The Stata Journal

The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go "beyond the Stata manual" in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

The *Stata Journal* is indexed and abstracted by *CompuMath Citation Index*, *Current Contents/Social and Behavioral Sciences*, *RePEc: Research Papers in Economics*, *Science Citation Index Expanded* (also known as *SciSearch*, *Scopus*, and *Social Sciences Citation Index*.

For more information on the *Stata Journal*, including information for authors, see the webpage

http://www.stata-journal.com

Subscriptions are available from StataCorp, 4905 Lakeway Drive, College Station, Texas 77845, telephone 979-696-4600 or 800-STATA-PC, fax 979-696-4601, or online at

http://www.stata.com/bookstore/sj.html

Subscription rates listed below include both a printed and an electronic copy unless otherwise mentioned.

U.S. and Canada		Elsewhere	
1-year subscription	$ 79	1-year subscription	$115
2-year subscription	$155	2-year subscription	$225
3-year subscription	$225	3-year subscription	$329
3-year subscription (electronic only)	$210	3-year subscription (electronic only)	$210
1-year student subscription	$ 48	1-year student subscription	$ 79
1-year university library subscription	$ 99	1-year university library subscription	$135
2-year university library subscription	$195	2-year university library subscription	$265
3-year university library subscription	$289	3-year university library subscription	$395
1-year institutional subscription	$225	1-year institutional subscription	$259
2-year institutional subscription	$445	2-year institutional subscription	$510
3-year institutional subscription	$650	3-year institutional subscription	$750

Back issues of the *Stata Journal* may be ordered online at

http://www.stata.com/bookstore/sjj.html

Individual articles three or more years old may be accessed online without charge. More recent articles may be ordered online.

http://www.stata-journal.com/archives.html

The *Stata Journal* is published quarterly by the Stata Press, College Station, Texas, USA.

Address changes should be sent to the *Stata Journal*, StataCorp, 4905 Lakeway Drive, College Station, TX 77845, USA, or emailed to sj@stata.com.

The *Stata Journal*, electronic version (ISSN 1536-8734) is a publication of Stata Press. Stata, Stata, Stata Press, Mata, Mata, and NetCourse are registered trademarks of StataCorp LP.

The Stata Journal (2012)
12, Number 3, pp. 479–504

Sensible parameters for univariate and multivariate splines

Roger B. Newson
National Heart and Lung Institute
Imperial College London
London, UK
r.newson@imperial.ac.uk

Abstract. The package `bspline`, downloadable from Statistical Software Components, now has three commands. The first, `bspline`, generates a basis of Schoenberg B-splines. The second, `frencurv`, generates a basis of reference splines whose parameters in the regression model are simply values of the spline at reference points on the X axis. The recent addition, `flexcurv`, is an easy-to-use version of `frencurv` that generates reference splines with automatically generated, sensibly spaced knots. `frencurv` and `flexcurv` now have the additional option of generating an incomplete basis of reference splines, with the reference spline for a baseline reference point omitted or set to 0. This incomplete basis can be completed by adding the standard unit vector to the design matrix and can then be used to estimate differences between values of the spline at the remaining reference points and the value of the spline at the baseline reference point. Reference splines therefore model continuous factor variables as indicator variables (or "dummies") model discrete factor variables. The method can be extended in a similar way to define factor-product bases, allowing the user to estimate factor-combination means, subset-specific effects, or even factor interactions involving multiple continuous or discrete factors.

Keywords: sg151_2, bspline, flexcurv, frencurv, polynomial, spline, B-spline, interpolation, linear, quadratic, cubic, multivariate, factor, interaction

1 Introduction

Splines are frequently used to model nonlinear predictive relationships between an X variable and a Y variable, especially when the fundamental mechanisms are unknown but the effect of X on Y is still thought to be important. For a natural number k, a kth-degree spline is defined using a sequence of positions (or knots) on the X axis and has these features: 1) in any interval between two consecutive knots, the spline is equal to a polynomial of degree k; and 2) the first $k-1$ derivatives of the spline are continuous at each knot. Therefore, a spline of degree 0 is a step function with steps at the knots, a spline of degree 1 is a continuous function linearly interpolated between the knots, and splines of degree $k > 1$ are interpolated between the knots as polynomials of degree k. Many Stata packages exist for implementing spline models—notably, the official Stata command `mkspline` (see [R] **mkspline**) for linear and restricted cubic splines; the `splinegen` package developed by Patrick Royston and Gareth Ambler for step, linear, and restricted cubic splines (Royston and Sauerbrei 2007); and the xblc package for graphing and tabulating linear splines, and unrestricted and restricted cubic splines (Orsini and Greenland 2011).

A frequent problem with spline models is interpreting the parameters. Usually, spline models are fit to the data using a basis of spline vectors whose linear combinations form a space of splines of the specified degree. The spline vectors are typically included in the design matrix for a linear or generalized linear model, with or without other vectors representing the effect on Y of covariates or factors other than X. The parameters corresponding to the vectors of the spline basis are then the coordinates of the fitted spline in these vectors and can be estimated in the usual way, with confidence limits. However, these coordinates are frequently not easy to explain to nonmathematical colleagues.

The bspline package, downloadable from Statistical Software Components (SSC), was designed to make this explanation easier. The original version was described in Newson (2000) and contained two commands. The bspline command generates a basis of unrestricted Schoenberg B-splines, whose parameters are not easy to interpret. The frencurv command generates a basis of reference splines, which span the same unrestricted spline space and whose corresponding parameters are values of the spline at reference points on the X axis or possibly are differences or ratios between these reference values (also known as "effects"). The method of frencurv was originally developed, pre-Stata, to model the time series of hospital asthma admissions, which are highly seasonal. An application appears in Newson et al. (1997).

The bspline package has since been upgraded, notably, with the addition of a third command, flexcurv, which is designed as a user-friendly front-end for frencurv. The package also has a manual (bspline.pdf) that is downloadable with the package as an auxiliary file and that documents the methods and formulas.

In this article, I describe the methods of the bspline package in greater detail, including the added improvements and the extension to multivariate splines with interactions. Section 2 illustrates the advantages of splines, with graphics generated using flexcurv. Section 3 gives the syntax of the package. Section 4 details the methods and formulas. (The casual reader may skip the highly technical sections 3 and 4, at least at first reading.) Finally, section 5 gives some practical examples using flexcurv.

2 Reasons for using splines

Splines are used to define a nonlinear regression model for an outcome Y with respect to a continuous predictor X when the underlying mechanism is not known. We will illustrate the advantages of splines in auto.dta, shipped with official Stata, whose observations correspond to car models. The Y variable will be mpg (mileage in miles per U.S. gallon of fuel), and the X variable will be weight (weight in U.S. pounds). The do-files used to create the figures in this section (demo1.do and demo2.do) are distributed as part of the online material for this article.

I will start by demonstrating linear splines. Figure 1 shows linear splines with 2, 3, 4, and 5 knots, evenly spaced from 1,500 to 5,100 pounds (inclusive). The spline with 2 knots (at 1,500 and 5,100 pounds) is a straight line over that domain. The spline with 3 knots (at 1,500, 3,300, and 5,100 pounds) is equal to a different straight line in each interval between consecutive knots and is continuous (but not differentiable) at the knots. The splines with 4 knots (at 1,500, 2,700, 3,900, and 5,100 pounds) and with 5 knots (at 1,500, 2,400, 3,300, 4,200, and 5,100 pounds) have the same features and are allowed to be progressively less linear as the number of knots increases. However, they are still undifferentiable at the knots, and this may seem "unnatural" to nonmathematical colleagues.

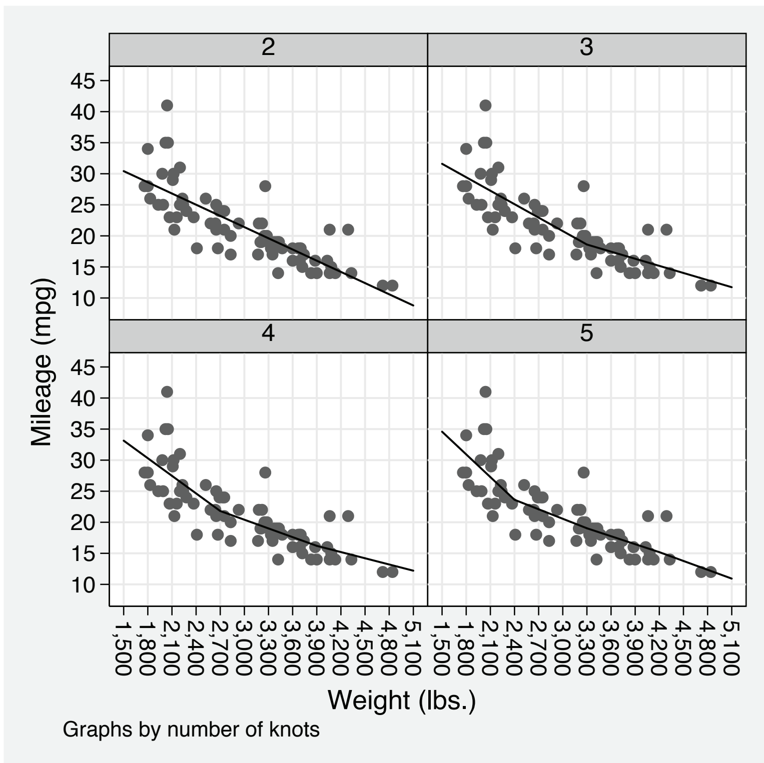

Figure 1. Linear splines for `mpg` with respect to `weight` with different numbers of knots

As a solution to this problem, we can vary the degree of the splines. Figure 2 illustrates splines of degree 0 (constant), 1 (linear), 2 (quadratic), and 3 (cubic) for mileage with respect to weight. The splines of degree 1, 2, and 3 each have five parameters, equal to their values at the reference points 1,500, 2,400, 3,300, 4,200, and 5,100 pounds. The constant spline (degree 0) only has four parameters, equal to its values at the first four of the reference points. The spline of degree 0 is simply a step function and is not even continuous (only right-continuous) at its knots. The spline of degree 1 is the linear spline in the lower right subgraph of figure 1 and, again, is continuous—but not differentiable—at its knots. However, the splines of degree 2 and 3 are differentiable throughout the domain, including at their knots, which cannot easily be located.

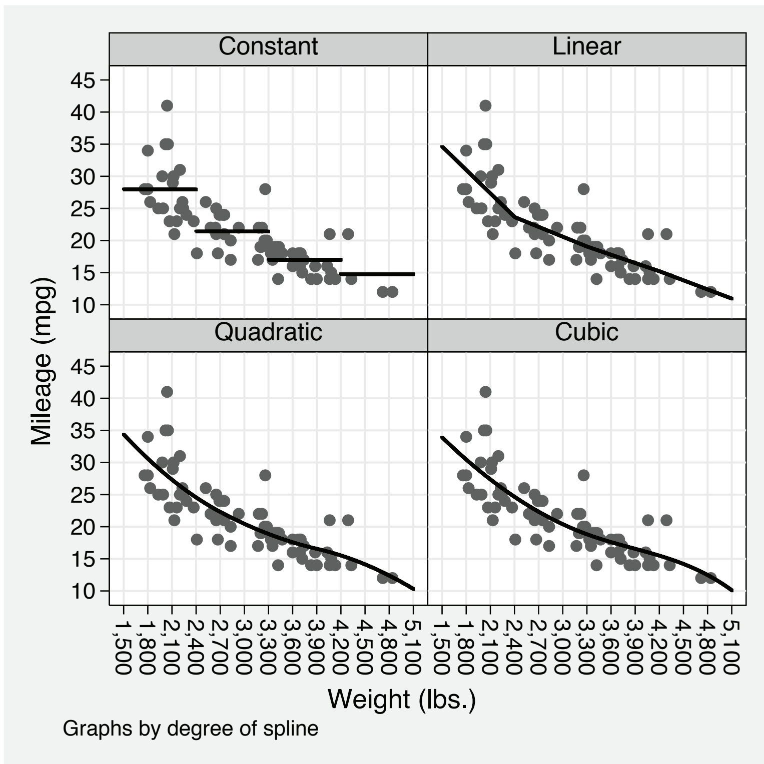

Figure 2. Splines of degree 0, 1, 2, and 3 for `mpg` with respect to `weight`

So from figure 1, we see that we can improve on a linear model by fitting separate linear models to intervals between knots, with the lines joined (or spliced) at the knots to form a spline. And then we see from figure 2 that we can improve further by upgrading to a quadratic or cubic spline to eliminate the visible joints that upset nonmathematical colleagues. These features make splines a good model family with which to model nonlinear predictive associations if the user has no specific mechanism in mind.

3 The bspline package

3.1 Syntax

`bspline` [*newvarlist*] [*if*] [*in*], `xvar(`*varname*`)` [`power(`*#*`)` `knots(`*numlist*`)` `noexknot` `generate(`*prefix*`)` `type(`*type*`)` `labfmt(`*format*`)` `labprefix(`*string*`)`]

`frencurv` [*newvarlist*] [*if*] [*in*], `xvar(`*varname*`)` [`power(`*#*`)` `refpts(`*numlist*`)` `noexref` `omit(`*#*`)` `base(`*#*`)` `knots(`*numlist*`)` `noexknot` `generate(`*prefix*`)` `type(`*type*`)` `labfmt(`*format*`)` `labprefix(`*string*`)`]

`flexcurv` [*newvarlist*] [*if*] [*in*], `xvar(`*varname*`)` [`power(`*#*`)` `refpts(`*numlist*`)` `omit(`*#*`)` `base(`*#*`)` `include(`*numlist*`)` `krule(regular|interpolate)` `generate(`*prefix*`)` `type(`*type*`)` `labfmt(`*format*`)` `labprefix(`*string*`)`]

3.2 Description

The `bspline` package contains three commands: `bspline`, `frencurv`, and `flexcurv`. `bspline` generates a basis of B-splines in the X variate based on a list of knots for use in the design matrix of a regression model. `frencurv` generates a basis of reference splines for use in the design matrix of a regression model, with the property that the parameters fit will be values of the spline at a list of reference points. `flexcurv` is an easy-to-use version of `frencurv` that generates reference splines with regularly spaced knots or with knots interpolated between the reference points. `frencurv` and `flexcurv` have the additional option of generating an incomplete basis of reference splines, which can be completed by the addition of the standard constant variable used in regression models. The splines are either given the names in the *newvarlist* (if present) or (more usually) generated as a list of numbered variables, prefixed by the `generate()` option.

3.3 Options for use with bspline and frencurv

`xvar(`*varname*`)` specifies the X variable on which the splines are to be based. `xvar()` is required.

`power(`*#*`)` (a nonnegative integer) specifies the power (or degree) of the splines. Examples are 0 for constant, 1 for linear, 2 for quadratic, 3 for cubic, 4 for quartic, or 5 for quintic. The default is `power(0)`.

`knots(`*numlist*`)` specifies a list of at least 2 knots on which the splines are to be based. If `knots()` is not specified, then `bspline` will initialize the list to the minimum and maximum of the `xvar()` variable, and `frencurv` will create a list of knots equal to the reference points (in the case of odd-degree splines such as linear, cubic, or quintic) or midpoints between reference points (in the case of even-degree splines such as constant, quadratic, or quartic). `flexcurv` does not have the `knots()` option, because it automatically generates a list of knots containing the required number of knots "sensibly" spaced on the `xvar()` scale.

`noexknot` specifies that the original knot list not be extended. If `noexknot` is not specified, then the knot list is extended on the left and right by a number of extra knots on each side specified by `power()`, spaced by the distance between the first and last two original knots, respectively. `flexcurv` does not have the `noexknot` option, because it specifies the knots automatically.

`generate(`*prefix*`)` specifies a prefix for the names of the generated splines, which (if there is no *newvarlist*) will be named *prefix*`1`, ..., *prefixN*, where N is the number of splines.

`type(`*type*`)` specifies the storage type of the splines generated (`float` or `double`). If *type* is specified as anything else (or if `type()` is not specified), then *type* is set to `float`.

`labfmt(`*format*`)` specifies the format used in the variable labels for the generated splines. If `labfmt()` is not specified, then the format is set to the format of the `xvar()` variable.

`labprefix(`*string*`)` specifies the prefix used in the variable labels for the generated splines. If `labprefix()` is not specified, then the prefix is set to `"Spline at "` for `flexcurv` and `frencurv` and to `"B-spline on "` for `bspline`.

3.4 Options for use with frencurv

`refpts(`*numlist*`)` specifies a list of at least two reference points, with the property that if the splines are used in a regression model, then the estimated parameters will be values of the spline at those points. If `refpts()` is not specified, then the list is initialized to two points equal to the minimum and maximum of the `xvar()` variable. If the `omit()` option is specified with `flexcurv` or `frencurv`, and the spline corresponding to the omitted reference point is replaced with a standard constant term in the regression model, then the estimated parameters will be relative values of the spline (differences or ratios) compared with the value of the spline at the omitted reference point.

`noexref` specifies that the original reference list not be extended. If `noexref` is not specified, then the reference list is extended on the left and right by a number of extra reference points on each side equal to `int(`*power*`/2)`, where *power* is the value of the `power()` option, spaced by the distance between the first and last two original reference points, respectively. If `noexref` and `noexknot` are both specified, then the number of knots must equal the number of reference points plus *power*`+1`. `flexcurv` does not have the `noexref` option, because it automatically chooses the knots and does not extend the reference points.

`omit(#)` specifies a reference point that must be present in the `refpts()` list (after any extension requested by `frencurv`) and whose corresponding reference spline will be omitted from the set of generated splines. If the user specifies `omit()`, then the set of generated splines will not be a complete basis of the set of splines with the specified power and knots, but can be completed by the addition of a constant variable equal to 1 in all observations. If the user then uses the generated splines as predictor variables for a regression command such as `regress` or `glm`, then the `noconstant` option should usually not be used. And if the omitted reference point is in the completeness region of the basis, then the intercept parameter `_cons` will be the value of the spline at the omitted reference point, and the model parameters corresponding to the generated splines will be differences between the values of the spline at the corresponding reference points and the value of the spline at the omitted reference point. (For the definition of the completeness region of a spline, see section 4.1.) If `omit()` is not specified, then the generated splines form a complete basis of the set of splines with the specified power and knots. If the user then uses the generated splines as predictor variables for a regression command, such as `regress` or `glm`, then the `noconstant` option should be used, and the fitted model parameters corresponding to the generated splines will be the values of the spline at the corresponding reference points.

`base(#)` is an alternative to `omit()` for use in Stata 11 or higher. It specifies a reference point that must be present in the `refpts()` list (after any extension requested by `frencurv`) and whose corresponding reference spline will be set to 0. If the user specifies `base()`, then the set of generated splines will not be a complete basis of the set of splines with the specified power and knots, but can be completed by the addition of a constant variable equal to 1 in all observations. The generated splines can then be used in the design matrix by a Stata 11 (or higher) estimation command.

3.5 Options for use with flexcurv only

Note that `flexcurv` uses all the options available to `frencurv` except for `knots()`, `noexknot`, and `noexref`.

`include(`*numlist*`)` specifies a list of additional numbers to be included within the boundaries of the completeness region of the spline basis, as well as the available values of the `xvar()` variable and the `refpts()` values (if provided). This allows the user to specify a nondefault infimum or supremum for the completeness region of the spline basis. If `include()` is not provided, then the completeness region will extend from the minimum to the maximum of the values available either in the `xvar()` variable or in the `refpts()` list.

`krule(regular|interpolate)` specifies a rule for generating knots based on the reference points, which may be `regular` (the default) or `interpolate`. If `regular` is specified, then the knots are spaced regularly over the completeness region of the spline. If `interpolate` is specified, then the knots are interpolated between the reference points in a way that produces the same knots as `krule(regular)` if the reference points are regularly spaced. Whichever `krule()` option is specified, any extra knots to the left of the completeness region are regularly spaced with a spacing equal to that between the first two knots of the completeness region, and any extra knots to the right of the completeness region are regularly spaced with a spacing equal to that between the last two knots of the completeness region. Therefore, `krule(regular)` specifies that all knots be regularly spaced whether or not the reference points are regularly spaced, whereas `krule(interpolate)` specifies that the knots be interpolated between the reference points in a way that will cause reference splines to be definable, even if the reference points are not regularly spaced.

3.6 Saved results

`bspline`, `frencurv`, and `flexcurv` save the following results in `r()`:

Scalars

`r(xsup)`	upper bound of completeness region
`r(xinf)`	lower bound of completeness region
`r(nincomp)`	number of X values out of completeness region
`r(nknot)`	number of knots
`r(nspline)`	number of splines
`r(power)`	power (or degree) of splines

Macros

`r(knots)`	final list of knots
`r(splist)`	*varlist* of generated splines
`r(labfmt)`	format used in spline variable labels
`r(labprefix)`	prefix used in spline variable labels
`r(type)`	storage type of splines (`float` or `double`)
`r(xvar)`	X variable specified by `xvar()` option

Matrices

`r(knotv)`	row vector of knots

`frencurv` and `flexcurv` save all the above results in `r()` and also save the following:

Scalars

`r(omit)`	omitted reference point specified by `omit()`
`r(base)`	base reference point specified by `base()`

Macros

`r(refpts)`	final list of reference points

Matrices

`r(refv)`	row vector of reference points

The result `r(nincomp)` is the number of values of the `xvar()` variable outside the completeness region of the space of splines defined by the reference splines or B-splines. The number lists `r(knots)` and `r(refpts)` are the final lists after any left and right extensions carried out by `bspline`, `frencurv`, or `flexcurv`; the vectors `r(knotv)` and `r(refv)` contain the same values in double precision (mainly for programmers). The scalars `r(xinf)` and `r(xsup)` are knots, such that the completeness region is `r(xinf)` $\leq x \leq$ `r(xsup)` for positive-degree splines and `r(xinf)` $\leq x <$ `r(xsup)` for zero-degree splines.

In addition, `bspline`, `frencurv`, and `flexcurv` save variable characteristics for the output spline basis variables. The characteristic *varname*`[xvar]` is set by `bspline`, frencurv, and flexcurv to be equal to the input X variable name set by xvar(). The characteristics *varname*[xinf] and *varname*[xsup] are set by bspline to be equal to the infimum and supremum, respectively, of the interval of X values for which the B-spline is nonzero. The characteristic *varname*[xvalue] is set by frencurv and flexcurv to be equal to the reference point on the X axis corresponding to the reference spline.

4 Methods and formulas

This section is intended mainly as a reference for the extensive family of methods and formulas used by the bspline package. Less mathematically minded readers may skip or skim through this section and progress to the examples.

4.1 B-splines

By definition, a kth-degree spline is defined with reference to a set of q knots $s_1 < s_2 < \cdots < s_q$, dividing the X axis into half-open intervals of the form $[s_i, s_{i+1})$. In each of those intervals, the regression is a kth-degree polynomial in X (usually a different one in each interval), but the polynomials in any two contiguous intervals have the same jth derivatives at the knot separating the two intervals for j from 0 to $k-1$. By convention, the 0th derivative is the function itself, so a spline of degree 0 is simply a right-continuous step function, and a first-degree spline is a simple linear interpolation of values between the knots.

Splines can be defined using plus functions. For a power k and a knot s, the kth-power plus function at s is defined as

$$P_k(x;s) = \begin{cases} (x-s)^k, & x \geq s \\ 0, & x < s \end{cases}$$

In Stata, we can calculate the plus functions of power 1 corresponding to a sequence of knots by using mkspline (see [R] **mkspline**) with the marginal option.

The plus functions are a basis for the space of splines: for any kth-degree spline $S(\cdot)$, with knots $s_1 < s_2 < \cdots < s_q$, there exists a q-vector α such that for any x,

$$S(x) = \sum_{j=1}^{q} \alpha_j P_k(x; s_j) \tag{1}$$

Based on (1), we might try to fit a spline model by creating a design matrix of plus functions and estimating the α_j. However, the high degree of correlation between the plus functions may cause wide confidence intervals. Moreover, it is not easy to explain to nonmathematical colleagues the parameters that these wide confidence intervals are intended to estimate.

B-splines are an alternative basis of the splines with a given set of knots, invented to solve the first of these problems. Ziegler (1969) defines the B-spline for a set of $k+2$ knots $s_1 < s_2 < \cdots < s_{k+2}$ as

$$B(x; s_1, \ldots, s_{k+2}) = (k+1) \sum_{j=1}^{k+2} \left\{ \prod_{1 \le h \le k+2, h \ne j} (s_h - s_j) \right\}^{-1} P_k(x; s_j) \tag{2}$$

The B-spline (2) is positive for x in the half-open interval $[s_1, s_{k+2})$ and is 0 for other x. If the s_j are part of an extended set of knots extending forward to $+\infty$ and backward to $-\infty$, then the set of B-splines based on sets of $k+2$ consecutive knots forms a basis of the set of all kth-degree splines defined on the full set of knots.

For the purposes of `bspline`, I have taken the liberty of redefining B-splines by scaling the $B(x; s_1, \ldots, s_{k+2})$ of (2) by a factor equal to the mean distance between two consecutive knots to arrive at the scale-invariant B-spline

$$\begin{aligned} A(x; s_1, \ldots, s_{k+2}) = & \frac{s_{k+2} - s_1}{k+1} B(x; s_1, \ldots, s_{k+2}) \\ & \begin{cases} \sum_{j=1}^{k+1} \prod_{h=1}^{k+2} \phi_{jh}(x), & \text{if} \quad s_1 \le x < s_{k+2} \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{3}$$

where the functions $\phi_{jh}(\cdot)$ are defined by

$$\phi_{jh}(x) = \begin{cases} 1, & \text{if} \quad h = j \\ (s_{k+2} - s_1)/(s_h - s_j), & \text{if} \quad h = j+1 \\ P_1(x; s_j)/(s_h - s_j), & \text{otherwise} \end{cases}$$

The scaled B-spline (3) has the advantage that it is dimensionless, being a sum of products of the dimensionless quantities $\phi_{jh}(x)$. That is to say, it is unaffected by the scale of units of the X axis and therefore has the same values, whether x is time in millennia or time in nanoseconds. By contrast, the original Ziegler B-spline (2) is expressed in units of x^{-1}. Therefore, if the scaled B-spline (3) appears in a design matrix, then its regression coefficient is expressed in units of the Y variate; in contrast, if the Ziegler B-spline (2) appears in a design matrix, then its regression coefficient is expressed in Y units multiplied by X units and will be difficult to interpret—even for a mathematician.

Given n data points, a Y variate, an X covariate, and a set of $q+k+1$ consecutive knots $s_h < \cdots < s_{h+q} < \cdots < s_{h+q+k}$, we can regress the Y variate with respect to a kth-degree spline in X by defining a design matrix $\mathbf{V}$, with one row for each of the n data points and one column for each of the first q knots, such that

$$\mathbf{V}_{ij} = A(X_i; s_{h+j-1}, \ldots, s_{h+j+k}) \tag{4}$$

We can then regress the Y variate with respect to the design matrix $\mathbf{V}$ and compute a vector β of regression coefficients, such that $\mathbf{V}\beta$ is the fitted spline. The parameter β_j measures the contribution to the fitted spline of the B-spline originating at the knot

s_{h+j-1} and terminating at the knot s_{h+j+k}. There will be no stability problems such as we are likely to have with the original plus-function basis because each B-spline is bounded and localized in its effect.

It is important to define enough knots. If the sequence of knots $\{s_j\}$ extends to $+\infty$ on the right and $-\infty$ on the left, then the kth-degree B-splines $A(\cdot; s_{h+j-1}, \dots, s_{h+j+k})$ on sets of $k+2$ consecutive knots are a basis for the full space of kth-degree splines on the full set of knots. If $S(\cdot)$ is one of these splines and $[s_j, s_{j+1})$ is an interval between consecutive knots, then the values of $S(x)$ in the interval are affected by the $k+1$ B-splines originating at the knots $s_{j-k}, \dots, s_j$ and terminating at the knots $s_{j+1}, \dots, s_{j+k+1}$. It follows that if we start by specifying a sequence of knots $s_0 < \cdots < s_m$ and we want to fit a spline for values of x in the interval $[s_0, s_m)$, then we must also use k extra knots $s_{-k} < \cdots < s_{-1}$ to the left of s_0 and k extra knots $s_{m+1} < \cdots < s_{m+k}$ to the right of s_m to define the $m+k$ consecutive B-splines affecting $S(x)$ for x in the interval $[s_0, s_m)$. These $m+k$ B-splines originate at the knots $s_{-k}, \dots, s_{m-1}$ and terminate at the knots $s_1, \dots, s_{m+k}$, respectively. Any spline $S(\cdot)$ in the full space of kth-degree splines defined using the full set of knots is equal to a linear combination of these $m+k$ B-splines in the interval $[s_0, s_m]$ (in the case of positive-degree splines, which are continuous) or $[s_0, s_m)$ (in the case of zero-degree splines, which are only right-continuous). We will refer to this interval as the *completeness region* for splines that are linear combinations of these $m+k$ B-splines. These linear combinations are 0 for $x < s_{-k}$ and $x \geq s_{m+k}$ and "incomplete" in the outer regions (s_{-k}, s_0) and (s_m, s_{m+k}) in which the spline is "returning to 0".

By default, `bspline` and `frencurv` assume that the `knots()` option specified by the user is only intended to span the completeness region and that the specified knots correspond to s_0, ..., s_m. (`flexcurv` has no `knots()` option, because it defines its own "sensibly spaced" knots, which are then input to `frencurv`.) By default, `bspline` and `frencurv` generate k extra knots on the left, with spacing equal to the difference between the first two knots, and k extra knots on the right, with spacing equal to the difference between the last two knots. If the user specifies the option `noexknot`, then `bspline` assumes that the user has specified the full set of knots, corresponding to s_{-k}, ..., s_{m+k}, and so does not generate any new knots.

4.2 Reference splines

If we have calculated the $n \times q$ matrix $\mathbf{V}$ of B-splines as in (4), and we also have a set of q reference X values $r_1 < r_2 < \cdots < r_q$, then we might prefer to reparameterize the spline by its values at the r_j. To do this, we first calculate a $q \times q$ square matrix $\mathbf{W}$, defined such that

$$\mathbf{W}_{ij} = A(r_i; s_{h+j-1}, \dots, s_{h+j+k}) \qquad (5)$$

the value of the jth B-spline at the ith reference point. If β is the column vector of regression coefficients with respect to the B-splines in $\mathbf{V}$, and γ is the column vector of values of the spline at the reference points, then it follows that

$$\gamma = \mathbf{W}\beta$$

If $\mathbf{W}$ is invertible, then the vector of values of the fitted spline at the data points is

$$\mathbf{V}\beta = \mathbf{V}\mathbf{W}^{-1}\mathbf{W}\beta = \mathbf{V}\mathbf{W}^{-1}\gamma = \mathbf{Z}\gamma \tag{6}$$

where $\mathbf{Z} = \mathbf{V}\mathbf{W}^{-1}$ is a transformed design matrix whose columns contain values of a set of reference splines for the estimation of the reference-point spline values γ.

Note that the argument of (6) will still apply whether $\mathbf{V}$ and $\mathbf{Z}$ are matrices of discrete column vectors or matrices of continuous functions on the real line. In addition, the argument still applies if $\mathbf{V}$ is a spline basis other than a B-spline basis, for example, a restricted "natural" spline basis of the kind discussed by Royston and Sauerbrei (2007).

The choice of reference points and knots is open to the user and constrained mainly by the requirement that the matrix $\mathbf{W}$ is invertible. This implies that each of the q B-splines must be positive for at least one of the q reference values and that each reference value must have at least one positive B-spline value. With the aim of satisfying this requirement, `frencurv` and `flexcurv` both start with a list of reference points and (at least by default) choose the knots accordingly.

4.3 Knot choice by frencurv

In the default method used by `frencurv` (if the user provides no `knots()` option), we try to create a one-to-one correspondence between the reference points and the B-splines, with the feature that each reference point is in the middle of the nonzero range of its corresponding B-spline. This is done by ensuring that each reference point is equal to the central knot of its B-spline in the case of odd-degree splines (such as linear, cubic, or quintic splines) and in between the two central knots of its B-spline in the case of even-degree splines (such as step, quadratic, or quartic splines). This choice means that for a spline of degree k, there will be $\texttt{int}(k/2)$ reference points outside the spline's completeness region on the left and another $\texttt{int}(k/2)$ reference points outside the spline's completeness region on the right, where $\texttt{int}(\cdot)$ is the truncation (or "integer-part") function. The parameters corresponding to these "extra" reference points will not be easy to explain to nonmathematicians: they describe the behavior of the spline as it returns to 0 outside its completeness region. However, for a quadratic or cubic spline, there is only one such external reference Y value at each end of the completeness region.

By default, `frencurv` starts with the reference points originally provided and chooses knots "appropriately". For an odd-degree spline, the knots are initialized to the original reference points themselves. For an even-degree spline, the knots are initialized to mid-points corresponding to the original reference points as follows. If the original reference points are $r_1 < \cdots < r_m$, then the original knots $s_0 < \cdots < s_m$ are initialized to

$$s_j = \begin{cases} r_1 - (r_2 - r_1)/2, & \text{if} \quad j = 0 \\ (r_j + r_{j+1})/2, & \text{if} \quad 1 \le j \le m-1 \\ r_m + (r_m - r_{m-1})/2, & \text{if} \quad j = m \end{cases}$$

`frencurv` assumes by default that the reference points initially provided are all in the completeness region and adds $\texttt{int}(k/2)$ extra reference points to the left, spaced by the difference between the first two original reference points, and adds `int`($k/2$) extra reference points to the right, spaced by the difference between the last two original reference points, where k is specified by the `power()` option. If `noexref` is specified, then the original `refpts()` list is assumed to be the complete list of reference points, and it is the user's responsibility to choose sensible ones. In either case, the original knots are extended on the left and right as described above unless `noexknot` is specified.

4.4 Knot choice by flexcurv

`flexcurv` uses an alternative method to define knots from reference points that guarantees that the reference points, the values of the X variable specified by `xvar()`, and (optionally) a list of other X values specified by the `include()` option will be in the completeness region of the generated spline basis. It also guarantees that the knots will be "sensibly" spaced, using a definition of sensibility specified by the `krule()` option.

Suppose that the user provides q reference points $r_1, \ldots, r_q$ in the `refpts()` option. `flexcurv` first calculates the numbers $x_{\rm inf}$ and $x_{\rm sup}$ as the minimum and maximum, respectively, of all values present in the `xvar()` variable, the `refpts()` list, or the `include()` list. The numbers $x_{\rm inf}$ and $x_{\rm sup}$ will be the infimum and the supremum, respectively, of the completeness region of the spline basis. The number of intervals between adjacent knots in and bordering the completeness region is then $m = q - k$. The original knots in and bordering the completeness region are $s_0, \ldots, s_m$.

If the user specifies `krule(regular)` (the default), then these s_j are spaced regularly and defined by the simple formula

$$s_j = \frac{j}{m} x_{\rm sup} + \frac{m-j}{m} x_{\rm inf}$$

If the user specifies `krule(interpolate)`, then these s_j are interpolated between the reference points using a more complicated formula. If the spline power k is 0, we define $s_0 = x_{\rm inf}$, $s_m = x_{\rm sup}$, and $s_j = r_{j+1}$ for other j. Otherwise, we first define, for each j from 0 to m,

$$\sigma(j) = 1 + j(q-1)/m, \quad \pi(j) = \mathrm{int}\left\{\sigma(j)\right\}, \quad \rho(j) = \sigma(j) - \pi(j)$$

We then define the s_j as

$$s_j = \begin{cases} x_{\rm inf}, & j = 0 \\ x_{\rm sup}, & j = m \\ \left\{1 - \rho(j)\right\} r_{\pi(j)} \; + \; \rho(j)\, r_{\pi(j)+1}, & \text{otherwise} \end{cases} \tag{7}$$

This formula ensures that the knots s_j are interpolated between the reference points in a way that will be regularly spaced if the reference points themselves are regularly spaced from $r_1 = x_{\rm inf}$ to $r_q = x_{\rm sup}$. However, if the reference points are not regularly spaced, then the user can specify `krule(interpolate)` to ensure that the reference splines will still be definable, which may not be the case if the user specifies `krule(regular)` with irregularly spaced reference points.

`flexcurv` then calls `frencurv` to generate the reference splines, with the reference points $r_1, \ldots, r_q$ as the `refpts()` option, with the knots $s_0, \ldots, s_m$ as the `knots()` option, with the `noexref` option, but without the `noexknot` option. This implies that whichever `krule()` option is specified, any extra knots to the left of the completeness region will be regularly spaced by the distance between the first two internal knots, and any extra knots to the right of the completeness region will be regularly spaced by the distance between the last two internal knots. `krule(regular)` specifies that the knots inside and outside the completeness region are regularly spaced; thus any pair of adjacent knots inside or outside the completeness region is separated by $(x_{\text{sup}} - x_{\text{inf}})/m$ x-axis units. Both `krule()` options result in the generation of a basis of q reference splines corresponding to the respective reference points, with a completeness region $x_{\text{inf}} \leq x \leq x_{\text{sup}}$ (for positive-degree splines) or $x_{\text{inf}} \leq x < x_{\text{sup}}$ (for zero-degree splines). In the case of zero-degree splines, the user must specify x_{sup} in the `include()` option as a number strictly greater than any reference points and `xvar()` values: x_{sup} is outside the completeness region for a zero-degree spline, which is a right-continuous step function with discontinuities at its knots, which include x_{sup}.

4.5 The omit() and base() options

From the definition of a reference spline basis as a basis of its corresponding spline space, it follows that each reference spline is equal to 1 at its own reference point and equal to 0 at all other reference points. In more formal language, if we consider the matrix $\mathbf{Z}$ of reference splines in (6) and suppose that for some reference point r_h and some i from 1 to n, $X_i = r_h$, then it follows that for each jth column of $\mathbf{Z}$,

$$\mathbf{Z}_{ij} = \begin{cases} 1, & j = h \\ 0, & j \neq h \end{cases} \tag{8}$$

(This follows because column h of $\mathbf{Z}$ is in the spline space spanned by $\mathbf{Z}$, with the hth coordinate 1 and all other coordinates 0; because the sum of columns h and j is in the same spline space, with the hth and jth coordinates 1 and all other coordinates 0; and because both of these splines are 1 where $X_i = r_h$. Graphic examples of reference splines of degrees 0 to 3 that illustrate this property are given in Newson [2011].)

Because the unit function is itself a spline (of any degree), it follows that its coordinates in the reference splines must all be 1, implying that a basis of reference splines must sum to 1, at least in the completeness region of their spline space.

A consequence of these properties is that if we start with a basis of reference splines, exclude a reference spline corresponding to a base reference point r_b, and include the unit function, then the resulting set of splines is an alternative basis of the same spline space. Any spline $S(\cdot)$ in that spline space will have coordinates in this alternative basis. The coordinate of $S(\cdot)$ in the unit function will be equal to $S(r_b)$, whereas the coordinate of $S(\cdot)$ in any of the surviving reference splines corresponding to another reference point r_j will be equal to $S(r_j) - S(r_b)$.

It follows that we can replace a baseline column b of $\mathbf{Z}$ with a unit vector to derive an alternative design matrix $\mathbf{Z}^{[b]}$. This alternative design matrix can be defined formally as

$$\mathbf{Z}_{ij}^{[b]} = \begin{cases} 1, & j = b \\ \mathbf{Z}_{ij}, & \text{otherwise} \end{cases} \tag{9}$$

If this design matrix is used by an estimation command, then the parameter corresponding to the unit vector will be the intercept parameter `_cons`, equal to the value of the spline at the base reference point r_b; the other parameters will be differences between the value of the spline at the reference point r_h and the value of the spline at the base reference point r_b, for $h \neq b$. Therefore, reference splines play the same role for continuous "X factors" that indicator (or "dummy") variables play for discrete factors. (These indicator variables are generated by the `xi:` prefix in Stata 10 and, in virtual form, by factor *varlist*s in Stata 11 or higher. They are really reference splines of degree 0, with integer reference points and knots.)

To perform the substitution (9), `flexcurv` and `frencurv` have an option `omit()` for users of Stata 10, which causes the base reference spline to be dropped, and an option `base()` for users of Stata 11 or higher, which causes the base reference spline to be set to 0. In either case, the reference splines can be included in the design matrix of an estimation command. In this case, we do not use the `noconstant` option, because we want to add the unit vector to the design matrix.

4.6 Multivariate splines and interactions

Reference splines are a generalization to continuous factors of indicator functions for discrete factors. This generalization extends to multifactor models, whose parameters frequently include conditional means for combinations of discrete factor levels or even "interactions", defined informally as "differences between differences". (More formally, interactions are defined recursively; thus an interaction of order 0 is a difference, and an interaction of order $k+1$ is a difference between interactions of order k.)

Multifactor models frequently use product bases, derived from two or more input bases of indicator functions and then included in a design matrix. The product bases are created by a matrix operator, which we will call the factor-product operator. Given an $n \times q$ matrix $\mathbf{F}$ and an $n \times p$ matrix $\mathbf{G}$, this operator $\bigotimes$ is defined as

$$\mathbf{F} \bigotimes \mathbf{G} = \bigoplus_{j=1}^{q} (\mathbf{F}_{*j} :* \mathbf{G}) \tag{10}$$

where $\bigoplus$ is the multifold version of the horizontal matrix concatenation operator represented by the comma operator in Mata (see [M-2] **op_join**), `:*` is the elementwise product operator represented by `:*` in Mata (see [M-2] **op_colon**), and $\mathbf{F}_{*j}$ represents the jth column of $\mathbf{F}$. The factor-product operator $\bigotimes$ corresponds to the `*` operator in `xi:` interaction *varlists* or to the `#` operator in factor *varlists* in Stata 11 or higher. It can also be implemented for a pair of Stata input variable lists by using the `prodvars` package, downloadable from SSC, which generates the output matrix as a list of new variables.

The factor-product operator is traditionally applied to matrices of factor-level identifier variables, but it may equally be applied in the same way to matrices of reference splines. To see this, we will replace $\mathbf{F}$ in (10) with the matrices $\mathbf{Z}$ of (6) and $\mathbf{Z}^{[b]}$ of (9) and suppose that $\mathbf{G}$ is an arbitrary design submatrix of arbitrary covariates, which may or may not include a unit vector.

We first consider the case $\mathbf{F} = \mathbf{Z}$ and its factor-product $\mathbf{Z} \bigotimes \mathbf{G}$. We imagine that this factor-product matrix is applied to a column vector of parameters:

$$\zeta = \bigodot_{j=1}^{q} \zeta^{(j)}$$

where $\bigodot$ is the multifold version of the vertical matrix concatenation operator represented by \ in Mata (see [M-2] **op_join**) and each $\zeta^{(j)}$ is a column vector of p parameters corresponding to the columns of $\mathbf{G}$. For a reference point r_h, if the ith X value is $X_i = r_h$, then it follows from (8) that for each j,

$$(\mathbf{Z}_{*j} \mathbin{:*} G)_{ij} = \begin{cases} \mathbf{G}_{ij}, & j = h \\ 0, & j \neq h \end{cases} \tag{11}$$

It follows that the column vector $\zeta^{(h)}$ is a vector of parameters corresponding to the covariates in $\mathbf{G}$ for a special model in force when the X variate is equal to the reference point r_h. Therefore, the full parameter vector ζ is a combined vector of parameters for a composite model derived from q submodels, each corresponding to X values equal to one of the q reference points. The composite model predicts interpolated values at nonreference X values, and $\mathbf{Z} \bigotimes \mathbf{G}$ is its design matrix.

We now consider the case $\mathbf{F} = \mathbf{Z}^{[b]}$ and its factor-product $\mathbf{Z}^{[b]} \bigotimes \mathbf{G}$. We assume that the corresponding parameter vector is $\xi = \bigodot_{j=1}^{q} \xi^{(j)}$, where each $\xi^{(j)}$ is a column vector of p parameters. This time, one reference point r_b is the base reference point, and (9) implies that the corresponding submatrix $\mathbf{Z}_{*b} \mathbin{:*} \mathbf{G}$ is a copy of $\mathbf{G}$. The other jth submatrices conform to (11) for rows i, in which the X value X_i is equal to a reference point r_h. It follows that for any such row i, we have the identity

$$\left\{ \left(\mathbf{Z}^{[b]} \bigotimes \mathbf{G} \right) \xi \right\}_i = \begin{cases} \left(\mathbf{G}\xi^{(b)} \right)_i, & h = b \\ \left(\mathbf{G}\xi^{(b)} \right)_i + \left(\mathbf{G}\xi^{(h)} \right)_i, & h \neq b \end{cases}$$

In other words, the parameters $\xi^{(b)}$ belong to a submodel with design matrix $\mathbf{G}$ for rows i where $X_i = r_b$. The parameters $\xi^{(h)}$, where $h \neq b$, are differences between the parameters of a submodel with the design matrix $\mathbf{G}$ for rows i where $X_i = r_h$ and the corresponding parameters of the submodel with the same design matrix $\mathbf{G}$ for rows i where $X_i = r_b$.

Because ζ and ξ are alternative parameterizations of the same supermodel, it follows (at least if the factor-product columns are linearly independent) that for each j from 1 to q, the parameters conform to the relation

$$\xi^{(j)} = \begin{cases} \zeta^{(b)}, & j = b \\ \zeta^{(j)} - \zeta^{(b)}, & j \neq b \end{cases}$$

So if the ζ parameters are means, then the corresponding ξ parameters are differences. And if the ζ parameters are differences, then the corresponding ξ parameters are "interactions".

We see that reference-spline bases (whether or not they are modified to include a unit vector) can be combined nonadditively (or "interactively") to form factor-product bases in the same way that identifier variable bases can be combined. Note that the matrix $\mathbf{G}$ may also contain reference splines in variables other than X, allowing the possibility of nonadditive multivariate splines. Alternatively, reference splines may be combined with other covariates in an additive (or "noninteractive") way.

5 Examples

These examples demonstrate the easy-to-use `flexcurv` command and are distributed in the file `example1.do`, which is part of the online material for this article. The more comprehensive `bspline` and `frencurv` commands are tools for special occasions, especially when the user has a reason for choosing a certain set of knots. Examples for these commands appear in the online help and in the manual `bspline.pdf`, distributed with the package as an ancillary file.

5.1 The cubic spline of figure 2

Let's first look at the cubic spline illustrated in the lower right subgraph of figure 2. After loading `auto.dta`, we generate the spline basis as follows:

```
. sysuse auto
(1978 Automobile Data)
. flexcurv, xvar(weight) power(3) refpts(1500(900)5100) generate(cs_)
. describe cs_*

              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
cs_1            float  %8.4f                  Spline at 1,500
cs_2            float  %8.4f                  Spline at 2,400
cs_3            float  %8.4f                  Spline at 3,300
cs_4            float  %8.4f                  Spline at 4,200
cs_5            float  %8.4f                  Spline at 5,100
```

We see that the five cubic reference splines `cs_1` to `cs_5` have variable labels that inform the user of the reference point to which each reference spline corresponds. We then fit the regression model as follows, using the `noconstant` option:

```
. regress mpg cs_*, noconstant noheader
```

mpg	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
cs_1	33.86387	3.733922	9.07	0.000	26.4149	41.31284
cs_2	24.6141	.7811342	31.51	0.000	23.05578	26.17242
cs_3	18.79659	.6841035	27.48	0.000	17.43184	20.16134
cs_4	15.47252	1.113113	13.90	0.000	13.25191	17.69312
cs_5	10.05772	5.322653	1.89	0.063	-.5606797	20.67613

The parameters corresponding to the reference splines are the values of the spline at the corresponding reference points.

Alternatively, we could fit the same model with a different parameterization, with an intercept equal to the mileage expected at the central reference point of 3,300 U.S. pounds and with effects on mileage of weights equal to the other reference points. This is done using the `base()` option to generate a slightly different spline basis and then using `regress` without the `noconstant` option:

```
. flexcurv, xvar(weight) power(3) refpts(1500(900)5100) base(3300)
> generate(bcs_)

. describe bcs_*

              storage  display     value
variable name   type   format      label      variable label

bcs_1           float  %8.4f                  Spline at 1,500
bcs_2           float  %8.4f                  Spline at 2,400
bcs_3           byte   %8.4f                  Spline at 3,300
bcs_4           float  %8.4f                  Spline at 4,200
bcs_5           float  %8.4f                  Spline at 5,100

. regress mpg bcs_*, noheader
note: bcs_3 omitted because of collinearity
```

mpg	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
bcs_1	15.06729	3.577033	4.21	0.000	7.931301	22.20327
bcs_2	5.817516	1.078029	5.40	0.000	3.666908	7.968124
bcs_3	0	(omitted)				
bcs_4	-3.324069	1.438353	-2.31	0.024	-6.193505	-.4546328
bcs_5	-8.738858	5.192156	-1.68	0.097	-19.09693	1.61921
_cons	18.79659	.684103	27.48	0.000	17.43184	20.16134

The spline `bcs_3` has storage type `byte` because it corresponds to the base reference weight of 3,300 U.S. pounds. It has therefore been set to 0 and `compress`ed. `regress` then omits the corresponding parameter because of collinearity, leaving an intercept (a mileage) and the effects of the other reference weights (mileage differences).

5.2 Polynomials as splines

By the definition of a spline, a polynomial limited to a bounded interval is a special case of a spline, with knots at the boundaries. And all polynomials fit to real-world data by real-world scientists are restricted to bounded intervals.

It is well known that a degree-k polynomial can be specified by $k+1$ bivariate points on the curve, each containing a reference point on the X axis and its corresponding Y value. `flexcurv` can implement this specification method, with the possibility of confidence intervals for the reference Y values. These reference Y values are easier to explain to nonmathematical colleagues than the usual parameters for a polynomial model.

In `auto.dta`, we might use `flexcurv` to regress `mpg` with respect to `weight`, using a quadratic model, as follows:

```
. flexcurv, xvar(weight) power(2) refpts(2000 3000 4000) generate(qs_)
. describe qs_*

              storage  display     value
variable name   type   format      label      variable label
--------------------------------------------------------------------------------
qs_1            float  %8.4f                  Spline at 2,000
qs_2            float  %8.4f                  Spline at 3,000
qs_3            float  %8.4f                  Spline at 4,000
. regress mpg qs_*, noconstant noheader
------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        qs_1 |   28.16455   .7356117    38.29   0.000     26.69779    29.63132
        qs_2 |   20.62851   .5388504    38.28   0.000     19.55407    21.70295
        qs_3 |   15.74126   .6508289    24.19   0.000     14.44354    17.03897
------------------------------------------------------------------------------
```

We start by using `flexcurv` to generate a basis of three quadratic reference splines in `weight` at reference points 2,000, 3,000, and 4,000 U.S. pounds and then `describe` them. Again, each reference spline has a variable label in case the user forgets its reference point. Then we use `regress` with the `noconstant` option to estimate the values (in miles per gallon) of the quadratic polynomial at these reference points. These parameters are easier to understand than the ones provided if we fit the same quadratic model using the command `regress mpg c.weight c.weight#c.weight, nohead` (not shown). The fitted and observed values, as well as the estimates and confidence limits for the parameters, are plotted in figure 3, which was produced using the SSC packages `parmest` and `eclplot` (Newson 2003).

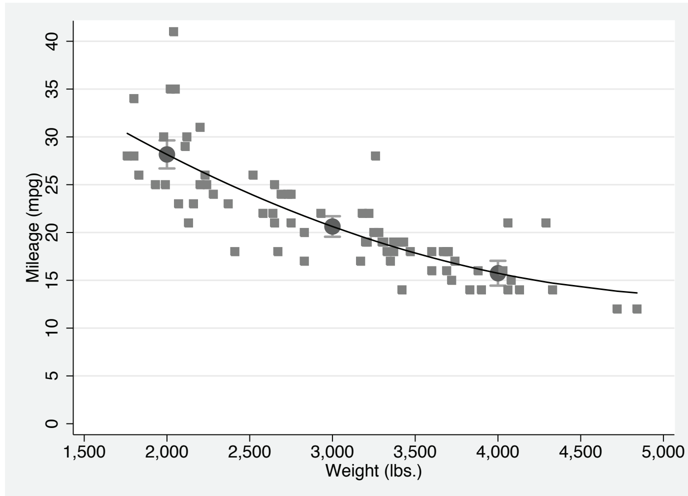

Figure 3. Quadratic regression of `mpg` with respect to `weight`

We can also fit the same model with a third parameterization, namely, the base level of `mpg` for cars weighing 2,000 pounds and the effects on `mpg` of increasing the weight to 3,000 and 4,000 pounds, respectively:

```
. flexcurv, xvar(weight) power(2) refpts(2000 3000 4000) base(2000)
> generate(bqs_)

. describe bqs_*

              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
bqs_1           byte   %8.4f                  Spline at 2,000
bqs_2           float  %8.4f                  Spline at 3,000
bqs_3           float  %8.4f                  Spline at 4,000

. regress mpg bqs_*, noheader
note: bqs_1 omitted because of collinearity
------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       bqs_1 |          0  (omitted)
       bqs_2 |  -7.536052   .8812637    -8.55   0.000    -9.293242   -5.778862
       bqs_3 |   -12.4233   1.029623   -12.07   0.000    -14.47631   -10.37029
       _cons |   28.16456   .7356117    38.29   0.000     26.69779    29.63133
------------------------------------------------------------------------------
```

This time, the spline `bqs_1` at 2,000 pounds has storage type `byte` because it represents the `base()` option. It has therefore been set to 0 and `compress`ed. The `regress` command is called without the `noconstant` option and outputs a parameter `_cons`, equal to the base mileage of 28.16 miles per gallon expected for 2,000-pound cars; an omitted parameter for `bqs_1` representing the zero-effect of this base mileage (with 0 confidence limits); and the two negative effects on mileage of increasing the weight to 3,000 and 4,000 pounds.

Of course, we can add other terms to this model to represent the additive (or "non-interactive") effects of other covariates or factors, such as the binary variable `foreign`, indicating non-U.S. origin:

```
. regress mpg foreign bqs_*, nohead
note: bqs_1 omitted because of collinearity
```

mpg	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
foreign	-2.2035	1.059246	-2.08	0.041	-4.316101	-.0908999
bqs_1	0	(omitted)				
bqs_2	-8.617167	1.005957	-8.57	0.000	-10.62349	-6.610849
bqs_3	-14.05203	1.275017	-11.02	0.000	-16.59497	-11.50909
_cons	29.75756	1.050386	28.33	0.000	27.66263	31.85249

The parameter for `foreign` is negative and tells the familiar `auto.dta` story that non-U.S. cars travel fewer miles per gallon (on average) than U.S. cars of the same weight, although U.S. cars are usually heavier than non-U.S. cars.

5.3 Linear splines with unevenly spaced reference points

We might fit a linear spline to the same data, with the reference points unevenly spaced. If splines are linear or reference points are unevenly spaced, then it is a good idea to use the option `krule(interpolate)` for two reasons. First, if the spline is linear, then (7) ensures that each reference point will also be a knot, with the possible exceptions of the first and last reference points if the completeness region extends beyond these. Second, if the reference points are unevenly spaced, then (7) ensures that the reference splines will exist because the matrix $\mathbf{W}$ of (5) will have no 0 rows or columns, which it might have (and therefore be singular) if we use the default `krule(regular)`.

We might fit a linear spline as follows:

```
. flexcurv, xvar(weight) power(1) krule(interpolate)
> refpts(1500 2000 2500 3000 4000 5000) generate(ls_)
. describe ls_*
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
ls_1            float  %8.4f                  Spline at 1,500
ls_2            float  %8.4f                  Spline at 2,000
ls_3            float  %8.4f                  Spline at 2,500
ls_4            float  %8.4f                  Spline at 3,000
ls_5            float  %8.4f                  Spline at 4,000
ls_6            float  %8.4f                  Spline at 5,000
. regress mpg ls_*, noconstant noheader
------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        ls_1 |   26.34741   4.410006     5.97   0.000     17.54738    35.14744
        ls_2 |   30.16913   1.149293    26.25   0.000     27.87575    32.46251
        ls_3 |   21.69784    1.32861    16.33   0.000     19.04664    24.34904
        ls_4 |    20.9661   1.096847    19.11   0.000     18.77738    23.15483
        ls_5 |   15.56144   1.071791    14.52   0.000     13.42271    17.70016
        ls_6 |   12.45729   2.860836     4.35   0.000     6.748579      18.166
------------------------------------------------------------------------------
```

The fitted and observed values for this model and confidence intervals for the parameters are displayed in figure 4. The first and last reference points are below the minimum and above the maximum car weight, respectively, so the reference points are also the knots, and the fitted values are interpolated linearly between them.

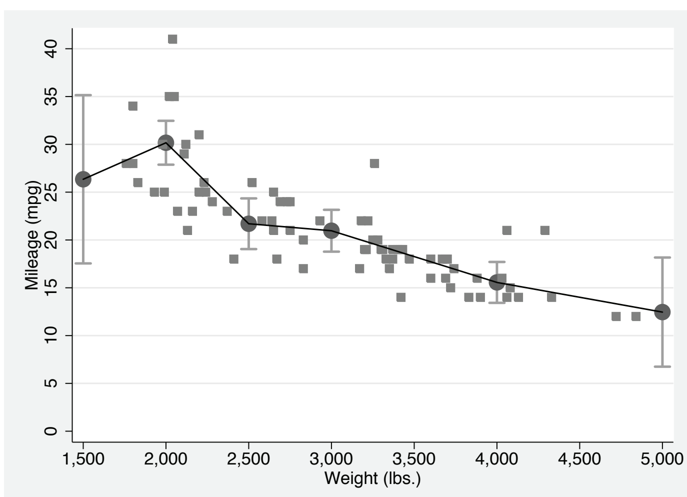

Figure 4. Linear spline regression of `mpg` with respect to `weight`

We might reparameterize the same model to measure differences between the spline at each reference point and the spline at the base reference point, which we will set to the "midrange" value of 3,000 pounds:

```
. flexcurv, xvar(weight) power(1) krule(interpolate)
> refpts(1500 2000 2500 3000 4000 5000) base(3000) generate(bls_)

. describe bls_*

              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------
bls_1           float  %8.4f                  Spline at 1,500
bls_2           float  %8.4f                  Spline at 2,000
bls_3           float  %8.4f                  Spline at 2,500
bls_4           byte   %8.4f                  Spline at 3,000
bls_5           float  %8.4f                  Spline at 4,000
bls_6           float  %8.4f                  Spline at 5,000

. regress mpg bls_*, noheader
note: bls_4 omitted because of collinearity
------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       bls_1 |   5.381306   4.590998     1.17   0.245    -3.779888     14.5425
       bls_2 |   9.203024   1.505075     6.11   0.000     6.199692    12.20636
       bls_3 |   .7317385   1.968027     0.37   0.711    -3.195399    4.658876
       bls_4 |          0  (omitted)
       bls_5 |  -5.404668   1.872296    -2.89   0.005    -9.140777    -1.66856
       bls_6 |  -8.508816   2.918556    -2.92   0.005     -14.3327   -2.684928
       _cons |    20.9661   1.096847    19.11   0.000     18.77738    23.15483
------------------------------------------------------------------------------
```

There are alternative parameterizations of linear splines that also produce sensible parameters. The `mkspline` package of official Stata generates a basis of linear splines whose corresponding parameters are either the local slopes in the intervals between knots or the differences between pairs of these local slopes in consecutive intervals between knots. See `mkspline` (see [R] **mkspline**) for the practical details of this method.

5.4 Multifactor cubic splines

We might also fit a multifactor model. If we add the binary factor variable `odd`—created by typing `generate odd=mod(_n,2)`, and equal to 1 for odd-numbered cars and to 0 for even-numbered cars—then we might want to measure separate effects of car weight on car mileage in odd-numbered and even-numbered cars by using a two-factor model, with `weight` as a continuous factor and `odd` as a discrete factor. To do this, we use factor-product bases, as we would if we had two discrete factors.

Two useful packages for this purpose are `prodvars` and `fvprevar`, both downloadable from SSC. The `prodvars` package inputs two *varlist*s, which function as the columns of $\mathbf{F}$ and $\mathbf{G}$, respectively, in (10), and outputs the factor-product matrix in a generated *newvarlist*, with names or labels or characteristics generated by user-specified rules. The `fvprevar` package is an alternative version of the `fvrevar` command (see [R] **fvrevar**) of official Stata and functions as an updated version of the `xi:` prefix (see [R] **xi**) of

Stata 10. Like `fvrevar`, `fvprevar` inputs a factor *varlist*. However, unlike `fvrevar`, it generates an output list of permanent variables instead of an output list of temporary variables. These permanent output variables can then be input to `prodvars` with a list of reference splines to generate a product-variable list of "interaction" reference splines.

In our case, we might start by using `flexcurv` to generate a list of cubic reference splines `a_*`, whose corresponding parameters might be differences in mileage between cars with a nonbase reference weight and cars with a base reference weight of 1,760 U.S. pounds:

```
. flexcurv, xvar(weight) power(3) refpts(1760(616)4840) base(1760)
> generate(a_) labprefix(weight==) labfmt(%9.0g)
. describe a_*
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
a_1             byte   %8.4f                  weight==1760
a_2             float  %8.4f                  weight==2376
a_3             float  %8.4f                  weight==2992
a_4             float  %8.4f                  weight==3608
a_5             float  %8.4f                  weight==4224
a_6             float  %8.4f                  weight==4840
```

(Note the use of the `labprefix()` option to specify a nonstandard prefix for the spline variable labels and of the `labfmt()` option to eliminate the commas from the reference-point values in these labels.) We then use `fvprevar` to generate a list of indicator (or "dummy") variables indicating even-numbered and odd-numbered cars:

```
. fvprevar ibn.odd, generate(b_)
. describe b_*
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
b_1             byte   %9.0g                  0bn.odd
b_2             byte   %9.0g                  1.odd
```

The generated output variables `b_*`, specified by the `generate()` option, have variable labels indicating the expanded factor *varlist* elements to which they correspond.

We can now use `prodvars` to input the two lists of variables `a_*` and `b_*`, which play the role of **F** and **G**, respectively, in (10), generating a list of output variables `c_*`, which contain the factor-product variables:

```
. prodvars a_*, rvarlist(b_*) generate(c_) lseparator(" & ")
. describe c_*

              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
c_1             byte   %10.0g                 weight==1760 & 0bn.odd
c_2             byte   %10.0g                 weight==1760 & 1.odd
c_3             double %10.0g                 weight==2376 & 0bn.odd
c_4             double %10.0g                 weight==2376 & 1.odd
c_5             double %10.0g                 weight==2992 & 0bn.odd
c_6             double %10.0g                 weight==2992 & 1.odd
c_7             double %10.0g                 weight==3608 & 0bn.odd
c_8             double %10.0g                 weight==3608 & 1.odd
c_9             double %10.0g                 weight==4224 & 0bn.odd
c_10            double %10.0g                 weight==4224 & 1.odd
c_11            double %10.0g                 weight==4840 & 0bn.odd
c_12            double %10.0g                 weight==4840 & 1.odd
```

We see that `prodvars` acts similarly to the `#` operator in factor *varlist*s in Stata 11 and above or to the `*` operator used in `xi:` *varlist*s. In a manner similar to `xi:`, we have used the option `lseparator(" & ")` to separate semi-informative variable labels for the output variables from the variable labels for the input variables.

We can now enter the variables `b_*` and `c_*` into an equal-variance regression model, this time with the `noconstant` option, because the two intercept terms `b_*` for even-numbered and odd-numbered cars provide the intercept parameters:

```
. regress mpg b_* c_*, noconstant noheader
note: c_1 omitted because of collinearity
note: c_2 omitted because of collinearity
------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         b_1 |   28.16762    2.45977    11.45   0.000     23.25061    33.08464
         b_2 |   32.52757   2.930847    11.10   0.000     26.66888    38.38625
         c_1 |          0  (omitted)
         c_2 |          0  (omitted)
         c_3 |  -3.003417    2.99441    -1.00   0.320    -8.989156    2.982323
         c_4 |   -7.31852   3.681023    -1.99   0.051    -14.67678    .0397399
         c_5 |  -6.786187   2.593622    -2.62   0.011    -11.97076    -1.60161
         c_6 |   -13.3264   2.794913    -4.77   0.000    -18.91335   -7.739447
         c_7 |  -11.25077   2.805387    -4.01   0.000    -16.85866   -5.642884
         c_8 |  -14.66254   3.240914    -4.52   0.000    -21.14103   -8.184041
         c_9 |    -15.833   4.438494    -3.57   0.001    -24.70542   -6.960573
        c_10 |  -16.29373   3.214685    -5.07   0.000    -22.71979   -9.867662
        c_11 |   -16.1599   4.192831    -3.85   0.000    -24.54125   -7.778546
        c_12 |   -21.5878   6.441925    -3.35   0.001    -34.46502   -8.710573
------------------------------------------------------------------------------
```

The parameters `c_1` and `c_2` are the omitted zero-effects on `mpg` of the baseline weight of 1,760 pounds, whereas the other `c_*` parameters are the negative effects on `mpg` of higher weights for even-numbered and odd-numbered cars listed primarily by ascending weight and secondarily by ascending oddness within each weight. Note that we could have had `ibn.odd` instead of `b_*` in the `regress` command, producing the same estimates for the same parameters.

6 Acknowledgments

I would like to thank Professor Patrick Royston of the MRC Clinical Trials Unit in London, UK, for redirecting my attention back to splines during a discussion at the 2009 UK Stata Users Group meeting and thereby prompting me (eventually) to add the latest improvements to `bspline`. I would also like to thank Buzz Burhans at Dairy-Tech Group in West Glover, VT, and my colleague Bernet S. Kato at Imperial College London for some very helpful comments on the draft; and an anonymous reviewer for suggesting section 2. My own work at Imperial College London is financed by the UK Department of Health.

7 References

Newson, R. 2000. sg151: B-splines and splines parameterized by their values at reference points on the x-axis. *Stata Technical Bulletin* 57: 20–27. Reprinted in *Stata Technical Bulletin Reprints*, vol. 10, pp. 221–230. College Station, TX: Stata Press.

———. 2003. Confidence intervals and p-values for delivery to the end user. *Stata Journal* 3: 245–269.

Newson, R., D. Strachan, E. Archibald, J. Emberlin, P. Hardaker, and C. Collier. 1997. Effect of thunderstorms and airborne grass pollen on the incidence of acute asthma in England, 1990–94. *Thorax* 52: 680–685.

Newson, R. B. 2011. Sensible parameters for polynomials and other splines. UK Stata Users Group meeting proceedings. http://ideas.repec.org/p/boc/usug11/01.html.

Orsini, N., and S. Greenland. 2011. A procedure to tabulate and plot results after flexible modeling of a quantitative covariate. *Stata Journal* 11: 1–29.

Royston, P., and W. Sauerbrei. 2007. Multivariable modeling with cubic regression splines: A principled approach. *Stata Journal* 7: 45–70.

Ziegler, Z. 1969. One-sided L_1-approximation by splines of an arbitrary degree. In *Approximations with Special Emphasis on Spline Functions*, ed. I. J. Schoenberg, 405–413. New York: Academic Press.

About the author

Roger B. Newson is a lecturer in medical statistics at Imperial College London, UK, working principally in asthma research. He wrote the SSC packages `bspline`, `parmest`, `eclplot`, `prodvars`, and `fvprevar`.