True*, *return_y=True*, *singular_ok=True*, *contrasts={}*, *offset=array([]*, *dtype=float64)*, *control=None*, *init={}*, ***kwargs*)

Computes fast MM-type estimators for linear (regression) models and returns a dictionary object.

Parameters

formula [str] A symbolic description of the model to be fit.

data [dict] An optional data frame, dict containing the variables in the model.

subset: array_like An optional vector specifying a subset of observations to be used in the fitting process.

weights: array_like An optional vector of weights to be used in the fitting process (in addition to the robustness weights computed in the fitting process).

na_action: str A function which indicates what should happen when the data contain nan. The default is set by the na_action setting of options, and is na_fail if that is unset. The "factory-fresh" default is na_omit. Another possible value is None, no action. Value na_exclude can be useful.

method: str string specifying the estimator-chain. "MM" is interpreted as SM. See Notes, notably the currently recommended setting = "KS2014".

model, return_x, return_y: bool. If True the corresponding components of the fit (the model frame, the model matrix, the response) are returned.

singular_ok: bool If False (the default in S but not in R) a singular fit is an error.

contrasts: dict An optional list. See the contrasts.arg of model.matrix.default.

offset: array_like This can be used to specify an a priori known component to be included in the linear predictor during fitting. An offset term can be included in the formula instead or as well, and if both are specified their sum is used.

control: object Instances of class LmrobControl.

init: dict An optional argument to specify or supply the initial estimate. *See Notes*.

kwargs: additional arguments can be used to specify control parameters directly instead of (but not in addition to!) via control.

Returns

coefficients: array_like The estimate of the coefficient vector

scale: The scale as used in the M estimator.

residuals: array_like Residuals associated with the estimator.

converged: bool True if the IRWLS iterations have converged.

iter: int Number of IRWLS iterations

rweights: array_like The "robustness weights" ψ(r i /S)/(r i /S).

fitted_values: array_like Fitted values associated with the estimator.

init: dict A similar dict that contains the results of intermediate estimates.

rank: int the numeric rank of the fitted linear model.

See also:

`LmrobControl`, *`lmrob_S`*, *`lmrob_M_S`*, *`lmrob_fit`*

Notes

Overview: This function computes an MM-type regression estimator as described in Yohai (1987) and Koller and Stahel (2011). By default it uses a bi-square redescending score function, and it returns a highly robust and highly efficient estimator (with 50% breakdown point and 95% asymptotic efficiency for normal errors). The computation is carried out by a call to lmrob.fit(). The argument setting of lmrob.control is provided to set alternative defaults as suggested in Koller and Stahel (2011) (setting="KS2011"; now do use its extension setting="KS2014"). For further details, see lmrob_control.

Initial Estimator `init`: The initial estimator may be specified using the argument init. This can either be a string, a function or a list. A string can be used to specify built in internal estimators (currently S and M-S, see See also below). A function taking arguments x, y, control, mf (where mf stands for model.frame) and returning a list containing at least the initial coefficients as coefficients and the initial scale estimate scale. Or a list giving the initial coefficients and scale as coefficients and scale. See also Examples. Note that if the init argument is a function or list, the method argument must not contain the initial estimator, e.g., use MDM instead of SMDM. The default, equivalent to init = "S", uses as initial estimator an S-estimator (Rousseeuw and Yohai, 1984) which is computed using the Fast-S algorithm of Salibian-Barrera and Yohai (2006), calling lmrob.S(). That function, since March 2012, by default uses nonsingular subsampling which makes the Fast-S algorithm feasible for categorical data as well, see Koller (2012). Note that convergence problems may still show up as warnings, e.g.,S refinements did not converge (to refine.tol=1e-07) in 200 (= k.max) steps and often can simply be remedied by increasing (i.e. weakening) refine.tol or increasing the allowed number of iterations k.max, see lmrob.control.

Examples

```
>>> from lmrob import *
>>> from rpy2.robjects.packages import importr
>>> from rpy2.robjects import pandas2ri, r
>>> import rpy2.robjects.numpy2ri as rpyn
>>>
>>> stackloss = r("stackloss")
>>> data = {"AirFlow" : rpyn.ri2py(stackloss.rx2("Air.Flow")),
...         "WaterTemp" : rpyn.ri2py(stackloss.rx2("Water.Temp")),
...         "AcidConc" : rpyn.ri2py(stackloss.rx2("Acid.Conc.")),
```

(continues on next page)

(continued from previous page)

```
...          "stack_loss" : rpyn.ri2py(stackloss.rx2("stack.loss"))
...      }
>>> formula = 'stack_loss ~ AirFlow + WaterTemp + AcidConc'
>>> # S Method
>>> m0 = lmrob(formula, data=data, method="S")
```

lmrob.**lmrob_fit**(*x*, *y*, *control*, *init*)

Compute MM-type estimators of regression: An S-estimator is used as starting value, and an M- estimator with fixed scale and redescending psi-function is used from there. Optionally a D-step (Design Adaptive Scale estimate) as well as a second M-step is calculated. Returns a dictionary object.

Parameters

x: array_like design matrix (n × p) typically including a column of 1s for the intercept.

y: array_like numeric response vector (of length n).

control: object An instance of the LmrobControl Class

init: dict Optional list of initial estimates. *See Notes.*

Returns

fitted_values: array_like $X\beta$, i.e., `X.dot(coefficients)`

residuals: array_like the raw residuals, `y - fitted_values`

rweights: array_like robustness weights derived from the final M-estimator residuals (even when not converged).

rank: int degree_freedom n - rank

coefficients: array_like estimated regression coefficient vector.

scale: float the robustly estimated error standard deviation

cov: array_like variance-covariance matrix of coefficients, if the RWLS iterations have converged (and control.cov is not "none").

control: object

iter: int

converged: bool boolean indicating if the RWLS iterations have converged.

init: dict A similar dict that contains the results of intermediate estimates (not for MM- estimates).

See also:

lmrob, *lmrob__M__fit*, *lmrob__D__fit*, *lmrob_S*

Notes

This function is the basic fitting function for MM-type estimation, called by lmrob and typically not to be used on its own. If given, init must be a list of initial estimates containing at least the initial coefficients and scale as coefficients and scale. Otherwise it calls `lmrob_S(..)` and uses it as initial estimator.

Examples

```
>>> from lmrob import *
>>> from rpy2.robjects.packages import importr
>>> from rpy2.robjects import pandas2ri, r
>>> import rpy2.robjects.numpy2ri as rpyn
```

```
>>> stackloss = r("stackloss")
>>> data = {"AirFlow": rpyn.ri2py(stackloss.rx2("Air.Flow")),
...         "WaterTemp": rpyn.ri2py(stackloss.rx2("Water.Temp")),
...         "AcidConc": rpyn.ri2py(stackloss.rx2("Acid.Conc.")),
...         "stack_loss": rpyn.ri2py(stackloss.rx2("stack.loss"))
...         }
>>> formula = 'stack_loss ~ AirFlow + WaterTemp + AcidConc'
>>> Y, X = patsy.dmatrices(formula, data, NA_action="drop",
...                        return_type='matrix')
>>> control = LmrobControl()
>>> m0 = lmrob_fit(X, Y, control=control, init=None)
>>> coefficients = m0.get("coefficients")
>>> residuals = m0.get("residuals")
>>> scale = m0.get("scale")
>>> converged = m0.get("converged")
```

lmrob.**lmrob_S**(*x_*, *y_*, *control*, *only_scale=False*)

Computes an S-estimator for linear regression, using the "fast S" algorithm and returns a dictionary object.

Parameters

x_: array_like

Design matrix (n × p)

y_: array_like Numeric vector of responses (or residuals for only_scale=True).

control: object Instances of class LmrobControl.

only_scale: bool Boolean indicating if only the scale of `y` should be computed. In this case, `y` will typically contain residuals.

Returns

coefficients: array_like

numeric vector (length p) of S-regression coefficient estimates.

scale: float The S-scale residual estimate

fitted_values: array_like numeric vector (length n) of the fitted values.

residuals: array_like numeric vector (length n) of the residuals.

rweights: array_like numeric vector (length n) of the robustness weights.

k_iter: int (maximal) number of refinement iterations used.

converged: bool boolean indicating if all refinement iterations had converged.

control: object the same object as the control argument.

Notes

This function is used by `lmrob_fit` and typically not to be used on its own (because an S-estimator has too low efficiency 'on its own'). By default, the subsampling algorithm uses a customized LU decomposition which ensures a non singular subsample (if this is at all possible). This makes the Fast-S algorithm also feasible for categorical and mixed continuous-categorical data. One can revert to the old subsampling scheme by setting the parameter subsampling in control to "simple".

Examples

```
>>> from lmrob import *
>>> from rpy2.robjects.packages import importr
>>> from rpy2.robjects import pandas2ri, r
>>> import rpy2.robjects.numpy2ri as rpyn
>>>
>>> stackloss = r("stackloss")
>>> data = {"AirFlow" : rpyn.ri2py(stackloss.rx2("Air.Flow")),
...         "WaterTemp" : rpyn.ri2py(stackloss.rx2("Water.Temp")),
...         "AcidConc" : rpyn.ri2py(stackloss.rx2("Acid.Conc.")),
...         "stack_loss" : rpyn.ri2py(stackloss.rx2("stack.loss"))
...       }
>>> formula = 'stack_loss ~ AirFlow + WaterTemp + AcidConc'
>>> Y, X = patsy.dmatrices(formula, data, NA_action="drop", return_type='matrix')
>>> control = LmrobControl()
>>> m0 = lmrob_S(X, Y, control)
>>> coefficients = m0.get("coefficients")
>>> residuals = m0.get("residuals")
>>> scale = m0.get("scale")
>>> converged = m0.get("converged")
>>> k_iter = m0.get("k_iter")
>>> fitted_values = m0.get("fitted_values")
>>> rweights = m0.get("rweights")
```

lmrob.**lmrob_M_S**(*x*, *y*, *control*, *mf*)

Computes an M-S-estimator for linear regression using the "M-S" algorithm.

Parameters

x: array_like Design matrix (n × p)

y: array_like Numeric vector of responses (or residuals for only_scale=True).

control: object Instances of class LmrobControl.

mf: array_like a model matrix as returned by model_frame.

Returns

coefficients: array_like Numeric vector (length p) of M-S-regression coefficient estimates.

scale: float The M-S-scale residual estimate

residuals: array_like numeric vector (legnth n) of the residuals.

rweights: array_like numeric vector (length n) of the robustness weights.

control: object the same as the control argument.

converged: bool Convergence status (always True), needed for lmrob.fit.

See also:

lmrob

Notes

This function is used by lmrob and not intended to be used on its own (because an M-S-estimator has too low efficiency 'on its own'). An M-S estimator is a combination of an S-estimator for the continuous variables and an L1- estimator (i.e. an M-estimator with ψ(t) = sign(t)) for the categorical variables. The S-estimator is estimated using a subsampling algorithm. If the model includes interactions between categorical (factor) and continuous variables, the subsampling algorithm might fail. In this case, one can choose to assign the interaction to the categorical side of variables rather than to the continuous side.

Note that the return status converged does not refer to the actual convergence status. The algorithm used does not guarantee convergence and thus true convergence is almost never reached. This is, however, not a problem if the estimate is only used as initial estimate part of an MM or SMDM estimate.

The algorithm sometimes produces the warning message "Skipping design matrix equilibration (dgeequ): row ?? is exactly zero.". This is just an artifact of the algorithm and can be ignored safely.

Examples

```
>>> from lmrob import *
>>> from rpy2.robjects.packages import importr
>>> from rpy2.robjects import pandas2ri, r
>>> import rpy2.robjects.numpy2ri as rpyn
>>>
>>> stackloss = r("stackloss")
>>> data = {"AirFlow" : rpyn.ri2py(stackloss.rx2("Air.Flow")),
...         "WaterTemp" : rpyn.ri2py(stackloss.rx2("Water.Temp")),
...         "AcidConc" : rpyn.ri2py(stackloss.rx2("Acid.Conc.")),
...         "stack_loss" : rpyn.ri2py(stackloss.rx2("stack.loss"))
...       }
>>> formula = 'stack_loss ~ AirFlow + WaterTemp + AcidConc'
>>> mf_dict = model_frame(formula, data, None, [], 'drop', [])
>>> Y, X = patsy.dmatrices(formula, data, NA_action="drop", return_type='matrix')
>>> control = LmrobControl()
>>> m0 = lmrob_M_S(X, Y, control, mf_dict)
>>> coefficients = m0.get("coefficients")
>>> residuals = m0.get("residuals")
>>> scale = m0.get("scale")
```

lmrob.**lmrob__D__fit**(*obj*, *x=array([], dtype=float64)*, *control=None*, *method=None*)

This function calculates a Design Adaptive Scale estimate for a given MM-estimate. This is sup- posed to be a part of a chain of estimates like SMD or SMDM. Returns a dictionart object

Parameters

obj: dict based on which the estimate is to be calculated.

x: array_like The design matrix; if missing, the method tries to get it from `obj.get("x")`.

control: object Instances of class LmrobControl.

method: str (optional) the method used for obj computation.

Returns

scale: float The Design Adaptive Scale estimate

converged: bool True if the scale calculation converged, False other.

See also:

lmrob_fit, *lmrob*

Notes

This function is used by `lmrob_fit` and typically not to be used on its own. Note that `lmrob_fit()` specifies control potentially differently than the default, but does use the default for method.

Examples

```
>>> from lmrob import *
>>> from rpy2.robjects.packages import importr
>>> from rpy2.robjects import pandas2ri, r
>>> import rpy2.robjects.numpy2ri as rpyn
>>>
>>> stackloss = r("stackloss")
>>> data = {"AirFlow": rpyn.ri2py(stackloss.rx2("Air.Flow")),
...         "WaterTemp": rpyn.ri2py(stackloss.rx2("Water.Temp")),
...         "AcidConc": rpyn.ri2py(stackloss.rx2("Acid.Conc.")),
...         "stack_loss": rpyn.ri2py(stackloss.rx2("stack.loss"))
...       }
>>> formula = 'stack_loss ~ AirFlow + WaterTemp + AcidConc'
>>> Y, X = patsy.dmatrices(formula, data, NA_action="drop",
...                        return_type='matrix')
>>> control = LmrobControl()
>>> init = lmrob_S(X, Y, control)
>>> m0 = lmrob__D__fit(init, X, control)
>>> # Getting the computed values
>>> converged_py = m0.get("converged")
>>> scale_py = m0.get("scale")
```

lmrob.**lmrob__M__fit**(*x=array([]*, *dtype=float64)*, *y=array([]*, *dtype=float64)*, *beta_initial=array([]*, *dtype=float64)*, *scale=None*, *control=None*, *method=None*, *obj=None*)

This function performs RWLS iterations to find an M-estimator of regression. When started from an S-estimated beta.initial, this results in an MM-estimator. Returns a dictionary

Parameters

x: array_like design matrix (n × p) typically including a column of 1s for the intercept.

y: array_like numeric response vector (of length n).

beta_initial: array_like numeric vector (of length p) of initial estimate. Usually the result of an S- regression estimator.

scale: float robust residual scale estimate. Usually an S-scale estimator.

control: object Instances of class LmrobControl.

obj: dict If specified, this is typically used to set values for the other arguments.

method: str (optional) the method used for obj computation.

Returns

coefficients: array_like the M-estimator (or MM-estim.) of regression

control: the control obj input used

scale: float The residual scale estimate

converged: bool True if the RWLS iterations converged, False otherwise

See also:

lmrob_fit, *lmrob*

Notes

This function is used by lmrob_fit and typically not to be used on its own.

Examples

```
>>> from lmrob import *
>>> from rpy2.robjects.packages import importr
>>> from rpy2.robjects import pandas2ri, r
>>> import rpy2.robjects.numpy2ri as rpyn
>>>
>>> stackloss = r("stackloss")
>>> data = {"AirFlow": rpyn.ri2py(stackloss.rx2("Air.Flow")),
...         "WaterTemp": rpyn.ri2py(stackloss.rx2("Water.Temp")),
...         "AcidConc": rpyn.ri2py(stackloss.rx2("Acid.Conc.")),
...         "stack_loss": rpyn.ri2py(stackloss.rx2("stack.loss"))
...       }
>>> formula = 'stack_loss ~ AirFlow + WaterTemp + AcidConc'
>>> Y, X = patsy.dmatrices(formula, data, NA_action="drop",
...                        return_type='matrix')
>>> control = LmrobControl()
>>> init = lmrob_S(X, Y, control)
>>> beta_inital = init.get("coefficients")
>>> scale = init.get("scale")
>>> m0 = lmrob__M__fit(X, Y, beta_inital, scale,
...                    control, method="MM")
>>> # Getting the computed values
>>> coefficients = m0.get("coefficients")
>>> residuals = m0.get("residuals")
>>> scale = m0.get("scale")
>>> converged = m0.get("converged")
>>> iter = m0.get("iter")
```

lmrob.**lmrob_lar**(*x*, *y*, *control=None*)

Compute MM-type estimators of regression: An S-estimator is used as starting value, and an M- estimator with fixed scale and redescending psi-function is used from there. Optionally a D-step (Design Adaptive Scale estimate) as well as a second M-step is calculated. Usage. Return a dictionary object.

Parameters

x: array_like Design matrix (n × p)

y: array_like Numeric vector of responses (or residuals for only_scale=True).

control: object Instances of class LmrobControl.

Returns

fitted_values: array_like $X\beta$, i.e., `X.dot(coefficients)`.

residuals: array_like the raw residuals, `y - fitted_values`

rweights: array_like robustness weights derived from the final M-estimator residuals (even when not converged).

rank: int `degree_freedom n - rank`

coefficients: array_like estimated regression coefficient vector.

scale: float the robustly estimated error standard deviation.

control: object the same as the control argument.

iter: bool converged boolean indicating if the RWLS iterations have converged.

init: dict A similar dict that contains the results of intermediate estimates (not for MM- estimates).

See also:

lmrob, `lmrob..M..fit`, `lmrob..D..fit`, `lmrob.S`

Notes

This function is the basic fitting function for MM-type estimation, called by lmrob and typically not to be used on its own. If given, init must be a list of initial estimates containing at least the initial coefficients and scale as coefficients and scale. Otherwise it calls lmrob.S(..) and uses it as initial estimator.

Examples

```
>>> from lmrob import *
>>> from rpy2.robjects.packages import importr
>>> from rpy2.robjects import pandas2ri, r
>>> import rpy2.robjects.numpy2ri as rpyn
>>>
>>> stackloss = r("stackloss")
>>> data = {"AirFlow" : rpyn.ri2py(stackloss.rx2("Air.Flow")),
...         "WaterTemp" : rpyn.ri2py(stackloss.rx2("Water.Temp")),
...         "AcidConc" : rpyn.ri2py(stackloss.rx2("Acid.Conc.")),
...         "stack_loss" : rpyn.ri2py(stackloss.rx2("stack.loss"))
...       }
>>> formula = 'stack_loss ~ AirFlow + WaterTemp + AcidConc'
>>> mf_dict = model_frame(formula, data, None, [], 'drop', [])
>>> Y, X = patsy.dmatrices(formula, data, NA_action="drop", return_type='matrix')
>>> control = LmrobControl()
>>> m0 = lmrob_lar(X, Y, control)
>>> coefficients = m0.get("coefficients")
>>> residuals = m0.get("residuals")
>>> scale = m0.get("scale")
```

4.2 utils

`lmrob.utils.`**`huberM`**(*x*, *k=1.5*, *weights=None*, *tol=1e-06*, *mu=None*, *s=None*, *se=False*, *warn0scale=True*)

(Generalized) Huber M-estimator of location with MAD scale.

Example

```
>>> module.huberM([1,2,4,5,6,8,10,12])
```

Parameters

- **x** (*list*) – numeric vector.
- **k** (*float*) – positive factor; the algorithm winsorizes at k standard deviations.
- **weights** (*list*) – numeric vector of non-negative weights of same length as x, or None.
- **tol** (*float*) – convergence tolerance.
- **mu** (*float*) – initial location estimator.
- **s** (*float*) – scale estimator held constant through the iterations.
- **se** (*bool*) – logical indicating if the standard error should be computed and returned (as SE component). Currently only available when weights is None.
- **warn0scale** (*bool*) – logical; if True, and s is 0 and len(x) > 1, this will be warned about.

Returns The tuple (mu, s , it , se) containing the location, scale parameters, number of iterations used and the se component.

Return type tuple

lmrob.utils.**mad**(*x*, *center=None*, *constant=1.4826*, *na_rm=False*, *low=False*, *high=False*)

Compute the median absolute deviation, i.e., the (lo-/hi-) median of the absolute deviations from the median, and (by default) adjust by a factor for asymptotically normal consistency.

Example

```
>>> module.mad([1,2,4,5,6,8,10,12])
```

Parameters

- **x** (*list*) – numeric vector.
- **center** (*float*) – optionally, the centre: defaults to the median.
- **constant** (*float*) – scale factor.
- **na_rm** (*bool*) – if True then NaN values are stripped from x before computation takes place.
- **low** (*bool*) – if True, compute the 'lo-median', i.e., for even sample size, do not average the two middle values, but take the smaller one.
- **high** (*bool*) – if True, compute the 'hi-median', i.e., take the larger of the two middle values for even sample size.

Returns the median absolute deviation

Return type float

Raises ValueError

class lmrob.utils.**psi_func**(*rho*, *psi*, *wgt*, *Dpsi*, *Dwgt*, *tDefs*, *Erho*, *Epsi2*, *EDpsi*, *name*, *xtras*)

The class "psi_func" is used to store ψ (psi) functions for M-estimation. In particular, an object of the class contains ρ(x) (rho), its derivative ψ(x) (psi), the weight function ψ(x)/x, and rst derivative of ψ, Dpsi = ψ0(x)

__init__(*rho*, *psi*, *wgt*, *Dpsi*, *Dwgt*, *tDefs*, *Erho*, *Epsi2*, *EDpsi*, *name*, *xtras*)

Example of docstring on the __init__ method.

The __init__ method may be documented in either the class level docstring, or as a docstring on the __init__ method itself.

Either form is acceptable, but the two should not be mixed. Choose one convention to document the __init__ method and be consistent with it.

Note: Do not include the *self* parameter in the `Args` section.

Args: rho (function): the $\rho()$ function. This is used to formulate the objective function; $\rho()$ can be regarded as generalized negative log-likelihood. psi (function): $\psi()$ is the derivative of ρ. wgt (function): The weight function ψ(x)/x. Dpsi (function): the derivative of ψ, Dpsi(x) = psi0(x). Dwgt (function): the derivative of the weight function. tDefs (list): named numeric vector oftuning parameterDefault values. Epsi2 (function): The function for computing E[ψ2(X)] when X is standard normal. EDpsi (function): The function for computing E[ψ0(X)] when X is standard normal. name (string): Name of ψ-function used for printing. xtras (string): Potentially further information.

lmrob.utils.**sumU**(*x*, *weights*)

Calculate weighted sum

Example

```
>>> module.sumU([1,2,4,5,6,8,10,12], [52,44,82,24,68,42,82,20])
```

Parameters

- **x** (*list*) – numeric vector.
- **weights** (*list*) – numeric vector of weights; of the same length as x.

Returns The weighted sum.

Return type float

Raises ValueError

lmrob.utils.**tauHuber**(*x*, *mu*, *k=1.5*, *s=None*, *resid=None*)

Calculate correction factor Tau for the variance of Huber-M-Estimators.

Example

```
>>> module.tauHuber([1,2,4,5,6,8,10,12])
```

Parameters

- **x** (*list*) – numeric vector.
- **mu** (*float*) – location estimator.
- **k** (*float*) – tuning parameter of Huber Psi-function.
- **s** (*float*) – scale estimator held constant through the iterations.
- **resid** (*float*) – the value of (x - mu)/s .

Returns The correction factor Tau for the variance of Huber-M-Estimators.

Return type float

lmrob.utils.**wgtHighMedian**(*x*, *weights=None*)

Compute the weighted Hi-Median of x.

Example

```
>>> module.wgtHighMedian([1,2,4,5,6,8,10,12])
```

Parameters

- **x** (*list*) – numeric vector
- **weights** (*list*) – numeric vector of weights; of the same length as x.

Returns The weighted Hi-Median of x.

Return type float

Raises ValueError

4.3 nlrob

4.3.1 nlrob

nlrob.**nlrob**(*formula*, *data*, *start=array([0.])*, *lower=array([-inf])*, *upper=array([inf])*, *weights=None*, *method='MM'*, *psi=None*, *scale=None*, *control=None*, *test_vec='resid'*, *maxit=20*, *tol=1e-06*, *algorithm='lm'*, *doCov=False*, *trace=False*)

Fits a nonlinear regression model by robust methods. Per default, by an M-estimator, using iterated reweighted least squares (called "IRLS" or also "IWLS"). This function returns a dictionary with the results

Parameters

formula: str A nonlinear formula including variables and parameters of the model, such as y ~ f(x, theta) (cf. nls). (For some checks: if f(.) is linear, then we need parentheses, e.g., y ~ (a + b * x)

data: pandas.core.frame.DataFrame Data frame containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which nlrob is called.

start: pandas.core.frame.DataFrame A named numeric vector of starting parameters estimates, only for method = "M".

lower: pandas.core.frame.DataFrame numeric vectors of lower and upper bounds; if needed, will be replicated to be as long as the longest of start, lower or upper. For (the default) method = "M", if the bounds are unspecified all parameters are assumed to be unconstrained; also, for method "M", bounds can only be used with the "port" algorithm. They are ignored, with a warning, in cases they have no effect. For methods "CM" and "mtl", the bounds must additionally have an entry named "sigma" as that is determined simultaneously in the same optimization, and hence its lower bound must not be negative.

upper: array_like numeric vectors of lower and upper bounds; if needed, will be replicated to be as long as the longest of start, lower or upper. For (the default) method = "M", if the bounds are unspecified all parameters are assumed to be unconstrained; also, for method "M", bounds can only be used with the "port" algorithm. They are ignored, with a warning, in cases they have no effect.

weights: arrray_like An optional vector of weights to be used in the fitting process (for intrinsic weights, not the weights w used in the iterative (robust) fit). I.e., sum(w * e^2) is

minimized with e = residuals, e[i] = y[i] - f(xreg[i], theta), where f(x, theta) is the nonlinear function, and w are the robust weights from resid * weights.

method: str a character string specifying which method to use. The default is "M", for historical and back-compatibility reasons. For the other methods, primarily see nlrob.algorithms. "M" Computes an M-estimator, using nls(*, weights=*) iteratively (hence, IRLS) with weights equal to ψ(r_i) / r_i, where r_i is the i-the residual from the previous fit. "MM" Computes an MM-estimator, starting from init, either "S" or "lts". "tau" Computes a Tau-estimator. "CM" Computes a "Constrained M" (=: CM) estimator. "mtl" Compute as "Maximum Trimmed Likelihood" (=: MTL) estimator. Note that all methods but "M" are "random", hence typically to be preceded by set.seed() in usage.

psi: func A function of the form g(x, 'tuning constant(s)', deriv) that for deriv=0 returns psi(x)/x and for deriv=1 returns psi'(x). Note that tuning constants can not be passed separately, but directly via the specification of psi, typically via a simple _Mwgt_psi1() call as per default.

scale: float When not None, a positive number specifying a scale kept fixed during the iterations (and returned as Scale component).

test_vec: str Character string specifying the convergence criterion. The relative change is tested for residuals with a value of "resid" (the default), for coefficients with "coef", and for weights with "w".

maxit: int maximum number of iterations in the robust loop.

tol: float non-negative convergence tolerance for the robust fit.

algorithm: str character string specifying the algorithm to use for nls, see there, only when method = "M". The default algorithm is a Gauss-Newton algorithm.

doCov: bool a logical specifying if nlrob() should compute the asymptotic variance-covariance matrix (see vcov) already. This used to be hard-wired to TRUE; however, the default has been set to FALSE, as vcov (obj) and summary(obj) can easily compute it when needed.

control: obj An optional object of control settings.

trace: bool logical value indicating if a "trace" of the nls iteration progress should be printed. Default is False. If True, in each robust iteration, the residual sum-of-squares and the parameter values are printed at the conclusion of each nls iteration.

Returns

coefficients: array_like Coefficients of the regressor

residuals: array_like Difference between the real values and the fitted_values

fitted_values: array_like Estimated values by th regressor

See also:

nlrob_MM, *nlrob_tau*, *nlrob_CM*, *nlrob_mtl*

Examples

```
>>> import pandas
>>> from nlrob import *
>>>
>>> formula = "density ~ Asym/(1 + np.exp(( xmid - np.log(conc) )/scal))"
>>> lower = pandas.DataFrame(data=dict(zip(["Asym", "xmid", "scal"], np.
↪zeros(3))),
```

(continues on next page)

(continued from previous page)

```
...          index=[0])
>>> upper = np.array([1])
>>> data = pandas.read_csv("Function=NLROBInput.csv")
>>> # M Method
>>> method = "M"
>>> Rfit = nlrob(formula, data, lower, lower, upper, method=method)
```

nlrob.**nlrob_MM**(*formula*, *data*, *lower*, *upper*, *tol=1e-06*, *psi='bisquare'*, *init='S'*, *ctrl=<nlrob.nlrob.NlrobControl object>*)

Compute an MM-estimator for nonlinear robust (constrained) regression. Returns a dictionary with all the variables of interest

Parameters

formula: str A nonlinear formula including variables and parameters of the model, such as y ~ f(x, theta)

data: pandas.core.frame.DataFrame Data frame containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which nlrob is called.

lower: pandas.core.frame.DataFrame A vector of starting estimates.

upper: array_like upper bound, the shape could be 1 or the the same of lower.

psi: str A function (possibly by name) of the form g(x, 'tuning constant(s)', deriv) that for deriv=0 returns (x)/x and for deriv=1 returns 0 (x).

init: str

ctrl: object NlrobControl Class

Returns

coefficients: array_like Numeric vector of coefficient estimates.

fitted_values [array_like]

residuals: array_like numeric vector of the residuals.

hessian: array_like hessian matrix

ctrl: object NlrobControl Class

See also:

nlrob, *nlrob_tau*, *nlrob_CM*, *nlrob_mtl*

Examples

```
>>> import pandas
>>> from nlrob import *
>>>
>>> formula = "density ~ Asym/(1 + np.exp(( xmid - np.log(conc) )/scal))"
>>> lower = pandas.DataFrame(data=dict(zip(["Asym", "xmid", "scal"], np.
↪zeros(3))),
...          index=[0])
>>> upper = np.array([1])
>>> data = pandas.read_csv("Submodule=NLROBMM.Data=Input.csv")
>>> Rfit_MM = nlrob_MM(formula, data, lower, lower, upper, method=method)
```

nlrob.**nlrob_tau**(*formula*, *data*, *lower*, *upper*, *tol=1e-06*, *psi='bisquare'*, *ctrl=<nlrob.nlrob.NlrobControl object>*, *tuning_chi_scale=None*, *tuning_chi_tau=None*)

Computes a Tau-estimator for nonlinear robust (constrained) regression. Returns a dictionary with all the variables of interest

Parameters

formula: str A nonlinear formula including variables and parameters of the model, such as y ~ f(x, theta)

data: pandas.core.frame.DataFrame Data frame containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which nlrob is called.

lower: pandas.core.frame.DataFrame Dataframe with the initial guesses

upper: array_like upper bound, the shape could be 1 or the the same of lower.

psi: str A function (possibly by name) of the form g(x, 'tuning constant(s)', deriv) that for deriv=0 returns (x)/x and for deriv=1 returns 0 (x).

init: str

ctrl: object NlrobControl Class

Returns

coefficients: array_like Numeric vector of coefficient estimates.

fitted_values [array_like]

residuals: array_like numeric vector of the residuals.

hessian: array_like hessian matrix

ctrl: object NlrobControl Class

See also:

nlrob, *nlrob_MM*, *nlrob_CM*, *nlrob_mtl*

Examples

```
>>> import pandas
>>> from nlrob import *
>>>
>>> formula = "density ~ Asym/(1 + np.exp(( xmid - np.log(conc) )/scal))"
>>> lower = pandas.DataFrame(data=dict(zip(["Asym", "xmid", "scal"], np.
↪zeros(3))),
...          index=[0])
>>> upper = np.array([1])
>>> data = pandas.read_csv("Submodule=NLROBTAU.Data=Input.csv")
>>> Rfit_tau = nlrob_tau(formula, data, lower, lower, upper, method=method)
```

nlrob.**nlrob_CM**(*formula*, *data*, *lower*, *upper*, *tol=1e-06*, *psi='bisquare'*, *ctrl=<nlrob.nlrob.NlrobControl object>*)

Compute an MM-estimator for nonlinear robust (constrained) regression. Returns a dictionary with all the variables of interest Parameters ———-

formula: str A nonlinear formula including variables and parameters of the model, such as y ~ f(x, theta)

data: pandas.core.frame.DataFrame Data frame containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which nlrob is called.

lower: pandas.core.frame.DataFrame Dataframe with the initial guesses

upper: array_like upper bound, the shape could be 1 or the the same of lower.

psi: str A function (possibly by name) of the form g(x, 'tuning constant(s)', deriv) that for deriv=0 returns (x)/x and for deriv=1 returns 0 (x).

init: str ctrl: object

NlrobControl Class

Returns

coefficients: array_like Numeric vector of coefficient estimates.

fitted_values [array_like]

residuals: array_like numeric vector of the residuals.

hessian: array_like hessian matrix

ctrl: object NlrobControl Class

See also:

nlrob, *nlrob_MM*, *nlrob_tau*, *nlrob_mtl*

Examples

```
>>> import pandas
>>> from nlrob import *
>>>
>>> formula = "density ~ Asym/(1 + np.exp(( xmid - np.log(conc) )/scal))"
>>> lower = pandas.DataFrame(data=dict(zip(["Asym", "xmid", "scal"], np.
↪zeros(3))),
...          index=[0])
>>> upper = np.array([1])
>>> data = pandas.read_csv("Submodule=NLROBCM.Data=Input.csv")
>>> Rfit_CM = nlrob_cm(formula, data, lower, lower, upper, method=method)
```

nlrob.**nlrob_mtl**(*formula*, *data*, *lower*, *upper*, *tol=1e-06*, *psi='bisquare'*, *ctrl=<nlrob.nlrob.NlrobControl object>*)

Compute a mtl-estimator for nonlinear robust (constrained) regression

Parameters

formula: str A nonlinear formula including variables and parameters of the model, such as y ~ f(x, theta)

data: pandas.core.frame.DataFrame Data frame containing the variables in the model. If not found in data, the variables are taken from environment(formula), typically the environment from which nlrob is called.

lower: pandas.core.frame.DataFrame Dataframe with the initial guesses

upper: array_like upper bound, the shape could be 1 or the the same of lower.

psi: str A function (possibly by name) of the form g(x, 'tuning constant(s)', deriv) that for deriv=0 returns (x)/x and for deriv=1 returns 0 (x).

init: str

ctrl: object NlrobControl Class

Returns

coefficients: array_like Numeric vector of coefficient estimates.

fitted_values [array_like]

residuals: array_like numeric vector of the residuals.

hessian: array_like hessian matrix

ctrl: object NlrobControl Class

See also:

nlrob, *nlrob_MM*, *nlrob_tau*, nlrob_cm

Examples

```
>>> import pandas
>>> from nlrob import *
>>>
>>> formula = "density ~ Asym/(1 + np.exp(( xmid - np.log(conc) )/scal))"
>>> lower = pandas.DataFrame(data=dict(zip(["Asym", "xmid", "scal"], np.
↪zeros(3))),
...          index=[0])
>>> upper = np.array([1])
>>> data = pandas.read_csv("Submodule=NLROBMTL.Data=Input.csv")
>>> Rfit_mtl = nlrob_mtl(formula, data, lower, lower, upper, method=method)
```

nlrob.**nls** (*formula*, *data*, *start*, *algorithm='lm'*, *weights=None*, *lower=array([-inf])*, *upper=array([inf])*)

Determine the nonlinear (weighted) least-squares estimates of the parameters of a nonlinear model.

Usage

nls(formula, data, start, control, algorithm, trace, subset, weights, na.action, model, lower, upper, . . .)

Parameters

formula: str a nonlinear model formula including variables and parameters. Will be coerced to a formula if necessary.

data: pandas.core.frame.DataFrame Data frame in which to evaluate the variables in formula and weights. Can also be a list or an environment, but not a matrix.

start: pandas.core.frame.DataFrame A vector of starting estimates.

algorithm: str Character string specifying the algorithm to use. The default algorithm is a "lm" algorithm. Other possible values are 'trf', 'dogbox'

lower: scalar, array_like Lower bounds on Parameters. An array with the length equal to the number of parameters, or a scalar (in which case the bound is taken to be the same for all parameters.)

upper: scalar, array_like Upper bounds on Parameters. An array with the length equal to the number of parameters, or a scalar (in which case the bound is taken to be the same for all parameters.)

Returns

coefficients: array_like Numeric vector of coefficient estimates.

residuals: array_like numeric vector of the residuals.

cov: array_like covariance matrix

See also:

nlrob, *nlrob_MM*, *nlrob_tau*, nlrob_cm

Examples

```
>>> import pandas
>>> from nlrob import *
>>>
>>> formula = "density ~ Asym/(1 + np.exp(( xmid - np.log(conc) )/scal))"
>>> lower = pandas.DataFrame(data=dict(zip(["Asym", "xmid", "scal"], np.
↪zeros(3))),
...          index=[0])
>>> upper = np.array([1])
>>> data = pandas.read_csv("Submodule=NLS.Data=Input.csv")
>>> Rfit_nls = nls(formula, data, lower)
```

4.4 utils

nlrob.utils.**JDEoptim**(*lower*, *upper*, *fn*, *constr=None*, *meq=0*, *eps=1e-05*, *NP=None*, *Fl=0.1*, *Fu=1*, *tau_F=0.1*, *tau_CR=0.1*, *tau_pF=0.1*, *jitter_factor=0.001*, *tol=1e-15*, *maxiter=None*, *fnscale=1*, *compare_to=None*, *add_to_init_pop=None*, *trace=False*, *triter=1*, *details=False*)

An implementation of a bespoke jDE variant of the Differential Evolution stochastic algorithm for global optimization of nonlinear programming problems

Usage

JDEoptim(lower, upper, fn,constr=None, meq=0, eps=1e-5, NP=None, Fl=0.1, Fu=1, tau_F=0.1, tau_CR=0.1, tau_pF=0.1, jitter_factor=0.001, tol=1e-15, maxiter=None, fnscale=1, compare_to=None, add_to_init_pop=None, trace=False, triter=1, details=False)

Parameters

lower, upper: array_like numeric vectors of lower or upper bounds, respectively, for the parameters to be optimized over. Must be finite as they bound the hyper rectangle of the initial random population.

fn: function (nonlinear) objective function to be minimized. It takes as first argument the vector of parameters over which minimization is to take place. It must return the value of the function at that point.

constr: function an optional function for specifying the nonlinear constraints under which we want to minimize fn. They should be given in the form $h_i(x) = 0, g_i(x) le 0$. This function takes the vector of parameters as its first argument and returns a real vector with the length of the total number of constraints. It defaults to None, meaning that bound-constrained minimization is used.

meq: integer an optional positive integer specifying that the first meq constraints are treated as equality constraints, all the remaining as inequality constraints. Defaults to 0 (inequality constraints only).

eps: float maximal admissible constraint violation for equality constraints. An optional real vector of small positive tolerance values with length meq used in the transformation of equalities into inequalities of the form $|h_i(x)| - epsilon le 0$. A scalar value is expanded to apply to all equality constraints. Default is 1e-5.

NP: int an optional positive integer giving the number of candidate solutions in the randomly distributed initial population. Defaults to 10*length(lower).

FL: float an optional scalar which represents the minimum value that the scaling factor F could take. Default is 0.1, which is almost always satisfactory.

Fu: float an optional scalar which represents the maximum value that the scaling factor F could take. Default is 1, which is almost always satisfactory.

tau_F: float an optional scalar which represents the probability that the scaling factor F is updated. Defaults to 0.1, which is almost always satisfactory.

tau_CR: float an optional scalar which represents the probability that the mutation probability pF in the mutation strategy DE/rand/1/either-or is updated. Defaults to 0.1.

jitter_factor: float an optional tuning constant for jitter. If None L only dither is used. Defaults to 0.001.

tol: float an optional positive scalar giving the tolerance for the stopping criterion. Default is 1e-15.

maxiter: int an optional positive integer specifying the maximum number of iterations that may be performed before the algorithm is halted. Defaults to 200*length(lower).

fnscale: float an optional positive scalar specifying the typical magnitude of fn. It is used only in the stopping criterion. Defaults to 1. See 'Details'.

compare_to: str an optional character string controlling which function should be applied to the fn values of the candidate solutions in a generation to be compared with the so-far best one when evaluating the stopping criterion. If "median" the median function is used; else, if "max" the max function is used. It defaults to "median".

add_to_init_pop: array_like an optional real vector of length length(lower) or matrix with length(lower) rows specifying initial values of the parameters to be optimized which are appended to the randomly generated initial population. It defaults to None.

trace: boolean an optional logical value indicating if a trace of the iteration progress should be printed. Default is False.

triter: int an optional positive integer that controls the frequency of tracing when trace=True. Default is triter=1, which means that iteration: <value of stopping test> (value of best solution) best solution {index of violated constraints} is printed at every iteration.

details: boolean an optional logical value. If True the output will contain the parameters in the final population and their respective fn values. Defaults to False.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

l

n

INDEX

Symbols

H

J

L

M

N

P

S

T

W