Title

stata.com

lasso fitting — The process (in a nutshell) of fitting lasso models

Description Remarks and examples Also see

Description

This entry describes the process of fitting lasso models.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Introduction

If you are to fit lasso models successfully, you need to understand how the software computes them. There are options you can specify to modify the process and specifying them is sometimes necessary just to find the solution. This entry explains the process.

The process of fitting lasso models applies to the three commands that directly fit lasso and related models:

`lasso`	`sqrtlasso`	`elasticnet`

The lasso inferential commands

`dsregress`	`poregress`	`xporegress`
`dslogit`	`pologit`	`xpologit`
`dspoisson`	`popoisson`	`xpopoisson`
	`poivregress`	`xpoivregress`

fit multiple lasso models under the hood, and you may want to try using different lasso model-selection methods with these commands. If you do, then this entry is also for you. All the options described here can be used with the inferential commands to specify different lasso model-selection methods and modify the settings that control the lasso-fitting process.

Model selection

Fitting lasso models requires that the software fit lots of models behind the scenes from which one will be selected. The trick to minimizing the time needed is to avoid fitting unnecessary models, while ensuring that you fit the right one so it is there to select.

Lasso has a way of ordering models. They are ordered on scalar parameter λ defined over 0 to $+\infty$. λ is a parameter of the penalty function. For ordinary lassos, the penalty is λ times the sum of the absolute values of the coefficients of the normalized variables. Every possible model has a λ associated with it. When λ is large, the penalty is large, and the model has few or no variables. Models with smaller λ's have more variables.

We can think of lasso as fitting model(λ), where λ varies over a range, and then selecting one of them.

Which model do we select? That depends on the problem. Do we want a good model for prediction or a parsimonious model that better reflects the "true" model?

One method of selection is called cross-validation (CV), and it works well for prediction. The criterion is the CV function $f(\lambda)$, an estimate of the out-of-sample prediction error, which we minimize. The model for the λ that minimizes the CV function is the selected model.

To find λ^* that minimizes $f(\cdot)$, and thus find the corresponding model(λ^*), we need to fit models with λ's near to λ^* to be certain that we identified the minimum. Only nearby λ's would be good enough if your models were fit on infinite-observation datasets because this perfect $f(\lambda)$ is globally convex. Because your datasets will be finite, the empirically estimated function will jiggle around its Platonic ideal, and that means that you will need to fit additional models to be reasonably certain that the one you select is the one that corresponds to λ^*.

Another method, adaptive lasso, works well when the goal is to find parsimonious models—models with fewer variables in them—that might better reflect the true model. Adaptive lasso starts by finding the CV solution and then, using weights on the coefficients in the penalty function, does another lasso and selects a model that has fewer variables.

A third method is called plugin lasso. It is faster than CV or adaptive lasso. It is not just faster, it will be approaching the finish line while the others are still working out where the finish line is. It is faster because it does not minimize $f(\cdot)$. It uses an iterative formula to calculate the smallest λ that is large enough to dominate the estimation error in the coefficients. Plugin will not produce as low an out-of-sample prediction error as CV, but it will produce more parsimonious models than CV. Plugin is the default selection method for the inferential commands because it is so fast. For prediction, CV has better theoretical properties.

We discuss CV, adaptive, and plugin lassos below, and we discuss a fourth selection method that we call none. None is a do-it-yourself (DIY) method. It calculates model(λ) over a range of λ's and stops. You then examine them and choose one.

The process

Step 1. Set the grid range

Step 1 consumes virtually no time, but the total time steps 2 and 3 consume will depend on the grid that step 1 sets. The grid that steps 2 and 3 will search and calculate over will range from λ_{gmax} to λ_{gmin} and have G points on it.

Large values of λ correspond to models with few or no variables in them. Small values correspond to models with lots of variables. Given any two values of λ, λ_1, and λ_2,

$$\lambda_1 > \lambda_2 \text{ usually implies that \# of variables in model 1} \leq \text{\# of variables in model 2}$$

Most of us think of parameters as running from smallest to largest, say, 0 to $+\infty$, but with λ, you will be better served if you think of them as running from $+\infty$ to 0.

The grid does not start at $+\infty$, it starts at λ_{gmax}. The software does an excellent job of setting λ_{gmax}. It sets λ_{gmax} to the smallest λ that puts no variables in the model. You cannot improve on this. There is no option for resetting λ_{gmax}.

The software does a poor job of setting λ_{gmin}. There simply does not exist a scheme to set it optimally. If we are to identify the minimum of the CV function, $f(\lambda^*)$, λ_{gmin} must be less than λ^*. That is difficult to do because obviously we do not know the value of λ^*.

Computing models for small λ's is computationally expensive because fitting a model for a small λ takes longer than fitting a model for a larger λ. Our strategy is to hope we set λ_{gmin} small enough and then stop iterating over λ as soon as we are assured that we have found the minimum of the CV function. If we did not set λ_{gmin} small enough, the software will tell us this.

The initial grid is set to

$$\lambda_{\text{gmax}},\ \lambda_2,\ \lambda_3,\ \ldots,\ \lambda_{\text{gmin}} \quad (\lambda_{\text{gmin}} \text{ too small we hope})$$

The software sets λ_{gmin} to *ratio* $\times\ \lambda_{\text{gmax}}$, where *ratio* defaults to 1e–4 when $p < N$, where p is the number of potential covariates and N the number of observations. When $p \geq N$, the default is 1e–2.

You can reset *ratio* with the `grid(, ratio(#))` option, or you can reset λ_{gmin} directly by specifying `grid(, min(#))`.

Finally, in addition to setting *ratio* or λ_{gmin}, you can reset the number of points on the grid. It is set to 100 by default, meaning the initial grid will be

$$\lambda_{\text{gmin}} = \lambda_1,\ \lambda_2,\ \lambda_3,\ \ldots,\ \lambda_{99},\ \lambda_{\text{gmin}} = \lambda_{100}$$

You can reset the number of points by specifying `grid(#)`. You can specify the number of points and a value for *ratio* by typing `grid(#, ratio(#))`. See [LASSO] **lasso options**.

Step 2. Fit the model for next lambda in grid

We have a grid range λ_{gmax} to λ_{gmin} and number of points on the grid, which we will simply denote by their indices:

$$\lambda_1,\ \lambda_2,\ \lambda_3,\ \ldots,\ \lambda_{99},\ \lambda_{100}$$

The software obtains the models

$$\text{model}(\lambda_1),\ \text{model}(\lambda_2),\ \text{model}(\lambda_3),\ \ldots,\ \text{model}(\lambda_{100})$$

By "obtains", we mean that the software chooses the variables that appear in each one. The software proceeds from left to right. The first model, model(λ_1), has no variables in it and was easy to find. Once found, model(λ_1) provides the starting point for finding model(λ_2), and model(λ_2) provides the starting point for finding model(λ_3), and so on. Working from the previous model to obtain the next model is known as a warm start in the literature. Regardless of what the technique is called, this is why the software does not allow you to set a different λ_{gmax} for λ_1. To calculate model(λ) for a small value of λ , the software has to work its way there from previous model(λ) results.

The grid points are not equally spaced. The grid points are not

$$\begin{aligned}\lambda_1 &= \lambda_{\text{gmax}}\\ \lambda_2 &= \lambda_1 - \Delta\\ \lambda_3 &= \lambda_2 - \Delta\\ \lambda_4 &= \lambda_3 - \Delta\\ &\vdots\end{aligned}$$

The grid points are instead chosen so that $\ln\lambda$ is equally spaced, which you can think of as the λ's being closer together as they get smaller:

$$\begin{aligned}\lambda_1 &= \lambda_{\text{gmax}}\\ \lambda_2 &= \lambda_1 - \Delta_1\\ \lambda_3 &= \lambda_2 - \Delta_2, \quad 0 < \Delta_2 < \Delta_1\\ \lambda_4 &= \lambda_3 - \Delta_3, \quad 0 < \Delta_3 < \Delta_2\\ &\vdots\end{aligned}$$

Model estimation involves not only choosing the variables that appear in the model but also estimating their coefficients as well.

The computation will not usually be carried out all the way to λ_{100}. Because small λ's are computationally expensive, we want to stop before we get to λ_{100}. There are two criteria for stopping. The first is when we have identified the minimum of the CV function.

After we fit model(λ_1), we compute the value of the CV function for λ_1, $f(\lambda_1)$. Likewise after fitting model(λ_2), we compute $f(\lambda_2)$. For early λ's, typically we have $f(\lambda_k) > f(\lambda_{k+1})$. Now if we see

$$f(\lambda_{k-1}) > f(\lambda_k) < f(\lambda_{k+1})$$

λ_k might give the minimum of the CV function. It is possible that the CV function is bouncing around a bit, and it might not be the true minimum. We discuss how we declare a minimum in more detail in the next section.

For now, assume that we have properly identified a minimum. We are done, and we need not do any more estimations of model(λ).

But what if we do not find a minimum of the CV function? Sometimes, the CV function flattens out and stays flat, barely changing and only slowly declining with each smaller λ.

As the software proceeds from the calculation of model(λ_{k-1}) to model(λ_k), it calculates the relative differences in the in-sample deviances between models:

$$\frac{\text{deviance}\{\text{model}(\lambda_{k-1})\} - \text{deviance}\{\text{model}(\lambda_k)\}}{\text{deviance}\{\text{model}(\lambda_{k-1})\}}$$

This relative difference is a measure of how much predictive ability is added by proceeding to model(λ_k). If it is small, that suggests the difference between the CV function values $f(\lambda_{k-1})$ and $f(\lambda_k)$ will be small, and changes in the function for smaller λ's smaller yet. So we think it is likely that we have gone far enough.

If the relative difference is less than 1e–5, the software sets the selected $\lambda^* = \lambda_{\text{stop}} = \lambda_k$ and stops estimating models for more λ's. The output tells you that the selected λ^* was determined by this stopping rule. This means model(λ^*) does not give the minimum of the CV function, but we believe something close to it.

If you do not want this default behavior, there are three things you can do. The first is to change the value of the stopping rule tolerance. If you want to use 1e–6 instead of 1e–5, specify

```
. lasso y x1 x2 ..., stop(1e-6)
```

With a smaller tolerance, it will iterate over more λ's, giving a greater chance that a minimum might be identified.

The second possibility is to turn off the early stopping rule by setting the tolerance to 0. If there is a minimum that can be identified, this will find it.

```
. lasso y x1 x2 ..., stop(0)
```

If, however, the CV function flattens out and stays flat, specifying `stop(0)` might mean that the software iterates to the end of the λ grid, and this might take a long time.

A third choice is to specify

```
. lasso y x1 x2 ..., selection(cv, strict)
```

This is the same as the default behavior in this case, except that it throws an error! The suboption `strict` says that if we do not find a minimum, end with an error. This is useful when using `selection(cv)` with the inferential commands. It alerts us to the fact that we did not find a minimum, and it leaves the lasso behind, so we can plot the CV function and decide what to do next.

Selection method none

If you specify `selection(none)` instead of `selection(cv)`, the software stops when the stopping rule tolerance is reached or when the end of the λ grid is reached.

You can specify `selection(none)` when you want to gain a feel for how the number of included variables changes over λ or if you want to choose λ^* yourself. We provide a suite of postestimation commands for this purpose:

- `lassoknots` shows you a table of the λ's and the properties of the models.
- `lassocoef` lists the variables in the selected model. It can compare multiple models in the same table.
- `lassoselect` lets you choose a model to be treated as the selected model(λ^*).
- `lassogof` evaluates the selected model. It can also compare multiple models in the same table.

What you do not have, however, is the CV function and other CV-based measures of fit, which allow you to evaluate how well models predict and so make an informed choice as to which model should be model(λ^*).

There is another way. Do not specify `selection(none)`, specify `selection(cv)` or `selection(adaptive)`. The above postestimation functions will work, and you can, based on your own criteria if you wish, select the model for yourself.

Step 3. Identifying a minimum of the CV function

The minimum is identified when there are values of $f(\cdot)$ that rise above it on both sides. For example, consider the following case:

$$f(\lambda_1) > f(\lambda_2) > \cdots > f(\lambda_{49})$$

and

$$f(\lambda_{49}) < f(\lambda_{50}) < f(\lambda_{51}) < f(\lambda_{52})$$

For linear models, $f(\lambda_{49})$ is an identified minimum, and the software sets $\lambda^* = \lambda_{49}$. Linear models require that there be three smaller λ's with larger CV function values by a relative difference of `cvtolerance(#)` or more.

Because the CV functions for nonlinear models are not as smooth, `lasso` has a stricter criterion for declaring that a minimum of the CV function is found than it has for linear models. `lasso` requires that five smaller λ's to the right of a nominal minimum be observed with larger CV function values by a relative difference of `cvtolerance(#)` or more.

If you want more assurance that you have found a minimum, you can change `cvtolerance(#)` to a larger value from its default of 1e–3.

```
. lasso y x1 x2 ..., cvtolerance(1e-2)
```

Making the tolerance larger typically means that a few more model(λ)'s are estimated to find the required three (or five) with CV function values larger than the minimum by this tolerance.

The software provides three options that control how λ^* is set when a identified minimum is not found. They work like this:

	λ^* is set to		
Options	Case 1	Case 2	Case 3
`selection(cv, strict)`	λ_{cvmin}	error	error
`selection(cv, stopok)`	λ_{cvmin}	λ_{stop}	error
`selection(cv, gridminok)`	λ_{cvmin}	λ_{stop}	λ_{gmin}

Case 1 is an identified minimum.
Case 2 is falling over range, stopping rule tolerance reached.
Case 3 is falling over range, stopping rule tolerance not reached.

λ_{cvmin} is the identified minimum of the CV function $f(\cdot)$.
λ_{stop} is the λ that meant the stopping rule tolerance.
λ_{gmin} is the last λ in the grid.
error indicates that λ^* is not set, and the software issues an error message.

You may specify only one of the three options. `selection(cv, stopok)` is the default if you do not specify one.

We emphasize that these options affect the setting of λ^* only when an identified minimum is not found.

`selection(cv, stopok)` is the default and selects $\lambda^* = \lambda_{\text{stop}}$ when the stopping rule tolerance was reached.

`selection(cv, strict)` is the purist's option. λ^* is found only when a minimum is identified. Otherwise, the software issues a minimum-not-found error.

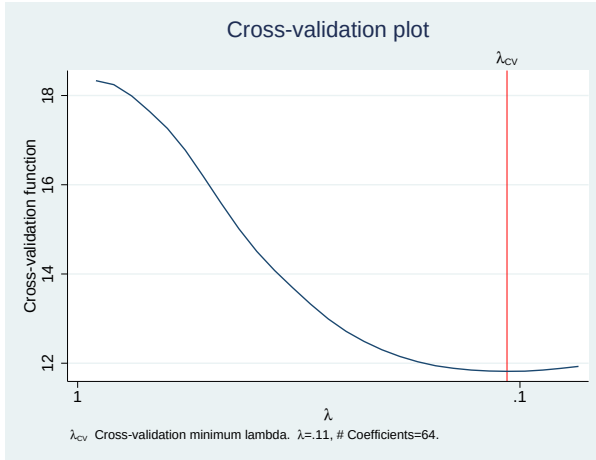
`selection(cv, gridminok)` is an option that has an effect only when the early stopping rule tolerance is not reached. We have fallen off the right edge of the grid without finding an identified minimum. λ^* is set to λ_{gmin}. There is no theoretical justification for this rule. Practically, it means that λ_{gmin} was set too large. We should make it smaller and refit the model.

Plotting the CV function

Run `lasso` if you have not already done so. After you do, there are two possible outcomes. The software ended by setting a λ^*, thus selecting a model, or it did not set a λ^*. You will have no doubts as to which occurred because when λ^* is not set, the software ends with an error message and a nonzero return code. Note that even when it ends with a nonzero return code, results of the lasso are left behind.

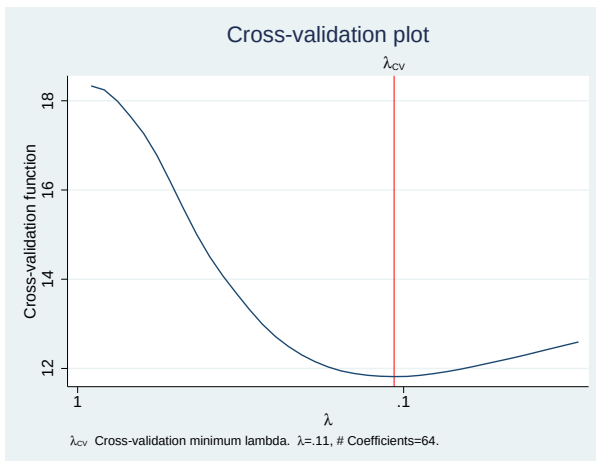
Regardless of how estimation ended, graph the CV function, $f(\cdot)$. It is easy to do. Type `cvplot` after running a lasso. Here is one:

```
. lasso linear y x1 x2 ...
. cvplot
```



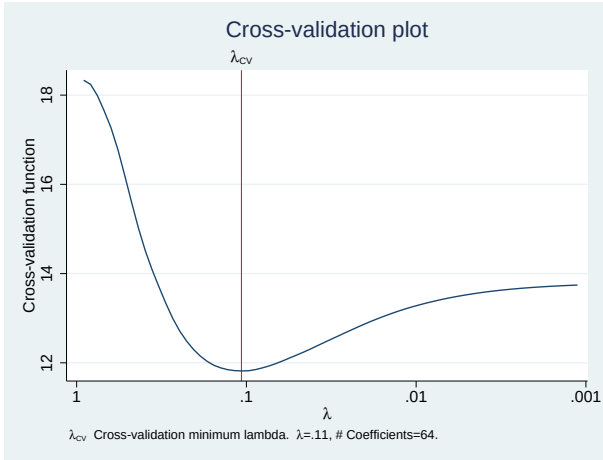
This lasso identified a minimum of the CV function. It identified the minimum and stopped iterating over λ. If we want to see more of the CV function, we can set `cvtolerance(#)` to a larger value.

```
. lasso linear y x1 x2 ..., cvtolerance(0.05)
. cvplot
```



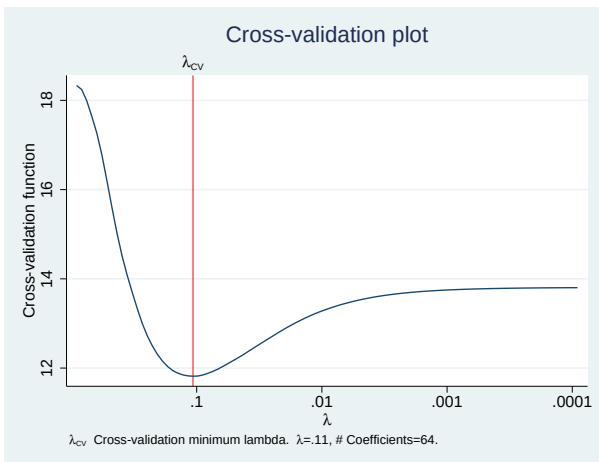
If we want to see more of the CV function, we can specify `selection(cv, alllambdas)`. When the `alllambdas` suboption is specified, estimation does not end when a minimum of the CV function is found. In fact, it estimates model(λ) for all λ's first and then computes the CV function because this is slightly more computationally efficient if we are not stopping after identifying a minimum.

```
. lasso linear y x1 x2 ..., selection(cv, alllambdas)
. cvplot
```



Actually, `alllambdas` is a lie. In this case, it estimated only 73 λ's. It ended when the stopping rule tolerance was reached. If we really want to see all 100 λ's, we need to turn off the stopping rule.

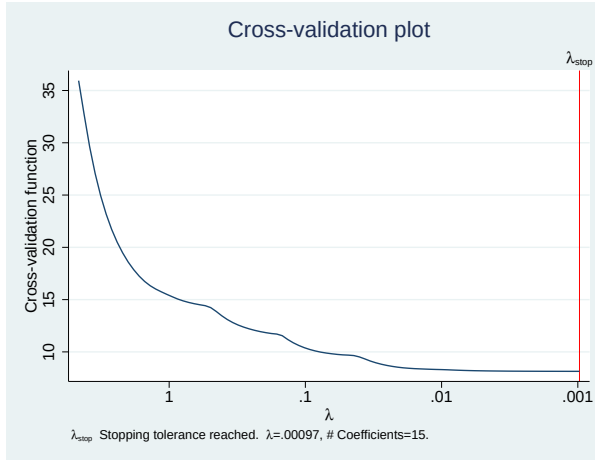
```
. lasso linear y x1 x2 ..., selection(cv, alllambdas) stop(0)
. cvplot
```



That is a plot of all 100 λ's. Clearly, in this case, the default behavior worked fine to identify a minimum.

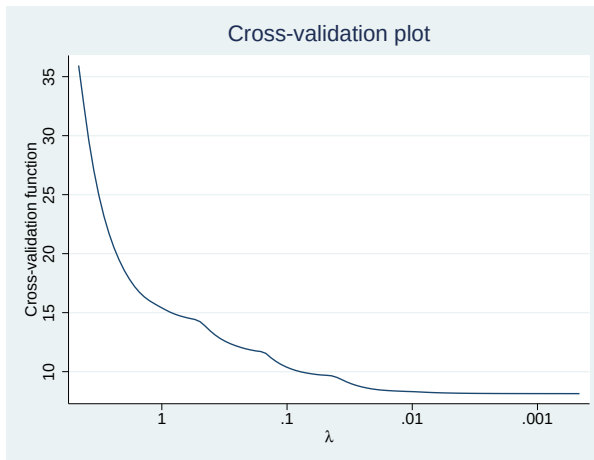
Here is an example of a CV function for which a minimum was not identified. The stopping rule tolerance was reached instead.

```
. lasso linear z w1 w2 ...
. cvplot
```



To try more λ's in a search for a minimum, we turn off the stopping rule

```
. lasso linear z w1 w2 ..., stop(0)
. cvplot
```



It went to the end of the grid without finding a minimum. The default stopping rule tolerance usually works fine. Setting `stop(0)` typically burns more computer time without identifying a minimum.

Selecting another model

Imagine that you have successfully found the model that minimizes the CV function, $f(\lambda)$, the estimate of out-of-sample prediction error. If your interest is in prediction, the model that minimizes the CV function really is best. If your interest is in model selection, however, you may want to look at alternatives that are close in the out-of-sample prediction sense.

You can use `lassoknots` to see a table of the λ's where variables were added or dropped. These are called knot points.

You can then use `lassoselect` to choose one of the models. This command sets λ^* to the λ you specify. Once you have selected a model, you can use all of `lasso`'s postestimation features on it. And then, if you wish, you can `lassoselect` another model. If you use `estimates store` after each `lassoselect`, you can compare multiple models side by side using `lassogof`.

See [LASSO] **lassoselect** for an example.

What exactly is CV?

We are done discussing using CV as a selection method, and yet we have never discussed CV itself. CV is about using one subsample of data to fit models and another to evaluate their prediction error.

Here are the details. The $f(\cdot)$ function is an estimate of the out-of-sample prediction error, and the function is calculated using CV. The method starts by dividing the data into K partitions called folds. Once that is done, for each fold k,

1. model(λ) is fit on all observations except those in fold k.
2. that result is used to predict the outcome in fold k.
3. steps 1 and 2 are repeated for each fold.
4. the prediction error is then averaged over all folds, which is to say, all observations. This is $f(\lambda)$.

Option `selection(cv, folds(#))` sets K, and `folds(10)` is used by default.

Adaptive lasso

In *Plotting the CV function*, we looked at a graph of the CV function for which $f(\lambda)$ had a long flat region and the stopping rule selected λ^*. We explained that you could use the `lasso` DIY postestimation commands to change the selected model to one with fewer variables in it.

Adaptive lasso is another approach for obtaining parsimonious models. It is a variation on CV, and in fact, for each step, it uses CV. It uses the CV-selected model(λ^*) as a starting point and then amplifies the important coefficients and attenuates the unimportant ones in one or more subsequent lassos that also use CV.

For the second lasso, variables not selected in the first lasso's model(λ^*) are dropped, and the penalty term uses weights equal to the inverse of the absolute value of the coefficients from model(λ^*). The justification being that important coefficients are large and unimportant ones, small. (Variables are standardized so that comparison of coefficient size makes sense.) These weights tend to drive small coefficients to zero in the second lasso. So the selected model from the second lasso almost always has fewer variables than the selected model from the first lasso.

Plugin selection

CV selects model(λ^*) such that $f(\lambda)$ is minimized. Adaptive is a variation on CV. It selects a final model(λ^*) that minimizes a more restricted $f(\lambda)$.

Plugins—`selection(plugin)`—are a whole different thing. Parameter λ still plays a role, but $f(\cdot)$ does not. Instead, the λ^* that determines model(λ^*) is produced by direct calculation using the plugin function, $\lambda^* = g(\cdot)$. The function returns the smallest value of λ that is large enough to dominate the estimation error in the coefficients.

No search over λ is required, nor is a grid necessary. This makes plugin the fastest of the methods provided. It is fast, but it is not instantaneous. The plugin formula is solved iteratively, and if it is trying to calculate a small value for λ^*, it can take a little time. Those small λ's again!

Plugin's selected model(λ^*) are almost always more parsimonious than the minimum-$f(\lambda)$ models selected by CV. Plugin will not produce models with as low an out-of-sample prediction error as CV, but it tends to select the most important variables and can be proven to do so for many data-generation processes. Plugin is popular when the problem is model selection instead of out-of-sample prediction.

Also see

[LASSO] **lasso** — Lasso for prediction and model selection

[LASSO] **lasso examples** — Examples of lasso for prediction

[LASSO] **cvplot** — Plot cross-validation function after lasso

[LASSO] **lassocoef** — Display coefficients after lasso estimation results

[LASSO] **lassogof** — Goodness of fit after lasso for prediction

[LASSO] **lassoknots** — Display knot table after lasso estimation

[LASSO] **lassoselect** — Select lambda after lasso