

**sqrtlasso** — Square-root lasso for prediction and model selection

<a href="#">Description</a>	<a href="#">Quick start</a>	<a href="#">Menu</a>	<a href="#">Syntax</a>
<a href="#">Options</a>	<a href="#">Remarks and examples</a>	<a href="#">Stored results</a>	<a href="#">Methods and formulas</a>
<a href="#">References</a>	<a href="#">Also see</a>		

## Description

`sqrtlasso` selects covariates and fits linear models using square-root lasso. Results from `sqrtlasso` can be used for prediction and model selection. Results from `sqrtlasso` are typically similar to results from `lasso`.

`sqrtlasso` saves but does not display estimated coefficients. The [\[LASSO\]](#) [lasso postestimation](#) commands can be used to generate predictions, report coefficients, and display measures of fit.

For an introduction to lasso, see [\[LASSO\]](#) [Lasso intro](#).

## Quick start

Fit a linear model for `y1`, and select covariates from `x1–x100` using cross-validation (CV)

```
sqrtlasso y x1-x100
```

As above, but force `x1` and `x2` to be in the model while square-root lasso selects from `x3–x100`

```
sqrtlasso y (x1 x2) x3-x100
```

Set a random-number seed for reproducibility

```
sqrtlasso y x1-x100, rseed(1234)
```

Calculate the CV function beyond the CV minimum to get the full coefficient paths, knots, etc.

```
sqrtlasso y x1-x100, selection(cv, alllambdas)
```

## Menu

Statistics > Lasso > Square-root lasso

## Syntax

```
sqrtlasso depvar [(alwaysvars)] othervars [if] [in] [weight] [, options]
```

*alwaysvars* are variables that are always included in the model.

*othervars* are variables that `sqrtlasso` will choose to include in or exclude from the model.

## 2 `sqrlasso` — Square-root lasso for prediction and model selection

---

<i>options</i>	Description
Model	
<code>noconstant</code>	suppress constant term
<code>selection(sel_method)</code>	selection method to select a value of the square-root lasso penalty parameter $\lambda^*$ from the set of possible $\lambda$ 's
<code>offset(varname_o)</code>	include <i>varname_o</i> in model with coefficient constrained to 1
<code>cluster(clustvar)</code>	specify cluster variable <i>clustvar</i>
Optimization	
<code>[no]log</code>	display or suppress an iteration log
<code>rseed(#)</code>	set random-number seed
<code>grid(#_g [ , ratio(#) min(#)])</code>	specify the set of possible $\lambda$ 's using a logarithmic grid with $\#_g$ grid points
<code>stop(#)</code>	tolerance for stopping the iteration over the $\lambda$ grid early
<code>cvtolerance(#)</code>	tolerance for identification of the CV function minimum
<code>bictolerance(#)</code>	tolerance for identification of the BIC function minimum
<code>tolerance(#)</code>	convergence tolerance for coefficients based on their values
<code>dtolerance(#)</code>	convergence tolerance for coefficients based on deviance
<code>penaltywt(matname)</code>	programmer's option for specifying a vector of weights for the coefficients in the penalty term
<hr/>	
<i>sel_method</i>	Description
<code>cv [ , cv_opts ]</code>	select $\lambda^*$ using CV; the default
<code>plugin [ , plugin_opts ]</code>	select $\lambda^*$ using a plugin iterative formula
<code>bic [ , bic_opts ]</code>	select $\lambda^*$ using BIC function
<code>none</code>	do not select $\lambda^*$
<hr/>	
<i>cv_opts</i>	Description
<code>folds(#)</code>	use # folds for CV
<code>alllambdas</code>	fit models for all $\lambda$ 's in the grid or until the <code>stop(#)</code> tolerance is reached; by default, the CV function is calculated sequentially by $\lambda$ , and estimation stops when a minimum is identified
<code>serule</code>	use the one-standard-error rule to select $\lambda^*$
<code>stopok</code>	when the CV function does not have an identified minimum and the <code>stop(#)</code> stopping criterion for $\lambda$ was reached at $\lambda_{stop}$ , set the selected $\lambda^*$ to be $\lambda_{stop}$ ; the default
<code>strict</code>	do not select $\lambda^*$ when the CV function does not have an identified minimum; this is a stricter alternative to the default <code>stopok</code>
<code>gridminok</code>	when the CV function does not have an identified minimum and the <code>stop(#)</code> stopping criterion for $\lambda$ was not reached, set the selected $\lambda^*$ to be the minimum of the $\lambda$ grid, $\lambda_{gmin}$ ; this is a looser alternative to the default <code>stopok</code> and is rarely used

---

<i>plugin_opts</i>	Description
<u>heteroskedastic</u>	assume model errors are heteroskedastic; the default
<u>homoskedastic</u>	assume model errors are homoskedastic

  

<i>bic_opts</i>	Description
<u>alllambdas</u>	fit models for all $\lambda$ 's in the grid or until the <code>stop(#)</code> tolerance is reached; by default, the BIC function is calculated sequentially by $\lambda$ , and estimation stops when a minimum is identified
<u>stopok</u>	when the BIC function does not have an identified minimum and the <code>stop(#)</code> stopping criterion for $\lambda$ was reached at $\lambda_{\text{stop}}$ , set the selected $\lambda^*$ to be $\lambda_{\text{stop}}$ ; the default
<u>strict</u>	do not select $\lambda^*$ when the BIC function does not have an identified minimum; this is a stricter alternative to the default <code>stopok</code>
<u>gridminok</u>	when the BIC function does not have an identified minimum and the <code>stop(#)</code> stopping criterion for $\lambda$ was not reached, set the selected $\lambda^*$ to be the minimum of the $\lambda$ grid, $\lambda_{\text{gmin}}$ ; this is a looser alternative to the default <code>stopok</code> and is rarely used
<u>postselection</u>	use postselection coefficients to compute BIC

*alwaysvars* and *othervars* may contain factor variables; see [U] 11.4.3 Factor variables.

`collect` is allowed; see [U] 11.1.10 Prefix commands.

Default weights are not allowed. `iweights` are allowed with all *sel\_method* options. See [U] 11.1.6 weight.

`penaltywt(matname)` does not appear in the dialog box.

See [U] 20 Estimation and postestimation commands for more capabilities of estimation commands.

## Options

See [LASSO] lasso fitting for an overview of the lasso estimation procedure and a detailed description of how to set options to control it.

### Model

`noconstant` omits the constant term. Note, however, when there are factor variables among the *othervars*, `sqrtlasso` can potentially create the equivalent of the constant term by including all levels of a factor variable. This option is likely best used only when all the *othervars* are continuous variables and there is a conceptual reason why there should be no constant term.

`selection(cv)`, `selection(plugin)`, `selection(bic)`, and `selection(none)` specify the selection method used to select  $\lambda^*$ . These options also allow suboptions for controlling the specified selection method.

`selection(cv [ , cv_opts ])` is the default. It selects  $\lambda^*$  to be the  $\lambda$  that gives the minimum of the CV function. `lasso postestimation` commands can be used after `selection(cv)` to assess alternative  $\lambda^*$  values.

*cv\_opts* are `folds(#)`, `alllambdas`, `serule`, `stopok`, `strict`, and `gridminok`.

`folds(#)` specifies that CV with # folds be done. The default is `folds(10)`.

`alllambdas` specifies that models be fit for all  $\lambda$ 's in the grid or until the `stop(#)` tolerance is reached. By default, models are calculated sequentially from largest to smallest  $\lambda$ , and the CV function is calculated after each model is fit. If a minimum of the CV function is found, the computation ends at that point without evaluating additional smaller  $\lambda$ 's.

`alllambdas` computes models for these additional smaller  $\lambda$ 's. Because computation time is greater for smaller  $\lambda$ , specifying `alllambdas` may increase computation time manyfold. Specifying `alllambdas` is typically done only when a full plot of the CV function is wanted for assurance that a true minimum has been found. Regardless of whether `alllambdas` is specified, the selected  $\lambda^*$  will be the same.

`serule` selects  $\lambda^*$  based on the “one-standard-error rule” recommended by [Hastie, Tibshirani, and Wainwright \(2015, 13–14\)](#) instead of the  $\lambda$  that minimizes the CV function. The one-standard-error rule selects the largest  $\lambda$  for which the CV function is within a standard error of the minimum of the CV function.

`stopok`, `strict`, and `gridminok` specify what to do when the CV function does not have an identified minimum. A minimum is identified at  $\lambda^*$  when the CV function at both larger and smaller adjacent  $\lambda$ 's is greater than it is at  $\lambda^*$ . When the CV function has an identified minimum, these options all do the same thing: the selected  $\lambda^*$  is the  $\lambda$  that gives the minimum. In some cases, however, the CV function declines monotonically as  $\lambda$  gets smaller and never rises to identify a minimum. When the CV function does not have an identified minimum, `stopok` and `gridminok` make alternative selections for  $\lambda^*$ , and `strict` makes no selection. You may specify only one of `stopok`, `strict`, or `gridminok`; `stopok` is the default if you do not specify one. With each of these options, estimation results are always left in place, and alternative  $\lambda^*$  can be selected and evaluated.

`stopok` specifies that when the CV function does not have an identified minimum and the `stop(#)` stopping tolerance for  $\lambda$  was reached, the selected  $\lambda^*$  is  $\lambda_{\text{stop}}$ , the  $\lambda$  that met the stopping criterion.  $\lambda_{\text{stop}}$  is the smallest  $\lambda$  for which coefficients are estimated, and it is assumed that  $\lambda_{\text{stop}}$  has a CV function value close to the true minimum. When no minimum is identified and the `stop(#)` criterion is not met, an error is issued.

`strict` requires the CV function to have an identified minimum, and if not, an error is issued.

`gridminok` is a rarely used option that specifies that when the CV function has no identified minimum and the `stop(#)` stopping criterion was not met,  $\lambda_{\text{gmin}}$ , the minimum of the  $\lambda$  grid, is the selected  $\lambda^*$ .

The `gridminok` selection criterion is looser than the default `stopok`, which is looser than `strict`. With `strict`, only an identified minimum is selected. With `stopok`, either the identified minimum or  $\lambda_{\text{stop}}$  is selected. With `gridminok`, either the identified minimum or  $\lambda_{\text{stop}}$  or  $\lambda_{\text{gmin}}$  is selected, in this order.

`selection(plugin [, plugin_opts])` selects  $\lambda^*$  based on a “plugin” iterative formula dependent on the data. The plugin method was designed for lasso inference methods and is useful when using `sqrtlasso` to manually implement inference methods, such as double-selection lasso. The plugin estimator calculates a value for  $\lambda^*$  that dominates the noise in the estimating equations, which makes it less likely to include variables that are not in the true model. See [Methods and formulas](#).

`selection(plugin)` does not estimate coefficients for any other values of  $\lambda$ , so it does not require a  $\lambda$  grid, and none of the grid options apply. It is much faster than `selection(cv)` because estimation is done only for a single value of  $\lambda$ . It is an iterative procedure, however, and if the plugin is computing estimates for a small  $\lambda$  (which means many nonzero coefficients),

the estimation can still be time consuming. Because estimation is done only for one  $\lambda$ , you cannot assess alternative  $\lambda^*$  as the other selection methods allow.

*plugin\_opts* are `heteroskedastic` and `homoskedastic`.

`heteroskedastic` assumes model errors are heteroskedastic. It is the default. Specifying `selection(plugin)` is equivalent to specifying `selection(plugin, heteroskedastic)`.

`homoskedastic` assumes model errors are homoskedastic. See [Methods and formulas](#).

`selection(bic [ , bic_opts ])` selects  $\lambda^*$  by using the Bayesian information criterion function. It selects the  $\lambda^*$  with the minimum BIC function value.

*bic\_opts* are `alllambdas`, `stopok`, `strict`, `gridminok`, and `postselection`.

`alllambdas` specifies that models be fit for all  $\lambda$ 's in the grid or until the `stop(#)` tolerance is reached. By default, models are calculated sequentially from largest to smallest  $\lambda$ , and the BIC function is calculated after each model is fit. If a minimum of the BIC function is found, the computation ends at that point without evaluating additional smaller  $\lambda$ 's.

`alllambdas` computes models for these additional smaller  $\lambda$ 's. Because computation time is greater for smaller  $\lambda$ , specifying `alllambdas` may increase computation time manyfold. Specifying `alllambdas` is typically done only when a full plot of the BIC function is wanted for assurance that a true minimum has been found. Regardless of whether `alllambdas` is specified, the selected  $\lambda^*$  will be the same.

`stopok`, `strict`, and `gridminok` specify what to do when the BIC function does not have an identified minimum. A minimum is identified at  $\lambda^*$  when the BIC function at both larger and smaller adjacent  $\lambda$ 's is greater than it is at  $\lambda^*$ . When the BIC function has an identified minimum, these options all do the same thing: the selected  $\lambda^*$  is the  $\lambda$  that gives the minimum. In some cases, however, the BIC function declines monotonically as  $\lambda$  gets smaller and never rises to identify a minimum. When the BIC function does not have an identified minimum, `stopok` and `gridminok` make alternative selections for  $\lambda^*$ , and `strict` makes no selection. You may specify only one of `stopok`, `strict`, or `gridminok`; `stopok` is the default if you do not specify one. With each of these options, estimation results are always left in place, and alternative  $\lambda^*$  can be selected and evaluated.

`stopok` specifies that when the BIC function does not have an identified minimum and the `stop(#)` stopping tolerance for  $\lambda$  was reached, the selected  $\lambda^*$  is  $\lambda_{\text{stop}}$ , the  $\lambda$  that met the stopping criterion.  $\lambda_{\text{stop}}$  is the smallest  $\lambda$  for which coefficients are estimated, and it is assumed that  $\lambda_{\text{stop}}$  has a BIC function value close to the true minimum. When no minimum is identified and the `stop(#)` criterion is not met, an error is issued.

`strict` requires the BIC function to have an identified minimum, and if not, an error is issued.

`gridminok` is a rarely used option that specifies that when the BIC function has no identified minimum and the `stop(#)` stopping criterion was not met, then  $\lambda_{\text{gmin}}$ , the minimum of the  $\lambda$  grid, is the selected  $\lambda^*$ .

The `gridminok` selection criterion is looser than the default `stopok`, which is looser than `strict`. With `strict`, only an identified minimum is selected. With `stopok`, either the identified minimum or  $\lambda_{\text{stop}}$  is selected. With `gridminok`, either the identified minimum or  $\lambda_{\text{stop}}$  or  $\lambda_{\text{gmin}}$  is selected, in this order.

`postselection` specifies to use the `postselection` coefficients to compute the BIC function. By default, the penalized coefficients are used.

`selection(none)` does not select a  $\lambda^*$ . Square-root lasso is estimated for the grid of values for  $\lambda$ , but no attempt is made to determine which  $\lambda$  should be selected. The postestimation command `lassoknots` can be run to view a table of  $\lambda$ 's that define the knots (the sets of nonzero coefficients) for the estimation. The `lassoselect` command can be used to select a value for  $\lambda^*$ , and `lassogof` can be run to evaluate the prediction performance of  $\lambda^*$ .

When `selection(none)` is specified, the CV function is not computed. If you want to view the knot table with values of the CV function shown and then select  $\lambda^*$ , you must specify `selection(cv)`. There are no suboptions for `selection(none)`.

`offset(varnameo)` specifies that `varnameo` be included in the model with its coefficient constrained to be 1.

`cluster(clustvar)` specifies the cluster variable `clustvar`. Specifying a cluster variable will affect how the log-likelihood function is computed and the sample split in cross-validation. The log-likelihood function is computed as the sum of the log likelihood at the cluster levels. If option `selection(cv)` is specified, the cross-validation sample is split by the clusters defined by `clustvar`. That is, the subsample in each fold is drawn on the cluster level. Therefore, all observations in a cluster are kept together in the same subsample.

#### Optimization

`[no]` `log` displays or suppresses a log showing the progress of the estimation.

`rseed(#)` sets the random-number seed. This option can be used to reproduce results for `selection(cv)`. The other selection methods, `selection(plugin)` and `selection(none)`, do not use random numbers. `rseed(#)` is equivalent to typing `set seed #` prior to running `sqrlasso`. See [R] [set seed](#).

`grid(#g [, ratio(#) min(#)])` specifies the set of possible  $\lambda$ 's using a logarithmic grid with  $\#_g$  grid points.

$\#_g$  is the number of grid points for  $\lambda$ . The default is  $\#_g = 100$ . The grid is logarithmic with the  $i$ th grid point ( $i = 1, \dots, n = \#_g$ ) given by  $\ln \lambda_i = [(i - 1)/(n - 1)] \ln r + \ln \lambda_{\text{gmax}}$ , where  $\lambda_{\text{gmax}} = \lambda_1$  is the maximum,  $\lambda_{\text{gmin}} = \lambda_n = \text{min}(\#)$  is the minimum, and  $r = \lambda_{\text{gmin}}/\lambda_{\text{gmax}} = \text{ratio}(\#)$  is the ratio of the minimum to the maximum.

`ratio(#)` specifies  $\lambda_{\text{gmin}}/\lambda_{\text{gmax}}$ . The maximum of the grid,  $\lambda_{\text{gmax}}$ , is set to the smallest  $\lambda$  for which all the coefficients in the lasso are estimated to be zero (except the coefficients of the *alwaysvars*).  $\lambda_{\text{gmin}}$  is then set based on `ratio(#)`. When  $p < N$ , where  $p$  is the total number of *othervars* and *alwaysvars* (not including the constant term) and  $N$  is the number of observations, the default value of `ratio(#)` is  $1e-4$ . When  $p \geq N$ , the default is  $1e-2$ .

`min(#)` sets  $\lambda_{\text{gmin}}$ . By default,  $\lambda_{\text{gmin}}$  is based on `ratio(#)` and  $\lambda_{\text{gmax}}$ , which is computed from the data.

`stop(#)` specifies a tolerance that is the stopping criterion for the  $\lambda$  iterations. The default is  $1e-5$ . This option does not apply when the selection method is `selection(plugin)`. Estimation starts with the maximum grid value,  $\lambda_{\text{gmax}}$ , and iterates toward the minimum grid value,  $\lambda_{\text{gmin}}$ . When the relative difference in the deviance produced by two adjacent  $\lambda$  grid values is less than `stop(#)`, the iteration stops and no smaller  $\lambda$ 's are evaluated. The value of  $\lambda$  that meets this tolerance is denoted by  $\lambda_{\text{stop}}$ . Typically, this stopping criterion is met before the iteration reaches  $\lambda_{\text{gmin}}$ .

Setting `stop(#)` to a larger value means that iterations are stopped earlier at a larger  $\lambda_{\text{stop}}$ . To produce coefficient estimates for all values of the  $\lambda$  grid, you can specify `stop(0)`. Note, however, that computations for small  $\lambda$ 's can be extremely time consuming. In terms of time, when you use `selection(cv)`, the optimal value of `stop(#)` is the largest value that allows estimates for just enough  $\lambda$ 's to be computed to identify the minimum of the CV function. When setting `stop(#)`

to larger values, be aware of the consequences of the default  $\lambda^*$  selection procedure given by the default `stopok`. You may want to override the `stopok` behavior by using `strict`.

`cvtolerance(#)` is a rarely used option that changes the tolerance for identifying the minimum CV function. For linear models, a minimum is identified when the CV function rises above a nominal minimum for at least three smaller  $\lambda$ 's with a relative difference in the CV function greater than  $\#$ . For nonlinear models, at least five smaller  $\lambda$ 's are required. The default is  $1e-3$ . Setting  $\#$  to a bigger value makes a stricter criterion for identifying a minimum and brings more assurance that a declared minimum is a true minimum, but it also means that models may need to be fit for additional smaller  $\lambda$ , which can be time consuming. See *Methods and formulas* for [LASSO] `lasso` for more information about this tolerance and the other tolerances.

`bictolerance(#)` is a rarely used option that changes the tolerance for identifying the minimum BIC function. A minimum is identified when the BIC function rises above a nominal minimum for at least two smaller  $\lambda$ 's with a relative difference in the BIC function greater than  $\#$ . The default is  $1e-2$ . Setting  $\#$  to a bigger value makes a stricter criterion for identifying a minimum and brings more assurance that a declared minimum is a true minimum, but it also means that models may need to be fit for additional smaller  $\lambda$ , which can be time consuming. See *Methods and formulas* in [LASSO] `lasso` for more information about this tolerance and the other tolerances.

`tolerance(#)` is a rarely used option that specifies the convergence tolerance for the coefficients. Convergence is achieved when the relative change in each coefficient is less than this tolerance. The default is `tolerance(1e-7)`.

`dtolerance(#)` is a rarely used option that changes the convergence criterion for the coefficients. When `dtolerance(#)` is specified, the convergence criterion is based on the change in deviance instead of the change in the values of coefficient estimates. Convergence is declared when the relative change in the deviance is less than  $\#$ . More-accurate coefficient estimates are typically achieved by not specifying this option and instead using the default `tolerance(1e-7)` criterion or specifying a smaller value for `tolerance(#)`.

The following option is available with `sqrtlasso` but is not shown in the dialog box:

`penaltywt(matname)` is a programmer's option for specifying a vector of weights for the coefficients in the penalty term. The contribution of each coefficient to the square-root lasso penalty term is multiplied by its corresponding weight. Weights must be nonnegative. By default, each coefficient's penalty weight is 1.

## Remarks and examples

[stata.com](http://www.stata.com)

We assume you have read the lasso introduction [LASSO] `Lasso intro`.

The square-root lasso is an alternative version of lasso. Lasso minimizes

$$\frac{1}{2N}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}')'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}') + \lambda \sum_{j=1}^p |\beta_j|$$

whereas square-root lasso minimizes

$$\sqrt{\frac{1}{N}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}')'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}')} + \frac{\lambda}{N} \sum_{j=1}^p |\beta_j|$$

In the square-root formulation, the standard deviation of the error term becomes a multiplicative constant that drops out of the minimization. This lack of dependence facilitates the derivation of plugin estimators for the lasso penalty parameter  $\lambda^*$  because there is no need to estimate the standard deviation of the error term as part of the plugin formula.

Square-root lasso is primarily used in combination with a plugin estimator for  $\lambda^*$ . The resulting square-root lasso estimation can be used with the double-selection or partialing-out methods described in [\[LASSO\] Lasso inference intro](#).

Square-root lasso can also be used on its own for prediction or model selection. To be consistent with **lasso**, the default selection method for  $\lambda^*$  is CV. To use the plugin estimator, specify the option `selection(plugin)`.

Square-root lasso was formulated by [Belloni, Chernozhukov, and Wang \(2011\)](#), who also derived the square-root lasso plugin estimator for  $\lambda$ , which is implemented here.

### ► Example 1: Square-root lasso and lasso

Let's compare square-root lasso with an ordinary lasso to illustrate that their results are numerically similar when used with CV.

We load the example dataset we used in [\[LASSO\] lasso examples](#). It has stored variable lists created by `v1`. See [\[D\] v1](#) for a complete description of the `v1` system and how to use it to manage large variable lists.

After we load the dataset, we type `v1 rebuild` to make the saved variable lists active again.

```
. use https://www.stata-press.com/data/r17/fakesurvey_v1
(Fictitious survey data with v1)
. v1 rebuild
Rebuilding v1 macros ...
```

Macro	Macro's contents	
	# Vars	Description
System		
\$vldummy	98	0/1 variables
\$vlcategorical	16	categorical variables
\$vlcontinuous	29	continuous variables
\$vluncertain	16	perhaps continuous, perhaps categorical variables
\$vlother	12	all missing or constant variables
User		
\$demographics	4	variables
\$factors	110	variables
\$idemographics		factor-variable list
\$ifactors		factor-variable list

We randomly split our data into two samples of equal sizes. One we will fit lassos on, and the other we will use to test their predictions. We use `splitsample` to generate a variable indicating the samples.

```
. set seed 1234
. splitsample, generate(sample) nsplit(2)
. label define svalues 1 "Training" 2 "Testing"
. label values sample svalues
```



We have four user-defined variable lists, `demographics`, `factors`, `idemographics`, and `ifactors`. The variable lists `idemographics` and `ifactors` contain factor-variable versions of the categorical variables in `demographics` and `factors`. That is, a variable `q3` in `demographics` is `i.q3` in `idemographics`. See the examples in [LASSO] [lasso examples](#) to see how we created these variable lists.

We are going to use `idemographics` and `ifactors` along with the system-defined variable list `vlcontinuous` as arguments to `sqrtlasso`. Together they contain the potential variables we want to specify. Variable lists are actually global macros, and when we use them as arguments in commands, we put a `$` in front of them.

We also set the [random-number seed](#) using the `rseed()` option so we can reproduce our results.

```
. sqrtlasso q104 $idemographics $factors $vlcontinuous if sample == 1,
> rseed(1234)

10-fold cross-validation with 100 lambdas ...
Grid value 1:      lambda = 104.6235  no. of nonzero coef. =      0
Folds: 1...5....10  CVF = 17.2848
  (output omitted)
Grid value 23:      lambda = 13.51264  no. of nonzero coef. =      87
Folds: 1...5....10  CVF = 12.35321
... cross-validation complete ... minimum found
Square-root lasso linear model          No. of obs          =          458
                                         No. of covariates =          277
Selection: Cross-validation             No. of CV folds    =           10
```

ID	Description	lambda	No. of nonzero coef.	Out-of-sample R-squared	CV mean prediction error
1	first lambda	104.6235	0	-0.0058	17.2848
17	lambda before	23.61373	53	0.2890	12.21892
* 18	selected lambda	21.51595	61	0.2901	12.19933
19	lambda after	19.60453	67	0.2899	12.20295
23	last lambda	13.51264	87	0.2812	12.35321

```
* lambda selected by cross-validation.
. estimates store sqrtcv
```

The square-root lasso with the default CV selection method selected a model with 61 variables in it.

Let's run lasso with the same potential variables.

```
. lasso linear q104 $idemographics $factors $vlcontinuous if sample == 1,
> rseed(1234)

10-fold cross-validation with 100 lambdas ...
Grid value 1:   lambda = .9469819   no. of nonzero coef. =      0
  (output omitted)
Grid value 25:   lambda = .1015418   no. of nonzero coef. =     78
Folds: 1...5....10   CVF = 12.26768
... cross-validation complete ... minimum found

Lasso linear model                               No. of obs      =      458
                                                No. of covariates =      277
Selection: Cross-validation                       No. of CV folds =      10
```

ID	Description	lambda	No. of nonzero coef.	Out-of- sample R-squared	CV mean prediction error
1	first lambda	.9469819	0	-0.0046	17.26383
19	lambda before	.1774471	47	0.2899	12.20399
* 20	selected lambda	.1616832	51	0.2912	12.18122
21	lambda after	.1473197	60	0.2908	12.18739
25	last lambda	.1015418	78	0.2862	12.26768

```
* lambda selected by cross-validation.
. estimates store lassocv
```

Lasso selected a model with 51 variables in it.

After we ran `sqrtlasso` and `lasso`, we used `estimates store` to keep the results in memory. This lets us compare the models. We can use `lassocoeff` to view the coefficient estimates. We display the standardized coefficients and sort them so that the biggest in absolute values are shown first.

```
. lassocoeff sqrtcv lassocv, display(coef, standardized) sort(coef, standardized)
```

	sqrtcv	lassocv
q19		
No	-.8446332	-.8119414
q85		
No	-.7089993	-.6940387
3.q156	-.6843823	-.6727969
q101		
No	.5981556	.5785246
q48		
No	-.5867942	-.5502145
q88		
No	.5793049	.553872
q38		
4	-.5275709	-.5089004
q5		
No	-.4795077	-.467305
q22	-.4610605	-.4410858
q31	.4556527	.4047143

q56			
No	-.4482692	-.4026312	
q139	-.4189969	-.4118033	
q73			
No	-.3565698	-.3368294	
q96			
No	-.3149921	-.2950566	
3.q16	-.263147	-.2278278	
q43			
No	-.2605833	-.2355772	
q50			
No	.2455526	.2307073	
q149			
No	-.2407299	-.2070948	
2.q84	-.2321074	-.2150944	
q109			
No	.1965246	.1530308	
q49			
No	.1937052	.1626059	
q159			
No	.1870743	.1771646	
q115			
No	.153256	.1272736	
3.q134	.1525998	.1418469	
q108			
No	-.1491124	-.1469051	
q91			
No	-.1475877	-.1252736	
q140			
No	-.142592	-.1192079	
2.q34	.1397604	.1155922	
q93	-.1379424	-.0964044	
q14			
No	-.1377481	-.0964684	
gender			
Female	-.1296337	-.1047897	
q153			
No	.1238655	.0835772	
q53	.1123144	.0813566	
q65			
3	.1035524	.084643	
q38			
3	.0922535	.086774	

q160		
No	-.0901901	-.0763008
q3		
No	-.082771	-.0574645
age	-.0707354	-.0590426
q102		
No	-.0578734	-.0427812
q44		
No	.0561402	.0301015
1.q110	-.0556488	-.0268615
q154		
No	.0492342	.0188979
q130		
No	-.0453674	-.0288351
q18	-.0428028	-.018666
q97		
No	.0427896	.021222
q142		
No	-.0427358	-.0188524
q75		
No	-.0341663	-.0011199
q111	-.0333302	-.0294021
3.q95	-.0214817	
q65		
4	-.0213682	
q38		
2	.0197855	
0.q74	.0165583	
0.q33	-.016441	
q20	.0147089	
q94		
No	.0136563	.013323
q52	.0132519	
0.q138	-.0125278	
0.q71	.012269	
q13		
No	.0094304	.0027091
q105		
Fair	.0052163	.00026
0.q59	.0036381	
_cons	-3.55e-15	0

**Legend:**

b - base level  
e - empty cell  
o - omitted



Square-root lasso with plugin selected only 8 variables. Let's see what lasso does.

```
. lasso linear q104 $idemographics $ifactors $vlcontinuous,
> selection(plugin) rseed(1234)

Computing plugin lambda ...
Iteration 1:   lambda = .1470747   no. of nonzero coef. =    8
Iteration 2:   lambda = .1470747   no. of nonzero coef. =   11
Iteration 3:   lambda = .1470747   no. of nonzero coef. =   13
Iteration 4:   lambda = .1470747   no. of nonzero coef. =   15
Iteration 5:   lambda = .1470747   no. of nonzero coef. =   15

Lasso linear model                               No. of obs =          914
                                                No. of covariates =      277

Selection: Plugin heteroskedastic
```

ID	Description	lambda	No. of nonzero coef.	In-sample R-squared	BIC
* 1	selected lambda	.1470747	15	0.1549	5206.721

\* lambda selected by plugin formula assuming heteroskedastic errors.

Lasso with plugin selected a few more—15 variables in total. We can see from the in-sample  $R^2$  that the predictive capabilities of models using plugin are much lower than those using CV. We expect this because plugin estimators were designed as a tool for inferential models, not for prediction.

◀

## Stored results

`sqrtlasso` stores the following in `e()`:

Scalars

<code>e(N)</code>	number of observations
<code>e(N_clust)</code>	number of clusters
<code>e(k_allvars)</code>	number of potential variables
<code>e(k_nonzero_sel)</code>	number of nonzero coefficients for selected model
<code>e(k_nonzero_cv)</code>	number of nonzero coefficients at CV mean function minimum
<code>e(k_nonzero_serule)</code>	number of nonzero coefficients for one-standard-error rule
<code>e(k_nonzero_min)</code>	minimum number of nonzero coefficients among estimated $\lambda$ 's
<code>e(k_nonzero_max)</code>	maximum number of nonzero coefficients among estimated $\lambda$ 's
<code>e(k_nonzero_bic)</code>	number of nonzero coefficients at BIC function minimum
<code>e(lambda_sel)</code>	value of selected $\lambda^*$
<code>e(lambda_gmin)</code>	value of $\lambda$ at grid minimum
<code>e(lambda_gmax)</code>	value of $\lambda$ at grid maximum
<code>e(lambda_last)</code>	value of last $\lambda$ computed
<code>e(lambda_cv)</code>	value of $\lambda$ at CV mean function minimum
<code>e(lambda_serule)</code>	value of $\lambda$ for one-standard-error rule
<code>e(lambda_bic)</code>	value of $\lambda$ at BIC function minimum
<code>e(ID_sel)</code>	ID of selected $\lambda^*$
<code>e(ID_cv)</code>	ID of $\lambda$ at CV mean function minimum
<code>e(ID_serule)</code>	ID of $\lambda$ for one-standard-error rule
<code>e(ID_bic)</code>	ID of $\lambda$ at BIC function minimum
<code>e(cvm_min)</code>	minimum CV mean function value
<code>e(cvm_serule)</code>	CV mean function value at one-standard-error rule
<code>e(devratio_min)</code>	minimum deviance ratio
<code>e(devratio_max)</code>	maximum deviance ratio
<code>e(L1_min)</code>	minimum value of $\ell_1$ -norm of penalized unstandardized coefficients
<code>e(L1_max)</code>	maximum value of $\ell_1$ -norm of penalized unstandardized coefficients
<code>e(L2_min)</code>	minimum value of $\ell_2$ -norm of penalized unstandardized coefficients

<code>e(L2_max)</code>	maximum value of $\ell_2$ -norm of penalized unstandardized coefficients
<code>e(ll_sel)</code>	log-likelihood value of selected model
<code>e(n_lambda)</code>	number of $\lambda$ 's
<code>e(n_fold)</code>	number of CV folds
<code>e(stop)</code>	stopping rule tolerance
<b>Macros</b>	
<code>e(cmd)</code>	<code>sqrtlasso</code>
<code>e(cmdline)</code>	command as typed
<code>e(depvar)</code>	name of dependent variable
<code>e(allvars)</code>	names of all potential variables
<code>e(allvars_sel)</code>	names of all selected variables
<code>e(alwaysvars)</code>	names of always-included variables
<code>e(othervars_sel)</code>	names of other selected variables
<code>e(post_sel_vars)</code>	all variables needed for post-square-root lasso
<code>e(clustvar)</code>	name of cluster variable
<code>e(lasso_selection)</code>	selection method
<code>e(sel_criterion)</code>	criterion used to select $\lambda^*$
<code>e(plugin_type)</code>	type of plugin $\lambda$
<code>e(model)</code>	linear, logit, poisson, or probit
<code>e(title)</code>	title in estimation output
<code>e(rngstate)</code>	random-number state used
<code>e(properties)</code>	b
<code>e(predict)</code>	program used to implement <code>predict</code>
<code>e(marginsnotok)</code>	predictions disallowed by <code>margins</code>
<b>Matrices</b>	
<code>e(b)</code>	penalized unstandardized coefficient vector
<code>e(b_standardized)</code>	penalized standardized coefficient vector
<code>e(b_postselection)</code>	postselection coefficient vector
<b>Functions</b>	
<code>e(sample)</code>	marks estimation sample

In addition to the above, the following is stored in `r()`:

<b>Matrices</b>	
<code>r(table)</code>	matrix containing the coefficients with their standard errors, test statistics, $p$ -values, and confidence intervals

Note that results stored in `r()` are updated when the command is replayed and will be replaced when any `r`-class command is run after the estimation command.

## Methods and formulas

This section provides the methods and formulas for the methods implemented in `sqrtlasso`. The square-root lasso was derived by [Belloni and Chernozhukov \(2011\)](#).

Methods and formulas are presented under the following headings:

*Notation*  
*Plugin estimators*

## Notation

`sqrtlasso` estimates the parameters by finding the minimum of a penalized objective function. The penalized objective function is

$$Q = \sqrt{\frac{1}{N} \sum_{i=1}^N w_i (y_i - \beta_0 - \mathbf{x}_i \boldsymbol{\beta}')^2} + \frac{\lambda}{N} \sum_{j=1}^p \kappa_j |\beta_j| \quad (1)$$

where  $N$  is the number of observations,  $w_i$  are observation-level weights,  $\beta_0$  is the intercept,  $\mathbf{x}_i$  is the  $1 \times p$  vector of covariates,  $\boldsymbol{\beta}$  is the  $1 \times p$  vector of coefficients,  $\lambda$  is the lasso penalty parameter that must be  $\geq 0$ , and  $\kappa_j$  are coefficient-level weights.

When  $\lambda = 0$ , there is no penalty term, and  $Q$  is the objective function for a version of the reweighted least-squares estimator.

By default, the coefficient-level weights  $\kappa_j$  are 1. The heteroskedastic plugin estimator uses coefficient-level weights that differ from 1. In addition, they may be set to other values using option `penaltywt()`.

`sqrtlasso` uses the coordinate descent algorithm to minimize  $Q$  for a given value of  $\lambda$ . See [Friedman et al. \(2007\)](#) for an introduction to the coordinate descent algorithm.

The numerical problem is made much easier and more stable by standardizing all the covariates to have mean 0 and standard deviation 1. The standardization also removes  $\beta_0$  from the problem.

The grid of values for  $\lambda$  is specified as described in [Methods and formulas](#) in [\[LASSO\] lasso](#).

As with lasso and elastic net, we need to select a value of  $\lambda^*$ . The available selection methods are `selection(cv)` (CV, the default), `selection(plugin)`, `selection(bic)`, and `selection(none)`. The square-root lasso was designed to facilitate the derivation of the plugin estimator for  $\lambda^*$  discussed below. CV and BIC for the square-root lasso use the same algorithm as the regular lasso; see [Methods and formulas](#) in [\[LASSO\] lasso](#) for details.

If option `cluster()` is specified, the penalized objective function with clusters is

$$Q = \sqrt{\frac{1}{N_{\text{clust}}} \sum_{i=1}^{N_{\text{clust}}} \left\{ \frac{1}{T_i} \sum_{t=1}^{T_i} w_{it} (y_{it} - \beta_0 - \mathbf{x}_{it} \boldsymbol{\beta}')^2 \right\}} + \frac{\lambda}{N_{\text{clust}}} \sum_{j=1}^p \kappa_j |\beta_j|$$

where  $N_{\text{clust}}$  is the total number of clusters and  $T_i$  is the number of observations in cluster  $i$ . For the  $t$ th observation in cluster  $i$ ,  $w_{it}$  is its observational level weight,  $y_{it}$  is the dependent variable, and  $\mathbf{x}_{it}$  are the covariates.

## Plugin estimators

The same formula for the plugin estimator is used for the homoskedastic and the heteroskedastic cases with the square-root lasso. This result is essentially why the square-root lasso was derived; see [Belloni, Chernozhukov, and Wang \(2011\)](#). In the homoskedastic case, the coefficient-level weights are all 1 because the variables have been normalized. In the heteroskedastic case, the coefficient-level weights are estimated using algorithm 1, which comes from [Belloni, Chernozhukov, and Wang \(2011, 769\)](#).

The formula for  $\lambda^*$  is

$$\lambda_{\text{sqrt}} = 2c\sqrt{N} \Phi^{-1} \left( 1 - \frac{\gamma}{2p} \right)$$

where  $c = 1.1$  per the recommendation of [Belloni and Chernozhukov \(2011\)](#),  $N$  is the sample size,  $\gamma$  is the probability of not removing variable  $x_j$  when it has a coefficient of 0, and  $p$  is the number of candidate covariates in the model. Also, per the recommendation of [Belloni and Chernozhukov \(2011\)](#), we set  $\gamma = 0.1 / \ln[\max\{p, N\}]$ .



**Algorithm 1: Estimate coefficient-level weights for the heteroskedastic case**

1. Remove the mean and standardize each of the covariates  $x_j$  to have variance one. Remove the mean from  $y$ .
2. Initialize the maximum number of iterations  $K = 15$ , initialize the iteration counter  $k = 0$ , and initialize each of the coefficient-level weights,

$$\kappa_{j,0} = \max_{1 \leq i \leq N} |x_{ij}| \quad \text{for } j \in \{1, \dots, p\}$$

3. Update  $k = k + 1$ , and estimate the square-root lasso coefficients  $\widehat{\beta}$  using the coefficient-level weights  $\kappa_{j,k-1}$  and the above formula for  $\lambda_{\text{sqrt}}$ .
4. Update the coefficient-level weights,

$$\kappa_{j,k} = \max \left\{ 1, \frac{\sqrt{\frac{1}{N} \sum_{i=1}^N (x_{ij} r_i)^2}}{\sqrt{\frac{1}{N} \sum_{i=1}^N r_i^2}} \right\}$$

where  $r_i = y_i - \mathbf{x}_i \widehat{\beta}$ .

## References

Belloni, A., and V. Chernozhukov. 2011. High dimensional sparse econometric models: An Introduction. In *Inverse Problems of High-Dimensional Estimation*, ed. P. Alquier, E. Gautier, and G. Stoltz, 121–156. Berlin: Springer.

Belloni, A., V. Chernozhukov, and L. Wang. 2011. Square-root lasso: Pivotal recovery of sparse signals via conic programming. *Biometrika* 98: 791–806. <https://doi.org/10.1093/biomet/asr043>.

Friedman, J. H., T. J. Hastie, H. Höfling, and R. J. Tibshirani. 2007. Pathwise coordinate optimization. *Annals of Applied Statistics* 1: 302–332. <https://doi.org/10.1214/07-AOAS131>.

Hastie, T. J., R. J. Tibshirani, and M. Wainwright. 2015. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Boca Raton, FL: CRC Press.

## Also see

- [LASSO] [lasso postestimation](#) — Postestimation tools for lasso for prediction
- [LASSO] [elasticnet](#) — Elastic net for prediction and model selection
- [LASSO] [lasso](#) — Lasso for prediction and model selection
- [R] [regress](#) — Linear regression
- [U] [20 Estimation and postestimation commands](#)