makespline — Spline generation

Description Quick start Menu Syntax
Options Remarks and examples Stored results Methods and formulas
References Also see

Description

`makespline` generates a set of variables that form B-spline, piecewise polynomial spline, and restricted cubic spline basis functions from a list of existing variables. B-spline and piecewise polynomial spline bases may be first, second, or third order, with knots at percentiles of the data or uniformly spaced over the range of the variables. Restricted cubic splines, also known as natural splines, may only be of third order.

Quick start

Generate variables from `x1` and `x2` that form third-order B-spline basis functions, with one knot at the median of each variable

```
makespline bspline x1 x2
```

Same as above, but with three knots at the 25th, 50th, and 75th percentiles

```
makespline bspline x1 x2, knots(3)
```

Same as above, but use second-order B-splines

```
makespline bspline x1 x2, knots(3) order(2)
```

Generate variables that form a linear spline for `x1` with knots at 10 and 20

```
makespline piecewise x1, knotslist(10 20) order(1)
```

Same as above, but do not rescale `x1` before creating spline variables

```
makespline piecewise x1, knotslist(10 20) order(1) norescalevars
```

Generate variables that form third-order piecewise polynomial splines for `x1` and `x2`, with knots at their 25th, 50th, and 75th percentiles

```
makespline piecewise x1 x2, knots(3)
```

Same as above, but with three knots at evenly spaced points over the range of `x1` and of `x2`

```
makespline piecewise x1 x2, knots(3) uniformknots
```

Specify values of knots in matrix `K` to generate variables forming restricted cubic splines

```
makespline rcs x1 x2, knotsmat(K)
```

Generate variables that form a linear spline for `x1` without rescaling the values of `x1`

```
makespline linear x1
```

Menu

Data > Create or change data > Other variable-creation commands > Spline generation

Syntax

`makespline` *basis* *varlist* [*if*] [*in*] [*weight*] [, *options*]

basis	Description
`bspline`	B-spline
`piecewise`	piecewise polynomial spline
`rcs`	restricted cubic spline
`linear`	linear spline—`piecewise` basis of order 1 without rescaling variables

options	Description
Main	
`bsepsilon(#)`	specify the distance (#) from the variable's boundary for B-spline knot placement; default is `bsepsilon(0.01)`
`local`	generate first-order polynomial spline variables centered around adjacent knots
`harrell`	place knots according to percentiles in Harrell (2001); only for `rcs` basis
`order(#)`	use a spline basis of order #; default is `order(3)`
`knots(#)`	use a spline basis function with # knots
`knotslist(`*knotvals*`)`	use knots specified in *knotvals*
`knotsmat(`*matname*`)`	use knots in matrix *matname*
`distinct(#)`	set minimum number of distinct values required for variables used to construct splines to #; default is `distinct(10)`
`replace`	replace existing variables having the same names as the new basis and rescaled variables, if they exist
`norescalevars`	do not rescale variables before generating spline basis
`uniformknots`	place knots at evenly spaced points over the range of each variable; default is placement at percentiles
`float`	set type for generated variables to float instead of double
`basis(`*stub* | *newvarlist*`)`	store elements of spline basis function using *stub* or *newvarlist*
`rescale(`*stub* | *newvarlist*`)`	store rescaled values of variables using *stub* or *newvarlist*

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

`fweight`s, `aweight`s, and `iweight`s are allowed; see **[U] 11.1.6 weight**.

Options

Main

`bsepsilon(#)` specifies the distance from the boundary of the variable where B-spline knots may be placed. The default is `bsepsilon(0.01)`.

`local` specifies that a basis function for a first-order piecewise polynomial be generated with variables centered around adjacent knots. When splines are generated for only one variable and used in estimation, the regression coefficients measure slopes for the intervals defined by knots.

`harrell` specifies that knots be placed according to the percentiles recommended in Harrell (2001, 23). This option may be used only with *basis* `rcs` and when specifying 3 to 7 knots.

`order(`*#*`)` specifies that a spline of order # be used as the basis. `order()` may be 1, 2, or 3 for *basis* `bspline` and *basis* `piecewise`. `order()` may only be 3 for *basis* `rcs`. For *basis* `linear` or when the `local` option is specified, `order()` may only be 1. The default is `order(3)`, cubic splines.

`knots(`*#*`)` specifies that a spline or B-spline basis function with # interior knots be used. The number of knots must be an integer greater than or equal to 1. The maximum number of knots is either 4,096 or two-thirds of the sample size, whichever is smaller. Also, the number of knots must be less than the number of distinct values in the variable used to generate the basis function. The default is `knots(1)` if exact knot values are not specified using `knotslist()` or `knotsmat()`. For *basis* `rcs`, the default is `knots(3)`, and the number of knots must be 3 or greater.

`knotslist(`*knotvals*`)` specifies in *knotvals* the values of knots to be used for each variable. The knot values must be specified in the order of *varlist*, and a backslash (\) must be used to separate knots for different variables. For example, if splines are generated for `x1` and `x2`, the knots may be specified as `knotslist(20 40 60 \ 5 10 15)`.

`knotsmat(`*matname*`)` specifies that, in *matname*, the knots for each variable be the values in each row. The number of knots should be the same for each variable, and there must be as many rows as there are variables. If rows of *matname* are not labeled with *varname*s, then rows are assumed to be in the order of *varlist*.

`distinct(`*#*`)` specifies the minimum number of distinct values required for the variables used to construct the basis functions. Intuitively, using discrete variables for continuous interpolation is difficult to justify. # specifies the number of distinct values necessary for a variable to be considered continuous. The default is `distinct(10)`.

`replace` specifies that the variables generated to form the basis function be replaced. If `basis(`*stub* | *newvarlist*`)` or `rescale(`*stub* | *newvarlist*`)` are specified, the variables named with *stub*, or those listed in *newvarlist*, are replaced. Otherwise, variables with the default names are replaced.

`norescalevars` specifies that the original values of the variables in *varlist* be used to generate the basis function. By default, variables are first rescaled to $[0, 1]$. `norescalevars` may not be used with *basis* `bspline` or *basis* `linear`.

`uniformknots` specifies that knots be placed at evenly spaced points over the range of each variable. The default is placement at percentiles of each of the specified variables.

`float` specifies that variables be generated as floats. Because of numerical precision and stability, the default is `double`.

`basis(`*stub* | *newvarlist*`)` specifies that the elements of the basis function be generated with the specified names.

When *stub* is specified, this prefix is used to generate enumerated variables for each element of the basis function.

When *newvarlist* is specified, variables with these names are generated for the elements of the basis function.

`rescale(`*stub* | *newvarlist*`)` specifies that the rescaled variables used to generate the basis function be stored with the specified names. This option applies only to *basis* `piecewise` and *basis* `rcs`.

When *stub* is specified, this prefix is used to generate enumerated variables for the rescaled variables.

When *newvarlist* is specified, variables with these names are generated for the rescaled variables.

Remarks and examples

stata.com

makespline generates new variables that form B-splines, piecewise polynomial splines, and restricted cubic splines from existing variables. Splines allow for different low-order polynomials in different regions of the original variables, and they approximate a smooth function by continuously connecting these low-order polynomials. Knots define the boundaries of the regions.

The standard piecewise polynomial variables created by makespline piecewise allow the functions to be linear, quadratic, or cubic in each region. makespline linear provides a convenient method for creating linear splines from the original variables, without rescaling. This is useful when you wish to directly interpret regression coefficients in the metric of the original variables. The terms in the standard piecewise polynomial spline function can be highly collinear and may be numerically unstable when used in estimation. B-splines, which can be created by makespline bspline, avoid this problem by creating orthogonal spline terms. For an introduction to piecewise polynomial splines and B-splines, see *Piecewise polynomial splines and B-splines* in [R] **npregress intro**. makespline rcs creates restricted cubic splines, also known as natural splines, in which the function is linear before the first knot, cubic between adjacent knots, and linear again after the last knot. This can improve performance in the tails over the standard cubic spline.

In addition to selecting the type of spline, makespline allows you to specify the location of knots—the locations where the function changes. You can specify the number of knots you wish to allow, and makespline will place the knots based on percentiles of the data or uniformly spaced across the range of values in the data. Alternatively, you can specify the exact values at which you wish the knots to be placed.

Regardless of the type of spline, we can refer to our newly created variables as a spline basis function. A basis is a collection of terms that can approximate a smooth function arbitrarily well. A basis function, such as one of the spline functions created by makespline, is a subset of the basis terms that can be used to approximate the mean function.

The basis function variables generated by makespline are useful for nonparametric and semi-parametric estimation. For instance, makespline can be used when we want to fit models such as

$$\mathbf{y} = \mathbf{x}_1\beta + g\left(\mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_k\right) + \epsilon \tag{1}$$

In the expression above, the outcome $\mathbf{y}$, the covariates $\mathbf{x}_1, \ldots, \mathbf{x}_k$, and the unobservable ϵ are $n \times 1$ vectors of covariates. The function $g(\cdot)$ is unknown and $\mathbf{x}_1$ enters the model linearly. These types of models are commonly used when we are interested in estimating the effect of $\mathbf{x}_1$ on the mean of $\mathbf{y}$. We are agnostic about the functional form in which the controls, $\mathbf{x}_2, \ldots, \mathbf{x}_k$, enter the model, but to get a precise estimate of the effect of $\mathbf{x}_1$, we need a reliable approximation of $g(\cdot)$. We may use makespline to generate the basis functions that best approximate $g(\cdot)$ and then use the basis functions to fit the model in (1).

For instance, we can generate basis functions with basis as the stub name:

```
makespline bspline x2-x5, basis(basis)
```

This would generate a third-order B-spline basis function for each of the variables in x2-x5, with knots at the medians of x2-x5. Each of the basis functions would consist of five variables; see *Methods and formulas* in [R] **npregress series** for details.

Once we have these basis functions, we can fit the model in (1) by typing

```
regress y x1 c.(basis*)##c.(basis*)
```

where c.(basis*)##c.(basis*) specifies that the terms in the basis functions be included in the model on their own as well as interacted with each of the other terms.

Above, we are assuming that we constructed a good approximation of the unknown function $g(\cdot)$. We could go further and select from among these spline basis terms by using a technique such as lasso for prediction, described in [LASSO] **lasso**, or, if we are interested in inferences on estimated effects, a technique such as the partialing-out or double-selection lasso method, described in [LASSO] **Lasso inference intro**.

Let's say we are interested in getting a reliable estimate of the effect of `x1` on the mean of the outcome. We would type

```
poregress y x1, controls(c.(basis*)##c.(basis*))
```

The method used by the above command is partialing-out lasso, which selects from the elements of the basis function to provide an optimal approximation of $g(\cdot)$ while accounting for the implied model selection error. The result is an estimate of the effect of `x1` on the outcome with reliable standard errors.

Of course, the model does not have to be like the one presented in (1). It could be

$$\mathbf{y} = g\left(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\right) + \epsilon$$

or

$$\mathbf{y} = g\left(\mathbf{x}_1\right) + g\left(\mathbf{x}_2\right) + \cdots + g\left(\mathbf{x}_k\right) + \epsilon$$

or we might instead be interested in using the basis functions for visualization.

▷ Example 1: Generating and naming B-spline basis functions

Below, we generate a third-order B-spline basis function with one knot placed at the median. The basis function is constructed from the variable `price`.

```
. sysuse auto
(1978 automobile data)
. makespline bspline price
```

The basis function consists of these five variables:

```
. describe _*

Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
_bsp_1_1        double   %10.0g               B-spline basis term 1 for price
_bsp_1_2        double   %10.0g               B-spline basis term 2 for price
_bsp_1_3        double   %10.0g               B-spline basis term 3 for price
_bsp_1_4        double   %10.0g               B-spline basis term 4 for price
_bsp_1_5        double   %10.0g               B-spline basis term 5 for price
```

The default naming convention is to give the elements of the basis function a name that starts with _bsp and two subscripts. The first subscript enumerates the basis functions, and the second subscript enumerates the elements within the basis function. For example, if we created basis functions for two variables, we would obtain the following:

```
. makespline bspline price mpg, replace
. describe _*

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
_bsp_1_1        double  %10.0g                B-spline basis term 1 for price
_bsp_1_2        double  %10.0g                B-spline basis term 2 for price
_bsp_1_3        double  %10.0g                B-spline basis term 3 for price
_bsp_1_4        double  %10.0g                B-spline basis term 4 for price
_bsp_1_5        double  %10.0g                B-spline basis term 5 for price
_bsp_2_1        double  %10.0g                B-spline basis term 1 for mpg
_bsp_2_2        double  %10.0g                B-spline basis term 2 for mpg
_bsp_2_3        double  %10.0g                B-spline basis term 3 for mpg
_bsp_2_4        double  %10.0g                B-spline basis term 4 for mpg
_bsp_2_5        double  %10.0g                B-spline basis term 5 for mpg
```

If we want to change the stub name _bsp to autobasis, we could use the basis() option as follows:

```
. makespline bspline price mpg, basis(autobasis)
. describe auto*

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
autobasis_1_1   double  %10.0g                B-spline basis term 1 for price
autobasis_1_2   double  %10.0g                B-spline basis term 2 for price
autobasis_1_3   double  %10.0g                B-spline basis term 3 for price
autobasis_1_4   double  %10.0g                B-spline basis term 4 for price
autobasis_1_5   double  %10.0g                B-spline basis term 5 for price
autobasis_2_1   double  %10.0g                B-spline basis term 1 for mpg
autobasis_2_2   double  %10.0g                B-spline basis term 2 for mpg
autobasis_2_3   double  %10.0g                B-spline basis term 3 for mpg
autobasis_2_4   double  %10.0g                B-spline basis term 4 for mpg
autobasis_2_5   double  %10.0g                B-spline basis term 5 for mpg
```

Alternatively, we could provide names for each of the variables that form a basis function. For example,

```
. makespline bspline mpg, basis(mpg1 mpg2 mpg3 mpg4 mpg5)
. describe mpg1-mpg5

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
mpg1            double  %10.0g                B-spline basis term 1 for mpg
mpg2            double  %10.0g                B-spline basis term 2 for mpg
mpg3            double  %10.0g                B-spline basis term 3 for mpg
mpg4            double  %10.0g                B-spline basis term 4 for mpg
mpg5            double  %10.0g                B-spline basis term 5 for mpg
```

◁

▷ Example 2: Generating and naming piecewise polynomial spline basis functions

Below, we generate a third-order piecewise polynomial spline with one knot at the median and show the variables we generated:

```
. makespline piecewise mpg

. describe *_sp*

Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
_rs_sp_1        double  %10.0g                mpg rescaled to [0,1]
_sp_1_1         double  %10.0g                Piecewise polynomial basis term 1
                                                for mpg
```

The only syntactical difference is that, after **makespline**, we specify **piecewise** instead of **bspline** to be the basis. **makespline** generates two variables in this case. They are the elements that are necessary to construct a basis function.

The default naming convention is to give the elements of the spline function a name that starts with **_sp** and has two subscripts. The first subscript enumerates the piecewise polynomial spline for a given variable, and the second subscript denotes the knot number. The rescaled variable starts with **_rs_sp** followed by a subscript denoting the element in the variable list.

Again, we may use a stub to modify the names that precede the subscripts, or we may specify a name for each new variable. Below, we also specified the names for the rescaled variables:

```
. makespline piecewise mpg price, basis(mpgsp pricesp) rescale(mpgrs pricers)

. describe mpgsp mpgrs pricesp pricers

Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
mpgsp           double  %10.0g                Piecewise polynomial basis term 1
                                                for mpg
mpgrs           double  %10.0g                mpg rescaled to [0,1]
pricesp         double  %10.0g                Piecewise polynomial basis term 1
                                                for price
pricers         double  %10.0g                price rescaled to [0,1]
```

The logic behind the variables generated is that they consist of all the elements needed to approximate the unknown function $g(\cdot)$ of the specified variables nonparametrically. In this case, a third-order piecewise polynomial spline approximation of $g(\cdot)$ consists of the levels, square, and cube of **mpgsp**, **mpgrs**, **pricesp**, and **pricers**. Specifically, to include the fully interacted basis functions in a model, we would need to include the term below in our specification:

```
c.(c.mpgrs##c.mpgrs##c.mpgrs mpgsp)##c.(c.pricers##c.pricers##c.pricers pricesp)
```

makespline simplifies this task by returning a local macro with the terms needed to fit $g(\cdot)$. The local macro has the name **r(regressors)**. In this case, it expands to the following:

```
. display "`r(regressors)'"
c.(c.mpgrs##c.mpgrs##c.mpgrs mpgsp)##c.(c.pricers##c.pricers##c.pricers pricesp)
```

Note that when you generate basis functions for more than one variable, as we did above, **r(regressors)** fully interacts these basis functions. These fully interacted basis functions can be included when fitting a model by adding **`r(regressors)'** to your list of covariates.

◁

▷ Example 3: Using makespline in semiparametric estimation

As we mentioned previously, basis functions are particularly useful for approximating unknown functions. For example, say we want to obtain the average marginal effect of x1 on the conditional mean of the continuous outcome y. We have two controls, x2 and x3, but it is unclear whether they enter the model linearly or with another functional form.

To approximate the unknown function of x2 and x3, we construct two B-spline basis functions with eight knots each. We use the simulated dataset and then the makespline bspline command:

```
. use https://www.stata-press.com/data/r18/splines, clear
(Simulated data)

. makespline bspline x2 x3, knots(8)
```

This yields basis functions with 12 elements. Once you fully interact the two basis functions, you get 168 regressors. Using all of them to approximate the unknown function would not be a sound idea. Thus, we will use poregress to perform partialing-out lasso linear regression. This estimator will select from the 168 covariates to provide a good approximation to the unknown function and at the same time provide a reliable estimate of the marginal effect of interest.

Rather than interact the basis terms manually, we can simply refer to the macro r(regressors), which contains the full interaction of the basis functions:

```
. poregress y x1, controls(`r(regressors)')

Estimating lasso for y using plugin
Estimating lasso for x1 using plugin

Partialing-out linear model          Number of obs               =      5,000
                                     Number of controls          =        168
                                     Number of selected controls =         19
                                     Wald chi2(1)                =    3535.78
                                     Prob > chi2                 =     0.0000
```

y	Coefficient	Robust std. err.	z	P>|z|	[95% conf.	interval]
x1	2.951242	.049632	59.46	0.000	2.853965	3.048519

```
Note: Chi-squared test is a Wald test of the coefficients of the variables
      of interest jointly equal to zero. Lassos select controls for model
      estimation. Type lassoinfo to see number of selected variables in each
      lasso.
```

We obtain an average marginal effect of 2.95.

A researcher does not know the true value of the effect; however, we do. These are simulated data. The model is given by

$$\mathbf{y} = 3\mathbf{x}_1 + 3\sin\{3(\mathbf{x}_2 - \mathbf{x}_3)\} + \epsilon$$

The unknown function of $\mathbf{x}_2$ and $\mathbf{x}_3$ is complex, yet we obtained a precise estimate of the average marginal effect.

◁

▷ Example 4: Using makespline for estimation and graphing

It is common to use linear splines to create a graph after estimation. The knots of a regressor define a piecewise polynomial that can be visualized conditional on the values of other covariates.

Below, we study the effect of mileage in miles per gallon (mpg) on car prices (price). We regress price on mpg, three linear polynomial basis terms defined by knots at the quartiles of mpg, and a dummy variable, foreign (1 if cars are foreign).

We first generate the variables that form the polynomial basis and then fit the regression.

```
. sysuse auto, clear
(1978 automobile data)
. makespline linear mpg, knots(3) basis(mpg)
. regress price mpg mpg_* i.foreign
      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(5, 68)        =     13.49
       Model |   316201619         5  63240323.8   Prob > F        =    0.0000
    Residual |   318863777        68  4689173.19   R-squared       =    0.4979
-------------+----------------------------------   Adj R-squared   =    0.4610
       Total |   635065396        73  8699525.97   Root MSE        =    2165.4

------------------------------------------------------------------------------
       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
         mpg |  -1330.299   213.0425    -6.24   0.000    -1755.419   -905.1798
     mpg_1_1 |   1698.953   622.5153     2.73   0.008     456.7432    2941.163
     mpg_1_2 |  -622.9298   651.5686    -0.96   0.342    -1923.115    677.2551
     mpg_1_3 |   139.4188   277.0762     0.50   0.616    -413.4783    692.3158
             |
     foreign |
    Foreign  |   1676.381   609.4723     2.75   0.008     460.1983    2892.564
       _cons |   28796.47   3449.408     8.35   0.000     21913.28    35679.65
------------------------------------------------------------------------------
```

The regression line for our model is given by the following command:

```
generate double xb = _b[_cons]   + _b[1.foreign]*foreign           +  ///
                     mpg*_b[mpg] + (mpg>18)*(mpg-18)*_b[mpg_1_1]   +  ///
                                   (mpg>20)*(mpg-20)*_b[mpg_1_2]   +  ///
                                   (mpg>25)*(mpg-25)*_b[mpg_1_3]
```

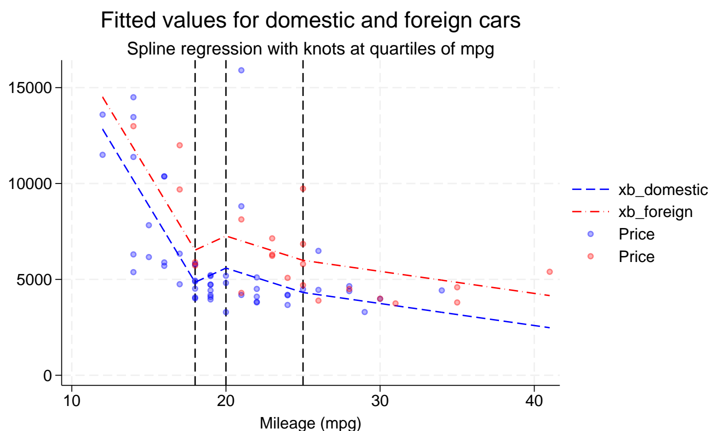
The effect of mpg changes at the knots. If mpg is less than or equal to 18, it is _b[mpg]; if it is greater than 18 but less than or equal to 20, it is (_b[mpg] + _b[mpg_1_1]); if it is greater than 20 but less than or equal to 25, it is (_b[mpg] + _b[mpg_1_1] + _b[mpg_1_2]); and if it is greater than 25, it is (_b[mpg] + _b[mpg_1_1] + _b[mpg_1_2] + _b[mpg_1_3]).

We can plot regression lines for foreign and domestic cars. We first generate the predictions for foreign and domestic cars.

```
. generate xb_domestic = _b[_cons]  + mpg*_b[mpg]
>                                   + (mpg>18)*(mpg-18)*_b[mpg_1_1]
>                                   + (mpg>20)*(mpg-20)*_b[mpg_1_2]
>                                   + (mpg>25)*(mpg-25)*_b[mpg_1_3]
. generate xb_foreign = _b[_cons] + _b[1.foreign] + mpg*_b[mpg]
>                                 + (mpg>18)*(mpg-18)*_b[mpg_1_1]
>                                 + (mpg>20)*(mpg-20)*_b[mpg_1_2]
>                                 + (mpg>25)*(mpg-25)*_b[mpg_1_3]
```

Then we plot both regression lines referencing the placement of the knots with vertical lines. In the graph, we also include the values of the dependent variable. We can inspect graphically how the effect of mpg differs across the regions defined by the knots.

```
. twoway line xb_domestic mpg,
>            lcolor(blue) lpattern(dash) sort                    ||
>        line xb_foreign mpg,
>            lcolor(red) lpatter(dash_dot) sort                  ||
>        scatter price mpg if foreign==0, mcolor(blue%30)        ||
>        scatter price mpg if foreign==1, mcolor(red%30)
>        xline(18 20 25)
>        title(Fitted values for domestic and foreign cars)
>        subtitle(Spline regression with knots at quartiles of mpg)
```



◁

Stored results

makespline stores the following in r():

Scalars

r(N_knots)	number of knots
r(local)	1 if local was specified, 0 otherwise
r(bsepsilon)	distance from variable's boundary for B-spline knot placement

Macros

r(basis)	spline type used to generate basis function
r(regressors)	regressors formed from basis functions
r(basisnames#)	variable names of basis function for variable #
r(wtype)	weight type
r(wexp)	weight expression

Matrices

r(minmax)	minimum and maximum of all variables
r(knots)	matrix of knots

Methods and formulas

See *Methods and formulas* in [R] **npregress series** for piecewise polynomial spline and B-spline computation.

When the `local` option is specified, let V_i, $i = 1, \dots, n$, be the variables to be created; k_i, $i = 1, \dots, n-1$, be the corresponding knots; and $\mathcal{V}$ be the original variable rescaled to be in $[0,1]$. Then

$$
\begin{aligned}
V_1 &= \min(\mathcal{V}, k_1) \\
V_i &= \max\Big\{ \min(\mathcal{V}, k_i), k_{i-1} \Big\} - k_{i-1} \quad i = 2, \dots, n-1 \\
V_n &= \max(\mathcal{V}, k_{n-1}) - k_{n-1}
\end{aligned}
$$

When the `rcs` basis is specified, let k_i, $i = 1, \dots, n$, be the knot values; V_i, $i = 1, \dots, n-1$, be the variables to be created; and $\mathcal{V}$ be the original variable rescaled to be in $[0,1]$. Then

$$
\begin{aligned}
V_1 &= \mathcal{V} \\
V_{i+1} &= \frac{(\mathcal{V} - k_i)^3_+ - (k_n - k_{n-1})^{-1}\{(\mathcal{V} - k_{n-1})^3_+ (k_n - k_i) - (\mathcal{V} - k_n)^3_+ (k_{n-1} - k_i)\}}{(k_n - k_1)^2} \\
& \quad i = 1, \dots, n-2
\end{aligned}
$$

where

$$
(u)_+ = \begin{cases} u, \text{ if } u > 0 \\ 0, \text{ if } u \le 0 \end{cases}
$$

When the `harrell` option is specified, the knots are placed using the percentiles recommended in Harrell (2001, 23). These percentiles are based on the chosen number of knots as follows:

No. of knots	Percentiles						
3	10	50	90				
4	5	35	65	95			
5	5	27.5	50	72.5	95		
6	5	23	41	59	77	95	
7	2.5	18.33	34.17	50	65.83	81.67	97.5

References

Chetverikov, D., D. Kim, and D. Wilhelm. 2018. Nonparametric instrumental-variable estimation. *Stata Journal* 18: 937–950.

de Boor, C. 2001. *A Practical Guide to Splines*. Rev. ed. New York: Springer.

Eubank, R. L. 1999. *Nonparametric Regression and Spline Smoothing*. 2nd ed. New York: Dekker.

Hansen, B. E. 2009. University of Wisconsin–Madison, ECON 718, NonParametric Econometrics, Spring 2009, course notes. Last visited on 2019/01/15. https://www.ssc.wisc.edu/~bhansen/718/718.htm.

———. 2018. Econometrics. https://www.ssc.wisc.edu/~bhansen/econometrics/Econometrics.pdf.

Harrell, F. E., Jr. 2001. *Regression Modeling Strategies: With Applications to Linear Models, Logistic Regression, and Survival Analysis*. New York: Springer.

Li, Q., and J. S. Racine. 2007. *Nonparametric Econometrics: Theory and Practice*. Princeton, NJ: Princeton University Press.

Schoenberg, I. J., ed. 1969. *Approximations with Special Emphasis on Spline Functions*. New York: Academic Press.

Schumaker, L. L. 2007. *Spline Functions: Basic Theory*. 3rd ed. Cambridge: Cambridge University Press.

Also see

[R] **npregress series** — Nonparametric series regression

[R] **npregress series postestimation** — Postestimation tools for npregress series

[R] **npregress intro** — Introduction to nonparametric regression

[R] **kdensity** — Univariate kernel density estimation

[R] **lpoly** — Kernel-weighted local polynomial smoothing