The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–37

Binscatter Regressions

Matias D. Cattaneo
University of Michigan
Ann Arbor, MI
cattaneo@umich.edu

Richard K. Crump
Federal Reserve Bank of New York
New York, NY
richard.crump@ny.frb.org

Max H. Farrell
University of Chicago
Chicago, IL
max.farrell@chicagobooth.edu

Yingjie Feng
University of Michigan
Ann Arbor, MI
yjfeng@umich.edu

Abstract. We introduce the `Stata` (and `R`) package Binsreg, which implements the binscatter methods developed in Cattaneo, Crump, Farrell, and Feng (2019). The package includes the commands `binsreg`, `binsregtest`, and `binsregselect`. The first command (`binsreg`) implements binscatter for the regression function and its derivatives, offering several point estimation, confidence intervals and confidence bands procedures, with particular focus on constructing binned scatter plots. The second command (`binsregtest`) implements hypothesis testing procedures for parametric specification and for nonparametric shape restrictions of the unknown regression function. Finally, the third command (`binsregselect`) implements data-driven number of bins selectors for binscatter implementation using either quantile-spaced or evenly-spaced binning/partitioning. All the commands allow for covariate adjustment, smoothness restrictions, weighting and clustering, among other features. A companion `R` package with the same capabilities is also available.

Keywords: st0001, binscatter, binned scatter plot, nonparametrics, semiparametrics, partitioning estimators, B-splines, tuning parameter selection, confidence bands, shape and specification testing.

March 13, 2019

The views expressed in this paper are those of the authors and do not necessarily reflect the position of the Federal Reserve Bank of New York or the Federal Reserve System.

1 Introduction

Binscatter has become a very popular methodology in applied microeconomics since its introduction (Chetty and Szeidl 2006; Chetty, Looney, and Kroft 2009; Chetty, Friedman, Olsen, and Pistaferri 2011b; Chetty, Friedman, Hilger, Saez, Schanzenbach, and Yagan 2011a). See Stepner (2014) for a popular `Stata` implementation of canonical binscatter, and Starr and Goldfarb (2018) for a very recent heuristic overview of these methods. Binscatter techniques offer flexible, yet parsimonious ways of visualizing and summarizing "big data" in regression settings. They can also be used for formal estimation and inference, including testing of substantive hypothesis such as linearity or monotonicity, of the regression function and its derivatives. Despite its popularity among empirical researchers, little was known about the statistical properties of binscatter until very recently: Cattaneo, Crump, Farrell, and Feng (2019) offered the first foundational, thorough analysis of binscatter, giving an array of theoretical and practical results that aid both in understanding current practices (i.e., their validity or lack thereof) and in offering theory-based guidance for future applications.

This paper introduces the `Stata` (and `R`) package Binsreg, which includes three commands implementing the main methodological results in Cattaneo, Crump, Farrell, and Feng (2019): `binsreg`, `binsregtest`, and `binsregselect`. The first command (`binsreg`) implements binscatter for the regression function and its derivatives, offering several point estimation, confidence intervals and confidence bands procedures, with particular focus on constructing binned scatter plots. The second command (`binsregtest`) implements hypothesis testing procedures for parametric specification and for nonparametric shape restrictions of the unknown regression function and its derivatives. Finally, the third command (`binsregselect`) implements data-driven number of bins selectors for binscatter implementation using either quantile-spaced or evenly-spaced binning.

There exists another very popular `Stata` command also implementing binscatter methods: `binscatter`. See Stepner (2014) for an introduction to this alternative command. The command `binsreg` offers several new capabilities relative to `binscatter`, in addition to also implementing covariate adjustment in a different, more principled way. First, the command `binsreg` implements binscatter methods allowing for within-bin higher-order polynomial fitting and for across-bins smoothness restrictions, which enables derivative estimation and also produces smooth approximations of the regression function and its derivatives. Further, the command `binsreg` implements valid confidence intervals and confidence bands. None of these features are available in the command `binscatter`. Second, both commands allow for covariate adjustment, but each command does it in very different way: `binscatter` employs a residualization approach, while `binsreg` employs a semi-linear approach. As shown in Cattaneo, Crump, Farrell, and Feng (2019), the residualization approach is, in general, inconsistent for the target function of interest, while the semi-linear approach is either consistent (under correct specification) or has a clear probability limit interpretation (under misspecification). Finally, the other two commands in the package Binsreg, `binsregtest` and `binsregselect`, are novel implementations in `Stata` (and in `R`).

The rest of the article is organized as follows. Section 2 gives an overview of the

main methods available in the package Binsreg and discusses some implementation details. Sections 3, 4 and 5 discuss, respectively, the syntax of the commands `binsreg`, `binsregtest` and `binsregselect`. Section 6 gives a numerical illustration, while Section 7 concludes. The latest version of this software, as well other related software and materials, can be found at:

https://sites.google.com/site/nppackages/binsreg/

2 Overview of Methods and Implementation Details

This section summarizes the main methods implemented in the package Binsreg. For further methodological and theoretical details see Cattaneo, Crump, Farrell, and Feng (2019, CCFF hereafter) and references therein.

Given a random sample $\{(y_i, x_i, \mathbf{w}_i') : i = 1, 2, \ldots, n\}$, binscatter seeks to flexibly approximate the function $\mu(x)$ in the partially linear regression model:

$$y_i = \mu(x_i) + \mathbf{w}_i'\boldsymbol{\gamma} + \epsilon_i, \qquad \mathbb{E}[\epsilon_i | x_i, \mathbf{w}_i] = 0, \tag{1}$$

where y_i is an outcome of interest and x_i is a covariate of interest, while $\mathbf{w}_i$ represents other covariates possibly entering the semi-linear regression model. In particular, if the latter covariates are not present, then $\mathbb{E}[y_i|x_i] = \mu(x_i)$, while otherwise $\mu(x)$ denotes the average (partial) relationship between y_i and x_i after controlling for $\mathbf{w}_i$. More general interpretations are also possible; see CCFF for more discussion.

In some applications, interest may be on the v-th derivative of $\mu(x)$, which we denote by $\mu^{(v)}(x) = d^v\mu(x)/dx^v$. We employ the usual notation $\mu(x) = \mu^{(0)}(x)$.

2.1 Binscatter Construction

To approximate $\mu(x)$ and its derivatives in model (1), binscatter first partitions the support of x_i into J quantile-spaced bins, leading to the partitioning scheme:

$$\widehat{\Delta} = \{\widehat{\mathcal{B}}_1, \ldots, \widehat{\mathcal{B}}_J\}, \qquad \widehat{\mathcal{B}}_j = \begin{cases} \big[x_{(1)}, x_{(\lfloor n/J \rfloor)}\big) & \text{if } j = 1 \\ \big[x_{(\lfloor n(j-1)/J \rfloor)}, x_{(\lfloor nj/J \rfloor)}\big) & \text{if } j = 2, \ldots, J-1 \\ \big[x_{(\lfloor n(J-1)/J \rfloor)}, x_{(n)}\big] & \text{if } j = J \end{cases},$$

where $x_{(i)}$ denotes the i-th order statistic of the sample $\{x_1, x_2, \ldots, x_n\}$, $\lfloor \cdot \rfloor$ is the floor operator, and $J < n$. Each estimated bin $\widehat{\mathcal{B}}_j$ contains roughly the same number of observations $N_j = \sum_{i=1}^n \mathbb{1}_{\widehat{\mathcal{B}}_j}(x_i)$, where $\mathbb{1}_{\mathcal{A}}(x) = \mathbb{1}(x \in \mathcal{A})$ with $\mathbb{1}(\cdot)$ denoting the indicator function. This binning approach is the most popular in empirical work but, for completeness, all commands in the package Binsreg also allow for evenly-spaced binning. See below for more implementation details.

Given the quantile-spaced partitioning/binning scheme, for a choice of number of bins J, a generalized binscatter estimator of the v-th derivative of $\mu(x)$, employing a

p-th order polynomial approximation within each bin, imposing s-times differentiability across bins, and adjusting for additional covariates $\mathbf{w}_i$, is given by

$$\widehat{\mu}^{(v)}(x) = \widehat{\mathbf{b}}_s^{(v)}(x)'\widehat{\boldsymbol{\beta}}, \qquad \begin{bmatrix} \widehat{\boldsymbol{\beta}} \\ \widehat{\boldsymbol{\gamma}} \end{bmatrix} = \underset{\boldsymbol{\beta},\boldsymbol{\gamma}}{\arg\min} \sum_{i=1}^{n} (y_i - \widehat{\mathbf{b}}_s(x_i)'\boldsymbol{\beta} - \mathbf{w}_i'\boldsymbol{\gamma})^2,$$

where $s \leq p$, $v \leq p$, and $\widehat{\mathbf{b}}_s(x) = \widehat{\mathbf{T}}_s\widehat{\mathbf{b}}(x)$ with

$$\widehat{\mathbf{b}}(x) = \begin{bmatrix} \mathbb{1}_{\widehat{\mathcal{B}}_1}(x) & \mathbb{1}_{\widehat{\mathcal{B}}_2}(x) & \cdots & \mathbb{1}_{\widehat{\mathcal{B}}_J}(x) \end{bmatrix}' \otimes \begin{bmatrix} 1 & x & \cdots & x^p \end{bmatrix}',$$

being the p-th order polynomial basis of approximation within each bin, hence of dimension $(p+1)J$, and $\widehat{\mathbf{T}}_s$ being a $[(p+1)J - (J-1)s] \times (p+1)J$ matrix of linear restrictions ensuring that the $(s-1)$-th derivative of $\widehat{\mu}(x)$ is continuous.

When $s = 0$, $\widehat{\mathbf{T}}_0 = \mathbf{I}_{(p+1)J}$, the identity matrix of dimension $(p+1)J$, and therefore no restrictions are imposed: $\widehat{\mathbf{b}}(x) = \widehat{\mathbf{b}}_0(x)$ is the basis used for (disjoint) piecewise p-th order polynomial fits. Consequently, the binscatter $\widehat{\mu}(x)$ is discontinuous at the bins' edges whenever $s = 0$. On the other hand, $p \geq s$ implies that a least squares p-th order polynomial fit is constructed within each bin $\widehat{\mathcal{B}}_j$, in which case setting $s = 1$ forces these fits to be connected at the boundaries of adjacent bins, $s = 2$ forces these fits to be connected and continuously differentiable at the boundaries of adjacent bins, and so on for each $s = 3, 4, \ldots, p$.

Enforcing smoothness on binscatter boils down to incorporating restrictions on the basis of approximation. The resulting constrained basis, $\widehat{\mathbf{b}}_s(x)$, corresponds to a choice of spline basis for approximation of $\mu(\cdot)$, with estimated quantile-spaced knots according to the partition $\widehat{\Delta}$. The package Binsreg employs $\widehat{\mathbf{T}}_s$ leading to B-splines, which tend to have very good finite sample properties. When $p = s = 0$ binscatter reduces to the canonical binscatter commonly found in empirical work.

Specifically, in canonical binscatter the basis $\widehat{\mathbf{b}}(x)$ is a J-dimensional vector of orthogonal dummy variables, that is, the j-th component of $\widehat{\mathbf{b}}(x)$ records whether the evaluation point x belongs to the j-th bin in the partition $\widehat{\Delta}$. Therefore, canonical binscatter can be expressed as the collection of J sample averages of the response variable y_i, one for each bin: $\bar{y}_j = \frac{1}{N_j}\sum_{i=1}^{n} \mathbb{1}_{\widehat{\mathcal{B}}_j}(x_i)y_i$ for $j = 1, 2, \ldots, J$. Empirical work employing canonical binscatter typically plots these binned samples averages along with some other estimate(s) of the regression function $\mu(x)$.

Finally, covariate adjustment in $\widehat{\mu}^{(v)}(x)$ is justified via model (1), and does not coincide with the one commonly used by most practitioners and other software implementations of binscatter (c.f., Stepner 2014). An alternative covariate-adjustment approach is based on residualization, that is, first regressing out the additional covariates $\mathbf{w}_i$ and then constructing binscatter based on the residuals only. This alternative covariate-adjustment approach is very hard to rationalize or justify in general, and will lead to an inconsistent estimator of $\mu(x)$ in model (1) unless very special assumptions hold. Furthermore, even when model (1) is misspecified, the approach to covariate adjustment employed by the package Binsreg enjoys a natural probability limit inter-

pretation, while the residualization approach does not. See CCFF for more discussion, numerical examples, and technical details.

Main implementation details

The command `binsreg` implements multiple versions of $\widehat{\mu}^{(v)}(x)$ for a common choice of partitioning/binning $\widehat{\Delta}$. The option `deriv()` is used to set a common value of v across all implementations. The options `dots(p,s)` and `line(p,s)` generate "dots" and a "line" tracing out two distinct implementations of $\widehat{\mu}^{(v)}(x)$ with the corresponding choices of p and s selected in each case. Defaults are `deriv(0)` so that the object of interest is $\mu(x)$, and `dots(0,0)` so that the "dots" represent canonical binscatter (i.e., sample averages within each bin). The line option is muted by default, and needs to be set explicitly to appear in the resulting plot: for example, the option `line(3,3)` adds a line tracing out $\widehat{\mu}^{(v)}(x)$, implemented with $p = 3$ and $s = 3$, a cubic B-spline estimator of $\mu^{(v)}(x)$.

The common partitioning/binning used by the command `binsreg` across all implementations is set to be quantile-spaced for some choice of J. The option `nbins()` sets J manually (e.g., `nbins(20)` corresponds to $J = 20$ quantile-spaced bins), but if this option is not supplied then the companion command `binsregselect` is used to choose J in a fully data-driven way, as described below. As an alternative, an evenly-spaced partitioning/binning can be implemented via the option `binspos()`.

Several other options are available for the command `binsreg`, including optimal evenly-spaced binning via the command `binsregselect` and substantive specification or shape restrictions hypothesis testing via the command `binsregtest`, as discussed in detail further below. See Section 3 for the full syntax of `binsreg`.

2.2 Choosing the number of bins

CCFF develops valid integrated mean square error (IMSE) approximations for binscatter and its extensions in the context of model (1). These expansions give IMSE-optimal selection of the number bins J forming the quantile-spaced (or other) partitioning scheme $\widehat{\Delta}$, depending on polynomial order p within bins and smoothness level s across bins, and on the target estimand set by derivative order v. Specifically, the IMSE-optimal choice of J is given by:

$$J_{\texttt{IMSE}} = \left\lceil \left(\frac{2(p-v+1)\mathscr{B}_n(p,s,v)}{(1+2v)\mathscr{V}_n(p,s,v)} \right)^{\frac{1}{2p+3}} n^{\frac{1}{2p+3}} \right\rceil,$$

where $\lceil \cdot \rceil$ denotes the ceiling operator, and $\mathscr{B}_n(p,s,v)$ and $\mathscr{V}_n(p,s,v)$ represent an approximation to the integrated (square) bias and variance of $\widehat{\mu}^{(v)}(x)$, respectively. These constants depend on the partitioning scheme and binscatter estimator used. Recall that these three integer choices must respect $p \geq s \geq 0$ and $p \geq v \geq 0$.

Both IMSE constants, $\mathscr{B}_n(p,s,v)$ and $\mathscr{V}_n(p,s,v)$, can be estimated consistently using

a preliminary choice of J. Thus, our main implementation offers two J selectors:

- $\widehat{J}_{\texttt{ROT}}$: implements a rule-of-thumb (ROT) approximation for the constants $\mathscr{B}_n(p,s,v)$ and $\mathscr{V}_n(p,s,v)$, employing a trimmed-from-below Gaussian reference model for the density of x_i, and global polynomial approximations for the other two unknown features needed, $\mu^{(v)}(x)$ and $\mathbb{V}[y_i|x_i=x]$. This J selector employs the correct rate but an inconsistent constant approximation.
- $\widehat{J}_{\texttt{DPI}}$: implements a direct-plug-in (DPI) approximation for the constants $\mathscr{B}_n(p,s,v)$ and $\mathscr{V}_n(p,s,v)$, based on the desired binscatter, set by the choices p and s, and employing a preliminary J. If a preliminary J is not provided by the user, then $J=\max\{\widehat{J}_{\texttt{ROT}},(\frac{2(p-v+1)}{1+2v}n)^{\frac{1}{2p+3}}\}$ is used for DPI implementation. Therefore, the selector $\widehat{J}_{\texttt{DPI}}$ employs the correct rate and a nonparametric consistent constant approximation in the latter case.

The precise form of the constants changes depending on whether quantile-spaced or evenly-space partitioning/binning is used, but the two methods for selecting J, ROT and DPI, remain conceptually the same. See CCFF and references therein for more details.

Main implementation details

The command `binsregselect` implements ROT and DPI data-driven, IMSE-optimal selection of J for all possible choices of $p\geq v, s\geq 0$, and for both quantile-spaced or evenly-space partitioning/binning. For DPI implementation, the user can provide the initialization value of J or, if not provided, then $\widehat{J}_{\texttt{ROT}}$ is used.

Several other options are available for the command `binsregselect`, including the possibility of generating an output file with the IMSE-optimal partitioning/binning structure selected and the corresponding grid of evaluation points, which can be used by the other companion commands (`binsreg` and `binsregtest`) for plotting, simulation, testing, and other calculations. See Section 5 for the full syntax of `binsregselect`.

2.3 Confidence intervals

Both confidence intervals and confidence bands for the unknown function $\mu^{(v)}(x)$ are constructed employing the same type of Studentized t-statistic:

$$\widehat{T}_p(x)=\frac{\widehat{\mu}^{(v)}(x)-\mu^{(v)}(x)}{\sqrt{\widehat{\Omega}(x)/n}},\qquad 0\leq v,s\leq p,$$

where the binscatter variance estimator is of the usual "sandwich" form

$$\widehat{\Omega}(x)=\widehat{\mathbf{b}}_s^{(v)}(x)'\widehat{\mathbf{Q}}^{-1}\widehat{\mathbf{\Sigma}}\widehat{\mathbf{Q}}^{-1}\widehat{\mathbf{b}}_s^{(v)}(x),$$

with $\widehat{\mathbf{Q}}=\frac{1}{n}\sum_{i=1}^n\widehat{\mathbf{b}}_s(x_i)\widehat{\mathbf{b}}_s(x_i)'$ and $\widehat{\mathbf{\Sigma}}=\frac{1}{n}\sum_{i=1}^n\widehat{\mathbf{b}}_s(x_i)\widehat{\mathbf{b}}_s(x_i)'(y_i-\widehat{\mathbf{b}}_s(x_i)'\widehat{\boldsymbol{\beta}}-\mathbf{w}_i'\widehat{\boldsymbol{\gamma}})^2$.

CCFF showed that $\widehat{T}_p(x) \to_d \mathsf{N}(0,1)$ pointwise in x, that is, for each evaluation point x on the support of x_i, provided the misspecification error introduced by binscatter is removed from the distributional approximation. Such a result justifies asymptotically valid confidence intervals for $\mu^{(v)}(x)$, pointwise in x, after bias correction. Specifically, for each x, the $(1-\alpha)\%$ confidence interval takes the form:

$$\widehat{I}_p(x) = \left[\, \widehat{\mu}^{(v)}(x) \pm \Phi^{-1}(1-\alpha/2) \cdot \sqrt{\widehat{\Omega}(x)/n} \,\right], \qquad 0 \leq v, s \leq p,$$

where $\Phi(u)$ denotes the distribution function of a standard normal random variable (e.g., $\Phi^{-1}(1-0.05/2) \approx 1.96$ for a 95% Gaussian confidence intervals), and provided the choice of J is such that the misspecification error can be ignored.

However, employing an IMSE-optimal binscatter (i.e., setting $J = J_{\texttt{IMSE}}$ for the selected polynomial order p) introduces a first-order misspecification error leading to invalidity of these confidence intervals, and hence cannot be directly used to form the confidence intervals $\widehat{I}_p(x)$ in general. To address this problem, we rely on a simple application of robust bias-correction (Calonico, Cattaneo, and Titiunik 2014; Calonico, Cattaneo, and Farrell 2018, 2019; Cattaneo, Farrell, and Feng 2018) to form valid confidence intervals based on IMSE-optimal binscatter, that is, without altering the partitioning scheme $\widehat{\Delta}$ used.

Our recommended implementation employs robust bias-corrected binscatter confidence intervals as follows. First, for a given choice of p, select the number of bins in $\widehat{\Delta}$ according to $J = J_{\texttt{IMSE}}$, which gives an IMSE-optimal binscatter (point estimator). Then, employ the confidence interval $\widehat{I}_{p+q}(x)$ with $q \geq 1$, which satisfy

$$\mathbb{P}\left[\mu^{(v)}(x) \in \widehat{I}_{p+q}(x)\right] \to 1-\alpha, \qquad \text{for all } x.$$

Main implementation details

The command `binsreg` implements confidence intervals, and reports them as part of the final binned scatter plot. Specifically, the option `ci(p,s)` estimates confidence intervals with the corresponding choices of p and s selected, and plots them as vertical segments along the support of x_i. The implementation is done over a grid of evaluations points, which can be modified via the option `cigrid()`, and the desired level is set by the option `level()`. Notice that `dots(p,s)`, `lines(p,s)`, and `ci(p,s)` may all take different choices of p and s, which allows for robust bias-correction implementation of the confidence intervals and permits incorporating different levels of smoothness restrictions.

Several other options are available for the command `binsreg`, including substantive hypothesis testing via the companion command `binsregtest`, as described in detail further below. See Section 3 for the full syntax of `binsreg`.

2.4 Confidence bands

In many empirical applications of binscatter, the goal is to conduct inference about the entire function $\mu^{(v)}(\cdot)$, simultaneously, that is, uniformly over all x on the support

of x_i. This goal is fundamentally different from pointwise inference. A leading example of uniform inference is reporting confidence bands for $\mu(x)$ and its derivatives, which are different from (pointwise) confidence intervals. The package Binsreg offers asymptotically valid constructions of both confidence intervals, as discussed above, and confidence bands, which can be implemented with the same choices of (p, s) used to construct $\widehat{\mu}^{(v)}(x)$ or different ones.

Following the theoretical work in CCFF, for a choice of p and partition/binning of size J, the $(1-\alpha)\%$ confidence band for $\mu^{(v)}(\cdot)$ is:

$$\widehat{I}_p(\cdot) = \Big[\ \widehat{\mu}^{(v)}(\cdot) \pm \mathfrak{c} \cdot \sqrt{\widehat{\Omega}(\cdot)/n}\ \Big], \qquad 0 \le v, s \le p,$$

where the quantile value $\mathfrak{c}$ is now approximated via simulations using

$$\mathfrak{c} = \inf\Big\{c \in \mathbb{R}_+ : \mathbb{P}\Big[\sup_x \big|\widehat{Z}_p(x)\big| \le c \ \Big|\ \mathbf{D}\Big] \ge 1-\alpha\Big\},$$

with $\mathbf{D} = \{(y_i, x_i, \mathbf{w}_i') : 1 \le i \le n\}$ denoting the original data,

$$\widehat{Z}_p(x) = \frac{\widehat{\mathbf{b}}^{(v)}(x)'\widehat{\mathbf{Q}}^{-1}\widehat{\mathbf{\Sigma}}^{-1/2}}{\sqrt{\widehat{\Omega}(x)/n}}\mathbf{N}_K, \qquad K = (p+1)J - (J-1)s, \qquad 0 \le v, s \le p,$$

and $\mathbf{N}_K \sim \mathsf{N}(\mathbf{0}, \mathbf{I})$ being a K-dimensional standard normal random vector. The distribution of $\sup_x \big|\widehat{T}_p(x)\big|$, which is unknown, is approximated by that of $\sup_x \big|\widehat{Z}_p(x)\big|$ conditional on the data $\mathbf{D}$, which can be simulated by taking repeated samples from $\mathbf{N}_K$ and recomputing the supremum each time. In other words, the quantiles used to construct confidence bands can be approximated by resampling from the standard normal random vector $\mathbf{N}_{(p+1)J-(J-1)s}$, keeping fixed the data $\mathbf{D}$ (and hence all quantities depending on it). See CCFF for more details.

A confidence band covers $(1-\alpha)\%$ of the time the entire function $\mu^{(v)}(x)$ in repeated sampling, whenever the misspecification error can be ignored. As before, we recommend employing robust bias correction to remove misspecification error introduced by binscatter, that is, following the same logic discussed above for the case of confidence intervals construction. To be more precise, first p is chosen, along with s and v, and the optimal partitioning/binning is selected according to $J = J_{\texttt{IMSE}}$. Then, the confidence bands are constructed using $\widehat{I}_{p+q}(x)$ with $q \ge 1$. This ensures that

$$\mathbb{P}\big[\mu^{(v)}(x) \in \widehat{I}_{p+q}(x), \text{ for all } x\big] \to 1-\alpha.$$

Main implementation details

The command `binsreg` implements confidence bands, and reports them as part of the final binned scatter plot. The option `cb(p,s)` estimates an asymptotically valid confidence band with the corresponding choices of p and s selected, and plots it as a shaded region along the support of x_i. The implementations is done over a grid of evaluations points, which can be either modified via the option `cbgrid()`, and the desired

level is set by the option `level()`. The options `dots(p,s)`, `lines(p,s)`, `ci(p,s)`, and `cb(p,s)` can all take different choices of p and s, which allows for robust bias correction implementations, as well as many other practically relevant possibilities.

See Section 3 for the full syntax of `binsreg`.

2.5 Parametric specification testing

In addition to implementing binscatter and producing binned scatter plots, with both point and uncertainty estimators, the package Binsreg also allows for formal testing of substantive hypotheses. The command `binsregtest` implements all hypothesis tests available. This command can be used as a stand-alone command, or can be called via `binsreg` when constructing binned scatter plots.

The command `binsregtest` implements two types of substantive hypothesis tests about $\mu^{(v)}(x)$: (i) parametric specification testing, and (ii) nonparametric shape restriction testing. This subsection discusses the first type of hypothesis testing, while the next subsection discusses the second one.

For a choice of (p, s, v), and partitioning/binning scheme of size J, the implemented parametric specification testing approach contrasts a (nonparametric) binscatter approximation $\widehat{\mu}^{(v)}(x)$ of $\mu^{(v)}(x)$ with a hypothesized parametric specification of the form $\mu(x) = m(x, \boldsymbol{\theta})$ for some $m(\cdot)$ known up to a finite parameter $\boldsymbol{\theta}$, which can be estimated using the available data. Formally, the null and alternative hypothesis are, respectively,

$$\begin{aligned} \ddot{\mathsf{H}}_0 : &\quad \sup_x \left|\mu^{(v)}(x) - m^{(v)}(x, \boldsymbol{\theta})\right| = 0, \quad \text{for some } \boldsymbol{\theta} \in \boldsymbol{\Theta}, \qquad vs. \\ \ddot{\mathsf{H}}_{\mathrm{A}} : &\quad \sup_x \left|\mu^{(v)}(x) - m^{(v)}(x, \boldsymbol{\theta})\right| > 0, \quad \text{for all } \boldsymbol{\theta} \in \boldsymbol{\Theta}, \end{aligned}$$

for choice of derivative order v.

For example, $\widehat{\mu}(x)$ is compared to $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ in order to assess whether there is a relationship between y_i and x_i or, more formally, whether $\mu(x)$ is a constant function. Similarly, it is possible to formally test for a linear, quadratic, or even non-linear parametric relationship $\mu(x) = m(x, \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ would be estimated from the data under the null hypothesis, that is, assuming that the postulated relationship is indeed correct.

Following CCFF, the command `binsregtest` employs the test statistic

$$\ddot{T}_p(x) = \frac{\widehat{\mu}^{(v)}(x) - m^{(v)}(x, \widehat{\boldsymbol{\theta}})}{\sqrt{\widehat{\Omega}(x)/n}}, \qquad 0 \leq v, s \leq p.$$

Then, a parametric specification hypothesis testing procedure is:

$$\text{Reject } \ddot{\mathsf{H}}_0 \qquad \text{if and only if} \qquad \sup_x |\ddot{T}_p(x)| \geq \mathfrak{c}, \tag{2}$$

where $\mathfrak{c} = \inf\{c \in \mathbb{R}_+ : \mathbb{P}[\sup_x |\widehat{Z}_p(x)| \leq c \mid \mathbf{D}] \geq 1-\alpha\}$ is again computed by simulation from a standard Gaussian random vector, conditional on the data $\mathbf{D}$, as in the case of

confidence bands already discussed. This testing procedure is an asymptotically valid α%-level test if the misspecification error is removed from the test statistic $\ddot{T}_p(x)$.

The command `binsregtest` employs robust bias correction by default: first p and s are chosen, and the partitioning/binning scheme is selected by setting $J = J_{\texttt{IMSE}}$ for these choices. Then, using this partitioning scheme, the testing procedure (2) is implemented with the choice $p + q$ instead of p, with $q \geq 1$. CCFF shows that, under regularity conditions, the resulting parametric specification testing approach controls Type I error with non-trivial power: for given p, $0 \leq v, s \leq p$, and $J = J_{\texttt{IMSE}}$,

$$\lim_{n\to\infty} \mathbb{P}\Big[\sup_x \big|\ddot{T}_{p+q}(x)\big| > \mathfrak{c} \Big] = \alpha, \qquad \text{under } \ddot{\mathsf{H}}_0,$$

and

$$\lim_{n\to\infty} \mathbb{P}\Big[\sup_x \big|\ddot{T}_{p+q}(x)\big| > \mathfrak{c} \Big] = 1, \qquad \text{under } \ddot{\mathsf{H}}_{\mathrm{A}},$$

where $q \geq 1$. This testing approach formalizes the intuitive idea that if the confidence band for $\mu^{(v)}(x)$ does not contain the parametric fit considered entirely, then such parametric fit is incompatible with the data, i.e., should be rejected.

Main implementation details

The command `binsregtest` implements parametric specification testing in two ways. First, polynomial regression (parametric) specification testing is implemented directly via the option `polyreg(P)`, where the null hypothesis is $m(x, \boldsymbol{\theta}) = \theta_0 + x\theta_1 + \cdots + x^P\theta_P$ and $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_P)'$ is estimated by least squares regression. For other parametrizations of $m(x, \boldsymbol{\theta})$, the command takes as input an auxiliary array/database (`dta` in Stata, or `csv` in `R`) via the option `testmodelparfit(`*`filename`*`)` containing the following columns/variables: grid of evaluation points in one column, and fitted values $m(x, \widehat{\boldsymbol{\theta}})$ (over the evaluation grid) for each parametric model considered in other columns/variables. The ordering of these variables is arbitrary, but they have to follow a naming rule: the evaluation grid has the same name as the independent variable x_i, and the names of other variables storing fitted values take the form `binsreg_fit*`. The binscatter (nonparametric) estimate used to construct the testing procedure is set by the options `testmodel(p,s)` and `deriv(v)`, and the partitioning/binning scheme selected.

See Section 4 for other options and more details on the syntax of this command.

2.6 Nonparametric shape testing

The second type of hypothesis tests implemented by the command `binsregtest` concern nonparametric testing of shape restrictions. For a choice of v, the null and alternative hypotheses of these hypothesis problems are:

$$\dot{\mathsf{H}}_0: \quad \sup_x \mu^{(v)}(x) \leq 0, \qquad vs. \qquad \dot{\mathsf{H}}_{\mathrm{A}}: \quad \sup_x \mu^{(v)}(x) > 0,$$

that is, one-sided testing problem to the right. For example, negativity, monotonicity and concavity of $\mu(x)$ correspond to $\mu(x) \leq 0$, $\mu^{(1)}(x) \leq 0$ and $\mu^{(2)}(x) \leq 0$, respectively. Of course, the analogous testing problem to the left is also implemented, but not discussed here to avoid unnecessary repetition.

The relevant Studentized test statistic for this class of testing problems is:

$$\dot{T}_p(x) = \frac{\widehat{\mu}^{(v)}(x)}{\sqrt{\widehat{\Omega}(x)/n}}, \qquad 0 \leq v, s \leq p.$$

Then, the testing procedure is:

$$\text{Reject } \dot{\mathsf{H}}_0 \qquad \text{if and only if} \qquad \sup_x \dot{T}_p(x) \geq \mathfrak{c}, \tag{3}$$

with $\mathfrak{c} = \inf\{c \in \mathbb{R}_+ : \mathbb{P}[\sup_x \widehat{Z}_p(x) \leq c \mid \mathbf{D}] \geq 1-\alpha\}$. As before, misspecification errors of binscatter need to be taken into account in order to control Type I error. As in previous cases, CCFF show that for given p, $0 \leq v, s \leq p$, and $J = J_{\texttt{IMSE}}$ accordingly, then

$$\lim_{n\to\infty} \mathbb{P}\Big[\sup_x \dot{T}_{p+q}(x) > \mathfrak{c}\Big] \leq \alpha, \qquad \text{under } \dot{\mathsf{H}}_0,$$

and

$$\lim_{n\to\infty} \mathbb{P}\Big[\sup_x \dot{T}_{p+q}(x) > \mathfrak{c}\Big] = 1, \qquad \text{under } \dot{\mathsf{H}}_{\mathrm{A}},$$

for any $q \geq 1$, that is, using a robust bias-correction approach. These results imply that the testing procedure (3) is an asymptotically valid hypothesis test provided it is implemented with the choice $q \geq 1$ after the IMSE-optimal partitioning/binning scheme for binscatter of order p is selected.

Main implementation details

The command `binsregtest` implements one-sided and two-sided nonparametric shape restriction testing as follows. Option `testshapel(a)` implements one-sided testing to the left: $\dot{\mathsf{H}}_0 : \sup_x \mu^{(v)}(x) \leq \mathtt{a}$. Option `testshaper(a)` for one-sided to the right: $\dot{\mathsf{H}}_0 : \inf_x \mu^{(v)}(x) \geq \mathtt{a}$. Option `testshape2(a)` for two-sided testing: $\dot{\mathsf{H}}_0 : \sup_x |\mu^{(v)}(x) - \mathtt{a}| = 0$. The constant `a` needs to be specified by the user. The binscatter (nonparametric) estimate used to construct the testing procedure is set by the options `testshape(p,s)` and `deriv(v)`, and the chosen partitioning/binning scheme.

See Section 4 for more details on the syntax of this command.

2.7 Extensions and other implementation details

Whenever possible, the package Binsreg is implemented using the general purpose least squares regression command `regress` in `Stata`. (In `R`, the function `lm()` is used as the building block for implementation.) This approach sacrifices speed of the implementation, but improves substantially in terms of stability and replicability.

This section reviews some specific extensions and other numerical issues of the package Binsreg and discusses related choices made for implementation, all of which can affect speed and/or robustness of the package.

Mass points and minimum effective sample size

All three commands in the package Binsreg incorporate specific implementation decisions to deal with mass points in the distribution of the independent variable x_i. The number of distinct values of x_i, denoted by N, is taken as the effective sample size as opposed to the total number of observations n. If x_i is continuously distributed, then $N = n$. However, in many applications, N can be substantially smaller than n, and this affects some of the implementations in the package.

First, assume that J is set by the user (via the option `nbins(J)`). Then, given the choice J, the commands `binsreg` and `binsregtest` perform a degrees of freedom check to decide whether the x_i data exhibit enough variation. Specifically, given p and s set by the option `dots(p,s)`, both commands check whether $N > N_1+(p+1)J-(J-1)s$ with $N_1 = 30$ by default. If this check is not passed, then the package Binsreg regards the data as having "too little" variation in x_i, and turns off all nonparametric estimation and inference results based on large sample approximations. Thus, in this extreme case, the command `binsreg` only allows for `dots(0,0)`, `ci(0,0)`, and `polyreg(P)` for any $P+1 < N$, while the command `binsregtest` does not return any results and issues a warning message instead.

If, on the other hand, for given J, the numerical check $N > N_1+(p+1)J-(J-1)s$ is passed, then all nonparametric methods implemented by the commands `binsreg` and `binsregtest` become available. However, before implementing each method (`dots(p,s)`, `lines(p,s)`, `ci(p,s)`, `cb(p,s)`, `polyreg(p,s)`, and the hypothesis testing procedures), a degrees of freedom check is performed in each individual case. Specifically, each nonparametric procedure is implemented only if $N > N_1+(p+1)J-(J-1)s$, where recall that p and s may change from one procedure to the next.

Second, as discussed above, whenever J is not set by the user via the option `nbins()`, the command `binsregselect` is employed to select J in a data-driven way, provided there is enough variation in x_i. To determine the latter, an initial degrees of freedom check is performed to assess whether J selection is possible or, alternatively, if the unique values of x_i should be used as bins directly. Specifically, if $N > N_2 + p + 1$, with p set by the option `dots(p,s)` and $N_2 = 20$ by default, then the data is deemed appropriate for ROT selection of J via the command `binsregselect`, and hence $\widehat{J}_{\texttt{ROT}}$ is implemented. If, in addition, $N > N_1 + (p+1)\widehat{J}_{\texttt{ROT}} - (\widehat{J}_{\texttt{ROT}} - 1)s$, then $\widehat{J}_{\texttt{DPI}}$ is also implemented whenever requested. Furthermore, the command `binsregselect` employs the following alternative formula for J selection:

$$J_{\texttt{IMSE}} = \left\lceil \left(\frac{2(p-v+1)\mathscr{B}_n(p,s,v)}{(1+2v)\mathscr{V}_n(p,s,v)} \right)^{\frac{1}{2p+3}} N^{\frac{1}{2p+3}} \right\rceil,$$

with a slightly different constant $\mathscr{V}_n(p,s,v)$, taking into account the frequency of data

at each mass point. All other estimators in the package Binsreg, including bias and standard error estimators, automatically adapt to the presence of mass points. Once the final J is estimated, the degrees of freedom checks discussed in the previous paragraphs are performed based on this choice.

If J is not set by the user and $N \leq N_2 + p + 1$, so that not even ROT estimation of J is possible, then N is taken as "too small." In this extreme case, the package Binsreg sets $J = N$ and constructs a partitioning/binning structure with each bin containing one unique value of x_i. In other words, the support of the raw data is taken as the binning structure itself. In this extreme case, the follow up degrees of freedom checks based on the formula $N > N_1 + (p+1)J - (J-1)s$ fail by construction, and hence the nonparametric asymptotic methods are turned off as explained above.

Finally, the specific numerical checks and corresponding adjustments mentioned in this subsection can be modified or omitted. This is controlled by two main options: `dfcheck()` and `masspoints()`, respectively. First, the default cutoff points N_1 and N_2, corresponding to the degrees of freedom checks for nonparametric binscatter and parametric global polynomial, respectively, can be modified using the option `dfcheck(N1 N2)`. Second, the option `masspoints()` controls how the package Binsreg handles the presence of mass points (i.e., repeated values) in x_i. Specifically, setting `masspoints(noadjust)` omits mass point checks and the corresponding effective sample size adjustments, that is, it sets $N = n$ and ignores the presence of mass points in x_i (if any). Setting `masspoints(nolocalcheck)` omits within-bin mass point checks, but still performs global mass point checks and adjustments. The option `masspoints(off)` corresponds to setting both `masspoints(noadjust)` and `masspoints(nolocalcheck)` simultaneously. Finally, setting `masspoints(veryfew)` forces the package to proceed as if N is so small that all checks are failed, thereby treating x_i as if it has very few distinct values.

Clustered data and minimum effective sample size

As discussed in CCFF, the main methodological results for binscatter can be extended to accommodate clustered data. All three commands in the package Binsreg allow for clustered data via the option `vce()`. In this case, the number of clusters G is taken as the effective sample size, assuming $N = n$ (see below for the other case). The only substantive change occurs in the command `binsregselect`, which now employs the following alternative formula for J selection:

$$J_{\texttt{IMSE}} = \left\lceil \left(\frac{2(p-v+1)\mathscr{B}_n(p,s,v)}{(1+2v)\mathscr{V}_n(p,s,v)} \right)^{\frac{1}{2p+3}} \; G^{\frac{1}{2p+3}} \right\rceil,$$

with a variance constant $\mathscr{V}_n(p,s,v)$ accounting for the clustered structure of the data. Accordingly, cluster-robust variance estimators are used in this case.

Minimum effective sample size

The package Binsreg requires some minimal variation in x_i in order to successfully implement nonparametric methods based on large sample approximations. The minimal variation is captured by the number of distinct values on the support of x_i, denoted by N, and the number of clusters, denoted by G. Thus, all three commands in the pacakge perform degrees of freedom numerical checks using $\min\{n, N, G\}$ as the general definition of effective sample size, and proceeding as explained above for the case of mass points in the distribution of x_i.

3 `binsreg` syntax

The main purpose of the command `binsreg` is to produce binned scatter plots. This command implements multiple binscatter estimators, accompanying confidence intervals and confidence bands, and also a global polynomial approximation for completeness. It also implements hypothesis testing via the companion command `binsregtest`. A partitioning/binning structure is required but, if not provided, then one is selected in a data-driven way using the companion command `binsregselect`.

This section describes the syntax of the command `binsreg`, grouping its many options according to their use.

`binsreg` *depvar indvar* [*othercovs*] [*if*][*in*][*weight*][, `deriv(`*v*)

`dots(`*p s*) `dotsgrid(`*dotsgridopt*) `dotsplotopt(`*string*)

`line(`*p s*) `linegrid(`*numeric*) `lineplotopt(`*string*)

`ci(`*p s*) `cigrid(`*cigridopt*) `ciplotopt(`*string*)

`cb(`*p s*) `cbgrid(`*numeric*) `cbplotopt(`*string*)

`polyreg(`*p*) `polyreggrid(`*numeric*) `polyregcigrid(`*numlist*)
`polyregplotopt(`*string*)

`by(`*varname*) `bycolors(`*colorstylelist*) `bysymbols(`*symbolstylelist*)
`bylpatterns(`*linepatternstylelist*)

`testmodel(`*p s*) `testmodelparfit(`*filename*) `testmodelpoly(`*p*)

`testshape(`*p s*) `testshapel(`*numlist*) `testshaper(`*numlist*)
`testshape2(`*numlist*)

`nbins(`*J*) `binspos(`*numlist*) `binsmethod(`*string*)
`nbinsrot(`*numeric*) `samebinsby`

`nsims(`*S*) `simsgrid(`*numeric*) `simsseed(`*num*)

`dfcheck(`*n1 n2*) `masspoints(`*string*)

`vce(`*vcetype*) `level(`*numeric*) `noplot`

`savedata(`*filename*`) replace` *twoway_options*]

depvar is the dependent variable (y_i).

indvar is the independent variable (x_i).

othercovs is a varlist for covariate adjustment ($\mathbf{w}_i$).

p, s and v are integers satisfying $0 \leq s, v \leq p$.

weights allow for f`weights`, a`weights` and p`weights`; see `weights` in Stata for more details. (In `R`, *weights* allows for the equivalent of f`weights` only; see `lm()` help for more details.)

Estimand

`deriv(`v`)` specifies the derivative order of the regression function $\mu^{(v)}(x)$ for estimation, testing and plotting. The default is `deriv(0)`, which corresponds to the function itself, $\mu(x)$.

Dots

`dots(`p s`)` sets a piecewise polynomial of degree p with s smoothness constraints when constructing $\widehat{\mu}^{(v)}(x)$ for point estimation and plotting as "dots". The default is `dots(0 0)`, which corresponds to piecewise constant (canonical binscatter).

`dotsgrid(`*dotsgridopt*`)` specifies the number and location of dots within each bin to be plotted. Two options are available: *mean* and a *numeric* non-negative integer. The option `dotsgrid(mean)` adds the sample average of *indvar* within each bin to the grid of evaluation points for each bin. The option `dotsgrid(`*numeric*`)` adds *numeric* number of evenly-spaced points to the grid of evaluation points. Both options can be used simultaneously: for example, `cigrid(mean 5)` generates six evaluation points within each bin containing the sample mean of *indvar* within each bin and five evenly-spaced points. Given this choice, the dots are point estimates evaluated over the selected grid within each bin. The default is `dotsgrid(mean)`, which corresponds to one dot per bin evaluated at the sample average of *indvar* within each bin (canonical binscatter).

`dotsplotopt(`*string*`)` standard graphs options to be passed on to the `twoway` command to modify the appearance of the plotted dots.

Line

`line(`p s`)` sets a piecewise polynomial of degree p with s smoothness constraints when constructing $\widehat{\mu}^{(v)}(x)$ for point estimation and plotting as a "line". By default, the line is not included in the plot unless explicitly specified. Recommended specification is `line(3 3)`, which adds a cubic B-spline estimate of the regression function of

interest to the binned scatter plot.

`linegrid(`*numeric*`)` specifies the number of evaluation points of an evenly-spaced grid within each bin used for evaluation of the point estimate set by the `line(p s)` option. The default is `linegrid(20)`, which corresponds to 20 evenly-spaced evaluation points within each bin for fitting/plotting the line.

`lineplotopt(`*string*`)` standard graphs options to be passed on to the `twoway` command to modify the appearance of the plotted line.

Confidence Intervals

`ci(`p s`)` specifies the piecewise polynomial of degree p with s smoothness constraints used for constructing confidence intervals $\widehat{I}_p(x) = [\, \widehat{\mu}^{(v)}(x) \pm \Phi^{-1}(1-\alpha/2) \cdot \sqrt{\widehat{\Omega}(x)/n}\,]$. By default, the confidence intervals are not included in the plot unless explicitly specified. Recommended specification is `ci(3 3)`, which adds confidence intervals based on a cubic B-spline estimate of the regression function of interest to the binned scatter plot.

`cigrid(`*numeric*`)` specifies the number and location of evaluation points in the grid used to construct the confidence intervals set by the `ci(p s)` option. Two options are available: *mean* and a *numeric* non-negative integer. The option `dotsgrid(mean)` adds the sample average of *indvar* within each bin to the grid of evaluation points for each bin. The option `dotsgrid(`*numeric*`)` adds *numeric* number of evenly-spaced points to the grid of evaluation points. Both options can be used simultaneously: for example, `cigrid(mean 5)` generates six evaluation points within each bin containing the sample mean of *indvar* within each bin and five evenly-spaced points. The default is `cigrid(mean)`, which corresponds to one evaluation point set at the sample average of *indvar* within each bin for confidence interval construction.

`ciplotopt(`*string*`)` standard graphs options to be passed on to the `twoway` command to modify the appearance of the confidence intervals.

Confidence Band

`cb(`p s`)` specifies the piecewise polynomial of degree p with s smoothness constraints used for constructing the confidence band $\widehat{I}_p(\cdot) = [\, \widehat{\mu}^{(v)}(\cdot) \pm \Phi^{-1}(1-\alpha/2) \cdot \sqrt{\widehat{\Omega}(\cdot)/n}\,]$. By default, the confidence band is not included in the plot unless explicitly specified. Recommended specification is `cb(3 3)`, which adds a confidence band based on a cubic B-spline estimate of the regression function of interest to the binned scatter plot.

`cbgrid(`*numeric*`)` specifies the number of evaluation points of an evenly-spaced grid within each bin used for evaluation of the point estimate set by the `cb(p s)` option. The default is `cbgrid(20)`, which corresponds to 20 evenly-spaced evaluation points within each bin for confidence band construction.

`cbplotopt(`*string*`)` standard graphs options to be passed on to the `twoway` command to modify the appearance of the confidence band.

Global Polynomial Regression

`polyreg(`p`)` sets the degree P of a global polynomial regression model for plotting. By default, this fit is not included in the plot unless explicitly specified. Recommended specification is `polyreg(3)`, which adds a fourth order global polynomial fit of the regression function of interest to the binned scatter plot.

`polyreggrid(`*numeric*`)` specifies the number of evaluation points of an evenly-spaced grid within each bin used for evaluation of the point estimate set by the `polyreg(p)` option. The default is `polyreggrid(20)`, which corresponds to 20 evenly-spaced evaluation points within each bin for confidence interval construction.

`polyregcigrid(`*numeric*`)` specifies the number of evaluation points of an evenly-spaced grid within each bin used for constructing confidence intervals based on polynomial regression set by the `polyreg(p)` option. The default is `polyregcigrid(0)`, which corresponds to not plotting confidence intervals for the global polynomial regression approximation.

`polyregplotopt(`*string*`)` standard graphs options to be passed on to the `twoway` command to modify the appearance of the global polynomial regression fit.

Subgroup Analysis

`by(`*varname*`)` specifies the variable containing the group indicator to perform subgroup analysis; both numeric and string variables are supported. When `by(`*varname*`)` is specified, `binsreg` implements estimation and inference by each subgroup separately, but produces a common binned scatter plot. By default, the binning structure is selected for each subgroup separately, but see the option `samebinsby` below for imposing a common binning structure across subgroups.

`bycolors(`*colorstylelist*`)` specifies an ordered list of colors for plotting each subgroup series defined by the option `by()`.

`bysymbols(`*symbolstylelist*`)` specifies an ordered list of symbols for plotting each subgroup series defined by the option `by()`.

`bylpatterns(`*linepatternstylelist*`)` specifies an ordered list of line patterns for plotting each subgroup series defined by the option `by()`.

Parametric Model Specification Testing

`testmodel(`p s`)` sets a piecewise polynomial of degree p with s smoothness constraints for parametric model specification testing, implemented via the companion command `binsregtest`. The null hypothesis is $\mathsf{H}_0 : \sup_x |\mu^{(v)}(x) - m^{(v)}(x, \boldsymbol{\theta})| = 0$. The

default is `testmodel(3 3)`, which corresponds to a cubic B-spline estimate of the regression function of interest for testing against the fitting from a parametric model specification.

`testmodelparfit(`*filename*`)` specifies a dataset which contains the evaluation grid and fitted values of the model(s) to be tested against. The file must have a variable with the same name as *indvar*, which contains a series of evaluation points at which the binscatter model and the parametric model of interest are compared with each other. Each parametric model is represented by a variable named as *binsreg_fit**, which must contain the fitted values at the corresponding evaluation points.

`testmodelpoly(`p`)` specifies the degree of a global polynomial model P to be tested against.

Nonparametric Shape Restriction Testing

`testshape(`p s`)` sets a piecewise polynomial of degree p with s smoothness constraints for nonparametric shape restriction testing, implemented via the companion command `binsregtest`. The default is `testmodel(3 3)`, which corresponds to a cubic B-spline estimate of the regression function of interest for one-sided or two-sided testing.

`testshapel(`*numlist*`)` specifies a *numlist* of null boundary values for hypothesis testing. Each number a in the *numlist* corresponds to one boundary of a one-sided hypothesis test to the left of the form $\dot{\mathsf{H}}_0 : \sup_x \mu^{(v)}(x) \leq a$.

`testshaper(`*numlist*`)` specifies a *numlist* of null boundary values for hypothesis testing. Each number a in the *numlist* corresponds to one boundary of a one-sided hypothesis test to the right of the form $\dot{\mathsf{H}}_0 : \inf_x \mu^{(v)}(x) \geq a$.

`testshape2(`*numlist*`)` specifies a *numlist* of null boundary values for hypothesis testing. Each number a in the *numlist* corresponds to one boundary of a two-sided hypothesis test of the form $\dot{\mathsf{H}}_0 : \sup_x |\mu^{(v)}(x) - a| = 0$.

Partitioning/Binning Selection

`nbins(`J`)` sets the number of bins J for partitioning/binning of *indvar*. If not specified, the number of bins is selected via the companion command `binsregselect` in a data-driven, optimal way whenever possible.

`binspos(`*numlist*`)` specifies the position of binning knots. The default is `binspos(qs)`, which corresponds to quantile-spaced binning (canonical binscatter). Other options are: `es` for evenly-spaced binning, or a *numlist* for manual specification of the positions of inner knots (which must be within the range of *indvar*).

`binsmethod(`*string*`)` specifies the method for data-driven selection of the number of bins via the companion command `binsregtest`. The default is `binsmethod(dpi)`, which corresponds to the IMSE-optimal direct plug-in rule $\widehat{J}_{\texttt{DPI}}$. The other option is: `rot`

for rule of thumb implementation, $\widehat{J}_{\texttt{ROT}}$.

`nbinsrot(`*numeric*`)` specifies an initial number of bins value used to construct the DPI number of bins selector via the the companion command `binsregtest`. If not specified, the data-driven ROT selector is used instead.

`samebinsby` forces a common partitioning/binning structure across all subgroups specified by the option `by()`. The knots positions are selected according to the option `binspos()` and using the full sample. If `nbins()` is not specified, then the number of bins is selected via the companion command `binsregselect` and using the full sample.

Simulation

`nsims(`S`)` specifies the number of random draws S for constructing confidence bands and hypothesis testing. The default is `nsims(500)`, which corresponds to 500 draws from a standard Gaussian random vector of size $[(p+1)\cdot J-(J-1)\cdot s]$.

`simsgrid(`*numeric*`)` specifies the number of evaluation points of an evenly-spaced grid within each bin used for evaluation of the supremum (or infimum) operation needed to construct confidence bands and hypothesis testing procedures. The default is `simsgrid(20)`, which corresponds to 20 evenly-spaced evaluation points within each bin for approximating the supremum (or infimum) operator.

`simsseed(`*numeric*`)` sets the seed for simulations.

Mass Points and Degrees of Freedom

`dfcheck(`*n1 n2*`)` sets cutoff values for minimum effective sample size checks, which take into account the number of unique values of *indvar* (i.e., adjusting for the number of mass points), number of clusters, and degrees of freedom of the different statistical models considered. Specifically, $N_1 = n1$ and $N_2 = n2$. The default is `dfcheck(20 30)`, as discussed above.

`masspoints(`*string*`)` specifies how mass points in *indvar* are handled. By default, all mass point and degrees of freedom checks are implemented. Available options:

noadjust	omits mass point checks and the corresponding effective sample size adjustments.
nolocalcheck	omits within-bin mass point and degrees of freedom checks.
off	sets `masspoints(noadjust)` and `masspoints(nolocalcheck)` simultaneously.
veryfew	forces the command to proceed as if *indvar* has only a few number of mass points (i.e., distinct values). In other words, forces the command to proceed as if the mass point and degrees of freedom checks were failed.

Other Options

`vce(`*vcetype*`)` specifies the *vcetype* for variance estimation used by the command `regress`. The default is `vce(robust)`.

`level(`*numeric*`)` sets the nominal confidence level $(1-\alpha)$ for confidence interval and confidence band estimation.

`noplot` omits binscatter plotting.

`savedata(`*filename*`)` specifies a *filename* for saving all data underlying the binscatter plot (and more).

`replace` overwrites the existing file when saving the graph data.

twoway_options any unrecognized options are appended to the end of the `twoway` command generating the binned scatter plot.

4 `binsregtest` syntax

The main purpose of the command `binsregtest` is to conduct hypothesis testing of parametric specifications and nonparametric shape restrictions for $\mu^{(v)}(x)$ using binscatter methods. This stand-alone command is used by the companion command `binsreg`. A partitioning/binning structure is required but, if not provided, then one is selected in a data-driven way using the companion command `binsregselect`.

This section describes the syntax of the command `binsregtest`, grouping its many options according to their use.

`binsregtest` *depvar indvar* [*othercovs*] [*if*] [*in*] [*weight*] [`, deriv(`*v*`)`
`testmodel(`*p s*`)` `testmodelparfit(`*filename*`)` `testmodelpoly(`*p*`)`
`testshape(`*p s*`)` `testshapel(`*numlist*`)` `testshaper(`*numlist*`)`
`testshape2(`*numlist*`)`
`bins(`*p s*`)` `nbins(`*J*`)` `binspos(`*numlist*`)` `binsmethod(`*string*`)`
`nbinsrot(`*numeric*`)`
`nsims(`*S*`)` `simsgrid(`*numeric*`)` `simsseed(`*num*`)`
`dfcheck(`*n1 n2*`)` `masspoints(`*string*`)`
`vce(`*vcetype*`)`]

depvar is the dependent variable (y_i).

indvar is the independent variable (x_i).

othercovs is a varlist for covariate adjustment $(\mathbf{w}_i)$.

p, s and v are integers satisfying $0 \leq s, v \leq p$.

weights allow for f`weights`, a`weights` and p`weights`; see `weights` in Stata for more details. (In `R`, *weights* allows for the equivalent of f`weights` only; see `lm()` help for more details.)

Estimand

`deriv(`v`)` specifies the derivative order of the regression function $\mu^{(v)}(x)$ for estimation, testing and plotting. The default is `deriv(0)`, which corresponds to the function itself, $\mu(x)$.

Parametric Model Specification Testing

`testmodel(`p s`)` sets a piecewise polynomial of degree p with s smoothness constraints for parametric model specification testing, implemented via the companion command `binsregtest`. The null hypothesis is $\ddot{\mathsf{H}}_0 : \sup_x |\mu^{(v)}(x) - m^{(v)}(x, \boldsymbol{\theta})| = 0$. The default is `testmodel(3 3)`, which corresponds to a cubic B-spline estimate of the regression function of interest for testing against the fitting from a parametric model specification.

`testmodelparfit(`*filename*`)` specifies a dataset which contains the evaluation grid and fitted values of the model(s) to be tested against. The file must have a variable with the same name as *indvar*, which contains a series of evaluation points at which the binscatter model and the parametric model of interest are compared with each other. Each parametric model is represented by a variable named as *binsreg_fit**, which must contain the fitted values at the corresponding evaluation points.

`testmodelpoly(`p`)` specifies the degree of a global polynomial model P to be tested against.

Nonparametric Shape Restriction Testing

`testshape(`p s`)` sets a piecewise polynomial of degree p with s smoothness constraints for nonparametric shape restriction testing, implemented via the companion command `binsregtest`. The default is `testmodel(3 3)`, which corresponds to a cubic B-spline estimate of the regression function of interest for one-sided or two-sided testing.

`testshapel(`*numlist*`)` specifies a *numlist* of null boundary values for hypothesis testing. Each number a in the *numlist* corresponds to one boundary of a one-sided hypothesis test to the left of the form $\dot{\mathsf{H}}_0 : \sup_x \mu^{(v)}(x) \leq a$.

`testshaper(`*numlist*`)` specifies a *numlist* of null boundary values for hypothesis testing. Each number a in the *numlist* corresponds to one boundary of a one-sided hypothesis test to the right of the form $\dot{\mathsf{H}}_0 : \inf_x \mu^{(v)}(x) \geq a$.

`testshape2(`*numlist*`)` specifies a *numlist* of null boundary values for hypothesis testing. Each number a in the *numlist* corresponds to one boundary of a two-sided hypothesis

test of the form $\dot{\mathsf{H}}_0 : \sup_x |\mu^{(v)}(x) - a| = 0$.

Partitioning/Binning Selection

`bins(`*p s*`)` sets a piecewise polynomial of degree p with s smoothness constraints for data-driven (IMSE-optimal) selection of the partitioning/binning scheme. The default is `bins(0 0)`, which corresponds to piecewise constant (canonical binscatter).

`nbins(`*J*`)` sets the number of bins J for partitioning/binning of *indvar*. If not specified, the number of bins is selected via the companion command `binsregselect` in a data-driven, optimal way whenever possible.

`binspos(`*numlist*`)` specifies the position of binning knots. The default is `binspos(qs)`, which corresponds to quantile-spaced binning (canonical binscatter). Other options are: `es` for evenly-spaced binning, or a *numlist* for manual specification of the positions of inner knots (which must be within the range of *indvar*).

`binsmethod(`*string*`)` specifies the method for data-driven selection of the number of bins via the companion command `binsregselect`. The default is `binsmethod(dpi)`, which corresponds to the IMSE-optimal direct plug-in rule $\widehat{J}_{\texttt{DPI}}$. The other option is: `rot` for rule of thumb implementation, $\widehat{J}_{\texttt{ROT}}$.

`nbinsrot(`*numeric*`)` specifies an initial number of bins value used to construct the DPI number of bins selector via the the companion command `binsregtest`. If not specified, the data-driven ROT selector is used instead.

Simulation

`nsims(`*S*`)` specifies the number of random draws S for constructing confidence bands and hypothesis testing. The default is `nsims(500)`, which corresponds to 500 draws from a standard Gaussian random vector of size $[(p+1)\cdot J - (J-1)\cdot s]$.

`simsgrid(`*numeric*`)` specifies the number of evaluation points of an evenly-spaced grid within each bin used for evaluation of the supremum (or infimum) operation needed to construct confidence bands and hypothesis testing procedures. The default is `simsgrid(20)`, which corresponds to 20 evenly-spaced evaluation points within each bin for approximating the supremum (or infimum) operator.

`simsseed(`*numeric*`)` sets the seed for simulations.

Mass Points and Degrees of Freedom

`dfcheck(`*n1 n2*`)` sets cutoff values for minimum effective sample size checks, which take into account the number of unique values of *indvar* (i.e., adjusting for the number of mass points), number of clusters, and degrees of freedom of the different statistical models considered. Specifically, $N_1 = n1$ and $N_2 = n2$. The default is `dfcheck(20 30)`, as discussed above.

`masspoints(`*string*`)` specifies how mass points in *indvar* are handled. By default, all mass point and degrees of freedom checks are implemented. Available options:

noadjust	omits mass point checks and the corresponding effective sample size adjustments.
nolocalcheck	omits within-bin mass point and degrees of freedom checks.
off	sets `masspoints(noadjust)` and `masspoints(nolocalcheck)` simultaneously.
veryfew	forces the command to proceed as if *indvar* has only a few number of mass points (i.e., distinct values). In other words, forces the command to proceed as if the mass point and degrees of freedom checks were failed.

Other Options

`vce(`*vcetype*`)` specifies the *vcetype* for variance estimation used by the command `regress`. The default is `vce(robust)`.

5 `binsregselect` syntax

The main purpose of the command `binsregselect` is to implement data-driven (IMSE-optimal) selection of partitioning/binning structure for binscatter. This stand-alone command is used by the companion commands `binsreg` and `binsregtest` whenever the user does not specify the binning structure manually.

This section describes the syntax of the command `binsregselect`, grouping its many options according to their use.

`binsregselect` *depvar indvar* [*othercovs*] [*if*][*in*][*weight*][`,` `deriv(`*v*`)`

`bins(`*p s*`)` `binspos(`*numlist*`)` `binsmethod(`*string*`)` `nbinsrot(`*numeric*`)`

`simsgrid(`*numeric*`)` `savegrid(`*filename*`)` `replace`

`dfcheck(`*n1 n2*`)` `masspoints(`*string*`)`

`vce(`*vcetype*`)` `useeffn(`*numeric*`)`]

depvar is the dependent variable (y_i).

indvar is the independent variable (x_i).

othercovs is a varlist for covariate adjustment ($\mathbf{w}_i$).

p, s and v are integers satisfying $0 \leq s, v \leq p$.

weights allow for f`weights`, a`weights` and p`weights`; see `weights` in Stata for more details. (In `R`, *weights* allows for the equivalent of f`weights` only; see `lm()` help for more details.)

Estimand

`deriv(`v`)` specifies the derivative order of the regression function $\mu^{(v)}(x)$ for estimation, testing and plotting. The default is `deriv(0)`, which corresponds to the function itself, $\mu(x)$.

Partitioning/Binning Selection

`bins(`p s`)` sets a piecewise polynomial of degree p with s smoothness constraints for data-driven (IMSE-optimal) selection of the partitioning/binning scheme. The default is `bins(0 0)`, which corresponds to piecewise constant (canonical binscatter).

`binspos(`*numlist*`)` specifies the position of binning knots. The default is `binspos(qs)`, which corresponds to quantile-spaced binning (canonical binscatter). Other options are: `es` for evenly-spaced binning, or a *numlist* for manual specification of the positions of inner knots (which must be within the range of *indvar*).

`binsmethod(`*string*`)` specifies the method for data-driven selection of the number of bins. The default is `binsmethod(dpi)`, which corresponds to the IMSE-optimal direct plug-in rule $\widehat{J}_{\texttt{DPI}}$. The other option is: `rot` for rule of thumb implementation, $\widehat{J}_{\texttt{ROT}}$.

`nbinsrot(`*numeric*`)` specifies an initial number of bins value used to construct the DPI number of bins selector. If not specified, the data-driven ROT selector is used instead.

Evaluation Points Grid Generation

`simsgrid(`*numeric*`)` specifies the number of evaluation points of an evenly-spaced grid within each bin used for evaluation of the supremum (or infimum) operation needed to construct confidence bands and hypothesis testing procedures. The default is `simsgrid(20)`, which corresponds to 20 evenly-spaced evaluation points within each bin for approximating the supremum (or infimum) operator.

`savegrid(`*filename*`)` specifies a *filename* for storing the simulation grid of evaluation points. It contains the following variables: *indvar*, which is a sequence of evaluation points grids used in approximation; all control variables in *covars*, which take values of zero for prediction purpose; *binsreg_isknot*, indicating whether the grid is an inner knot; and *binsreg_bin*, indicating which bin the grid belongs to.

`replace` overwrites the existing file when saving the grid.

Mass Points and Degrees of Freedom

`dfcheck(`*n1 n2*`)` sets cutoff values for minimum effective sample size checks, which take into account the number of unique values of *indvar* (i.e., adjusting for the number of mass points), number of clusters, and degrees of freedom of the different statistical

models considered. Specifically, $N_1 = n1$ and $N_2 = n2$. The default is `dfcheck(20 30)`, as discussed above.

`masspoints(`*string*`)` specifies how mass points in *indvar* are handled. By default, all mass point and degrees of freedom checks are implemented. Available options:

noadjust	omits mass point checks and the corresponding effective sample size adjustments.
nolocalcheck	omits within-bin mass point and degrees of freedom checks.
off	sets `masspoints(noadjust)` and `masspoints(nolocalcheck)` simultaneously.
veryfew	forces the command to proceed as if *indvar* has only a few number of mass points (i.e., distinct values). In other words, forces the command to proceed as if the mass point and degrees of freedom checks were failed.

Other Options

`vce(`*vcetype*`)` specifies the *vcetype* for variance estimation used by the command `regress`. The default is `vce(robust)`.

`useeffn(`*numeric*`)` specifies the effective sample size to be used when computing the (IMSE-optimal) number of bins. This option is useful for extrapolating the optimal number of bins to larger (or smaller) datasets than the one used to compute it.

6 Illustration of Methods

We illustrate the package Binsreg using a simulated dataset, which is available in the file `binscatter_simdata.dta`. In this dataset, `y` is the outcome variable, `x` is the independent variable for binning, `w` is a continuously distributed covariate, and `t` is a binary covariate, and `id` is a group identifier. Summary statistics of the simulated data are as follows.

```
. use binsreg_simdata, clear
. sum
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
           x |      1,000    .4907072    .2932553   .0002281   .9985808
           w |      1,000    .0120224    .5799381  -.9993055   .9973198
           t |      1,000        .515     .500025          0          1
          id |      1,000       250.5    144.4095          1        500
           y |      1,000    .5283884    1.727878  -5.159858   5.751276
```

The basic syntax for `binsreg` is the following:

```
. binsreg y x w
Sorting dataset on x...
Note: This step is omitted if dataset already sorted by x.
Binscatter plot
Bin selection method: IMSE-optimal plug-in choice
Placement: Quantile-spaced
```

```
Derivative: 0
```

# of observations	1000
# of distinct values	1000
# of clusters	.
Bin selection:	
Degree of polynomial	0
# of smoothness constraints	0
# of bins	21

	p	s	df
dots	0	0	21

The main output is a binned scatter plot as shown in Figure 1. By default, the (nonparametric) mean relationship between `y` and `x` is approximated by piecewise constants (`dots(0 0)`). Each dot in the figure represents the point estimate corresponding to each bin, which is the canonical binscatter plot. The number of bins, whenever not specified, is automatically selected via the companion command `binsregselect`. In this case, 21 bins are used. Other useful information is also reported, including total sample size, the number of distinct values of `x`, bin selection results, and the degrees of freedom of the statistical model(s) employed.

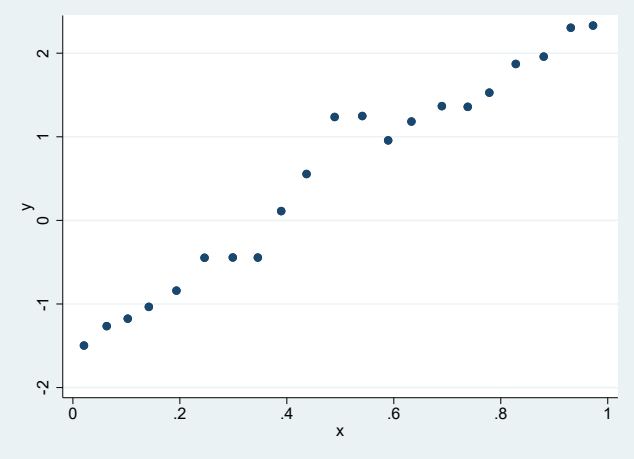


Figure 1: Canonical Binned Scatter Plot.

Users may specify the number of bins manually rather than relying on the automatic data-driven procedures. For example, a popular ad-hoc choice in practice is setting $J = 20$ quntile-spaced bins:

```
. binsreg y x w, nbins(20) polyreg(1)
Sorting dataset on x...
Note: This step is omitted if dataset already sorted by x.
Binscatter plot
Bin selection method: IMSE-optimal plug-in choice
```

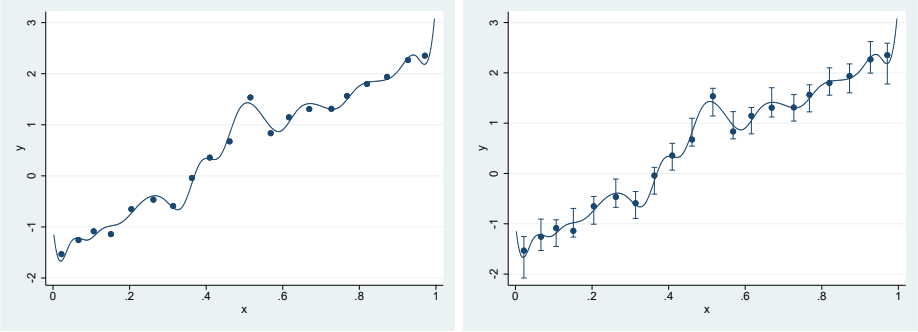
```
Placement: Quantile-spaced
Derivative: 0
```

# of observations	1000
# of distinct values	1000
# of clusters	.
Bin selection:	
Degree of polynomial	0
# of smoothness constraints	0
# of bins	20

	p	s	df
dots	0	0	20
polyreg	1	NA	2

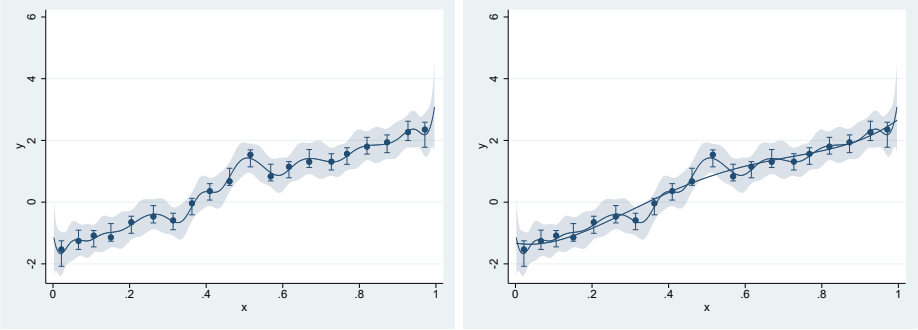
The option `polyreg(1)` adds a linear prediction line to the canonical binscatter plot, but the resulting binned scatter plot is not reported here to conserve space.

Figure 2: Binned Scatter Plot with Lines, Confidence Intervals and Bands.



(a) Add cubic B-spline fit

(b) Add confidence intervals



(c) Add confidence band

(d) Add a polynomial fit of degree 4

The command `binsreg` allows users to add a binscatter-based line approximating the unknown regression function, pointwise confidence intervals, a confidence band, and a

global polynomial regression approximation. For example, the following syntax cumulatively adds in four distinct plots a fitted line, confidence intervals and a confidence band, all three based on cubic B-splines, and also a fitted line based on a global polynomial of degree 4. The results are shown in Figure 2.

```
. qui binsreg y x w, nbins(20) dots(0,0) line(3,3)
. qui binsreg y x w, nbins(20) dots(0,0) line(3,3) ci(3,3)
. qui binsreg y x w, nbins(20) dots(0,0) line(3,3) ci(3,3) cb(3,3)
. qui binsreg y x w, nbins(20) dots(0,0) line(3,3) ci(3,3) cb(3,3) polyreg(4)
```

By construction, a cubic B-spline fit is a piecewise cubic polynomial function which is continuous, and has continuous first- and second-order derivatives. Thus, the prediction line and confidence band generated are quite smooth. In this case, it is arguably undersmoothed because of the "large" choice of $J = 20$. The degree and smoothness of polynomials can be changed by adjusting the values of `p` and `s` in the options `dots()`, `line()`, `ci()` and `cb()`.

The command `binsreg` also allows for the standard *weight* options, *vce* options, factor variables, and *twoway* graph options, among other features. This is illustrated in the following code:

```
. binsreg y x w i.t, dots(0,0) line(3,3) ci(3,3) cb(3,3) polyreg(4) ///
>                   vce(cluster id) savedata(output/graphdat) replace ///
>                                   title("Binned Scatter Plot")
Sorting dataset on x...
Note: This step is omitted if dataset already sorted by x.
Binscatter plot
Bin selection method: IMSE-optimal plug-in choice
Placement: Quantile-spaced
Derivative: 0
Output file: output/graphdat.dta
```

# of observations	1000
# of distinct values	1000
# of clusters	500

Bin selection:	
Degree of polynomial	0
# of smoothness constraints	0
# of bins	20

	p	s	df
dots	0	0	20
line	3	3	23
CI	3	3	23
CB	3	3	23
polyreg	4	NA	5

Specifically, a dummy variable based on the binary covariate `t` is added to the estimation, standard errors are clustered at the group level indicator `id`, and a graph title is added to the resulting binned scatter plot. Note that any unrecognized options for the command `binsreg` will be understood as `twoway` options and therefore appended to the final plot command. Thus, users may easily modify, for example, axis properties, legends, etc. The option `savedata(graphdat)` saves the underlying data used in the

binned scatter plot in the file `graphdat.dta`.

In addition, the command `binsreg` can be used for subgroup analysis. The following command implements binscatter estimation and inference across two subgroups separately, defined by the variable `t`, and then produces a common binned scatter plot (Figure 3):

```
. binsreg y x w, by(t) dots(0,0) line(3,3) cb(3,3) ///
>                bycolors(blue red) bysymbols(O T)
Sorting dataset on x...
Note: This step is omitted if dataset already sorted by x.
Binscatter plot
Bin selection method: IMSE-optimal plug-in choice
Placement: Quantile-spaced
Derivative: 0
Group: t = 0
```

# of observations	485
# of distinct values	485
# of clusters	.
Bin selection:	
Degree of polynomial	0
# of smoothness constraints	0
# of bins	20

	p	s	df
dots	0	0	20
line	3	3	23
CB	3	3	23

```
Group: t = 1
```

# of observations	515
# of distinct values	515
# of clusters	.
Bin selection:	
Degree of polynomial	0
# of smoothness constraints	0
# of bins	15

	p	s	df
dots	0	0	15
line	3	3	18
CB	3	3	18

Figure 3 highlights a difference across the two subgroups defined by the variable `t`, which corresponds to the fact that our simulated data adds a 1 to the outcome variable for those units with `t` == `1`. The colors, symbols, and line patterns in Figure 3 can be modified via the options `bycolors()`, `bysymbols()`, and `bylpatterns()`. When the number of bins is unspecified, the command `binsreg` selects the number of bins for each subsample separately, via the companion command `binsregselect`. This means that, by default, the choice of binning/partitioning structure will be different across subgroups in general. However, if the option `samebinsby` is specified, then a common

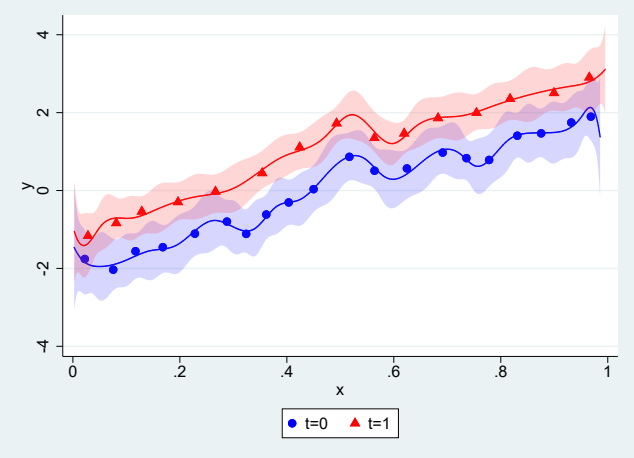


Figure 3: Binned Scatter Plot: Group Comparison

binning scheme for all subgroups is constructed based on the full sample.

Next, we illustrate the syntax of the command `binsregtest`. The basic syntax is the following:

```
. binsregtest y x w, testmodelpoly(1)
Hypothesis tests based on binscatter estimates
Bin selection method: IMSE-optimal plug-in choice
Placement:
Derivative: 0
```

# of observations	1000
# of distinct values	1000
# of clusters	.
Bin selection:	
Degree of polynomial	0
# of smoothness constraints	0
# of bins	21

```
Model specification Tests:
Degree: 3     # of smoothness constraints: 3
```

H0: mu =	sup |T|	p value
poly. degree 1	6.503	0.000

A test for linearity of the regression function $\mu(x)$ is implemented using the binscatter estimator. By default, a cubic B-spline is employed in the inference procedure, which can be adjusted by the option `testmodel()`. In addition, when unspecified, the number of bins is selected using a data-driven procedure via the companion command `binsregselect`. The selected number of bins is IMSE-optimal for piecewise constant point estimates by default. A summary of the sample and binning scheme is displayed, and then the test statistic and p-value are reported. In this case, the test statistic is

the supremum of the absolute value of the t-statistic evaluated over a sequence of grid points, and the p-value is calculated based on simulation. Clearly, the p-value is quite small, and thus the null hypothesis of linearity of the regression function is rejected.

The command `binsregtest` can implement testing for any parametric model specification by comparing the fitted values based on the binscatter estimator (computed by the command) and the parametric model of interest (provided by the user). For example, the following code creates an auxiliary database with a grid of evaluation points, implements a linear regression first, makes an out-of-sample prediction using the auxiliary dataset, and then tests for linearity based on the binscatter estimator by specifying the auxiliary file containing the fitted values.

```
. qui binsregselect y x w, simsgrid(30) savegrid(output/parfitval) replace
. qui reg y x w
. use output/parfitval, clear
. predict binsreg_fit_lm
(option xb assumed; fitted values)
. save output/parfitval, replace
file output/parfitval.dta saved
. use binsreg_simdata, clear
. binsregtest y x w, testmodelparfit(output/parfitval)
Hypothesis tests based on binscatter estimates
Bin selection method: IMSE-optimal plug-in choice
Placement:
Derivative: 0
```

# of observations	1000
# of distinct values	1000
# of clusters	.
Bin selection:	
Degree of polynomial	0
# of smoothness constraints	0
# of bins	21

```
Model specification Tests:
Degree: 3     # of smoothness constraints: 3
Input file: output/parfitval.dta
```

H0: mu =	sup |T|	p value
binsreg_fit_lm	6.503	0.000

The first command, `qui binsregselect y x w, simsgrid(30) savegrid(output/parfitval) replace` generates the auxiliary file containing the grid of evaluation points. Since the parameter of interest is only the mean relation between `y` and `x`, i.e., $\mu(x)$, at the out-of-sample prediction step, the testing dataset `parfitval.dta` must contain a variable `x` containing a sequence of evaluation points at which the binscatter and parametric models are compared, and the covariate `w` whose values are set as zeros. In addition, the variable containing fitted values has to follow a specific naming rule, i.e., takes the form of `binsreg_fit*`. The companion command `binsregselect` can be used to construct the required auxiliary dataset, as illustrated above. We discuss this other command further below.

In addition to model specification tests, the command `binsregtest` can test for nonparametric shape restrictions on the regression function. For example, the following syntax tests whether the regression function is increasing:

```
. binsregtest y x w, deriv(1) nbins(20) testshaper(0)
Hypothesis tests based on binscatter estimates
Bin selection method: IMSE-optimal plug-in choice
Placement:
Derivative: 1
```

# of observations	1000
# of distinct values	1000
# of clusters	.
Bin selection:	
Degree of polynomial	0
# of smoothness constraints	0
# of bins	20

```
Shape Restriction Tests:
Degree: 3     # of smoothness constraints: 3
```

H0: inf mu >=	inf T	p value
0	-2.680	0.202

The null hypothesis here is that the infimum of the first-order derivative of the regression function is no less than 0. The output reports the test statistic, which is the infimum of the t-statistic over a sequence of evaluation points, and the corresponding simulation-based p-value.

The command `binsregtest` may implement many tests simultaneously (given the derivative of interest). For example,

```
. binsregtest y x w, nbins(20) testshaper(-2 0) testshapel(4) testmodelpoly(1) ///
>                    nsims(1000) simsgrid(30)
Hypothesis tests based on binscatter estimates
Bin selection method: IMSE-optimal plug-in choice
Placement:
Derivative: 0
```

# of observations	1000
# of distinct values	1000
# of clusters	.
Bin selection:	
Degree of polynomial	0
# of smoothness constraints	0
# of bins	20

```
Shape Restriction Tests:
Degree: 3     # of smoothness constraints: 3
```

H0: sup mu <=	sup T	p value
4	-1.683	1.000

H0: inf mu >=	inf T	p value

```
        -2          |    1.461            1.000
         0          |   -9.694            0.000
-------------------------------------------------

Model specification Tests:
Degree: 3     # of smoothness constraints: 3

-------------------------------------------------
H0: mu =            | sup |T|           p value
-------------------------------------------------
poly. degree  1     |    6.108            0.000
-------------------------------------------------
```

The above syntax tests three shape restrictions and one model specification (linearity), employing 1000 random draws from $\mathbf{N}_K$ and 30 evaluation points to evaluate the supremum/infimum in the simulation.

As already mentioned, the commands `binsreg` and `binsregtest` rely on data-driven bin selection procedures via the command `binsregselect` whenver the option `nbins()` is not employed by the user. Its basic syntax is as follows:

```
. binsregselect y x w
Bin selection for binscatter estimates
Method: IMSE-optimal: plug-in choice
Position: Quantile-spaced

-------------------------------------------------
          # of observations     1000
      # of distince values     1000
             # of clusters        .
         eff. sample size     1000
-------------------------------------------------
       Degree of polynomial        0
 # of smoothness constraint        0
-------------------------------------------------

-------------------------------------------------
    method       |  # of bins       df
-------------------------------------------------
   ROT-POLY      |     18           18
   ROT-REGUL     |     18           18
   ROT-UKNOT     |     18           18
      DPI        |     21           21
   DPI-UKNOT     |     21           21
-------------------------------------------------
```

The following choices of number of bins are reported: `ROT-POLY`, the rule-of-thumb (ROT) choice based on global polynomial estimation; `ROT-REGUL`, the ROT choice regularized as discussed in Section 2, or the user's choice specified in the option `nbinsrot()`; `ROT-UKNOT`, the ROT choice with unique knots; `DPI`, the direct plug-in (DPI) choice; and `DPI-UKNOT`, the DPI choice with unique knots.

The direct plug-in choice is implemented based on the rule-of-thumb choice, which can be set by users directly:

```
. binsregselect y x w, nbinsrot(20) binspos(es)
Bin selection for binscatter estimates
Method: IMSE-optimal: plug-in choice
Position: Evenly-spaced

-------------------------------------------------
          # of observations     1000
      # of distince values     1000
             # of clusters        .
         eff. sample size     1000
```

Degree of polynomial	0
# of smoothness constraint	0

method	# of bins	df
ROT-POLY	.	.
ROT-REGUL	20	20
ROT-UKNOT	20	20
DPI	22	22
DPI-UKNOT	22	22

Notice that in the example above an even-spaced, rather than quantile-spaced, binning scheme is selected via the option `binspos(es)`. The binning used in the commands `binsreg` and `binsregtest` may be adjusted similarly.

In addition, as illustrated above, the command `binsregselect` also provides a convenient option `savegrid()`, which can be used to generate the auxiliary dataset needed for parametric specification testing of user-chosen models via the command `binsregtest`. Specifically, the following command was (quietly) used above:

```
. binsregselect y x w, simsgrid(30) savegrid(output/parfitval) replace
Bin selection for binscatter estimates
Method: IMSE-optimal: plug-in choice
Position: Quantile-spaced
Output file: output/parfitval.dta
```

# of observations	1000
# of distince values	1000
# of clusters	.
eff. sample size	1000
Degree of polynomial	0
# of smoothness constraint	0

method	# of bins	df
ROT-POLY	18	18
ROT-REGUL	18	18
ROT-UKNOT	18	18
DPI	21	21
DPI-UKNOT	21	21

The resulting file, `parfitval.dta`, includes `x` and `w` as well as some other variables related to the binning scheme. The variable `x` contains a sequence of evalution points, in this case set to 30 within each bin via the option `simsgrid()`, and the values of `w` are set to zero on purpose (this is used to generate the fitting model correctly).

When an extremely large dataset is available, the data-driven procedures for selecting the binning scheme could be very time-consuming. In such a scenario, one could use a small (random) subsample to estimate the leading constants in the integrated mean squared error (IMSE) expansions, and then extrapolate the optimal number of bins to the full sample. The following code illustrates how this method is implemented:

```
. binsregselect y x w if t==0, useeffn(1000)
```

```
Bin selection for binscatter estimates
Method: IMSE-optimal: plug-in choice
Position: Quantile-spaced
```

# of observations	485
# of distince values	485
# of clusters	.
eff. sample size	1000
Degree of polynomial	0
# of smoothness constraint	0

method	# of bins	df
ROT-POLY	19	19
ROT-REGUL	19	19
ROT-UKNOT	19	19
DPI	25	25
DPI-UKNOT	25	25

In this example 485 observations with `t == 0` are used to compute the leading constants $\mathscr{B}_n(p,s,v)$ and $\mathscr{V}_n(p,s,v)$ in the IMSE expansion, but then the reported optimal numbers of bins are calculated based on the effective sample size specified in the option `useeffn()`. This method also applies to extrapolating the optimal number of bins to a smaller sample based on a larger one.

Finally, we emphasize another two features of the package Binsreg. First, all the three commands in this package implement mass point and degree of freedom checks as well as effective sample size adjustments by default. One may turn off all these checks and adjustments by setting `masspoints(off)` to speed up computation:

```
. qui binsreg y x w, masspoints(off)
```

Second, the main command `binsreg` is highly integrated with other companion commands. Specifically, `binsreg` can simultaneously implement binscatter plotting and hypothesis testing with the number of bins automatically selected via the commands `binsregtest` and `binsregselect`. For example,

```
. binsreg y x w, dots(0,0) line(3,3) ci(3,3) cb(3,3) polyreg(4) ///
>                testmodelpoly(1) testshapel(4)
Sorting dataset on x...
Note: This step is omitted if dataset already sorted by x.
Binscatter plot
Bin selection method: IMSE-optimal plug-in choice
Placement: Quantile-spaced
Derivative: 0
```

# of observations	1000
# of distinct values	1000
# of clusters	.
Bin selection:	
Degree of polynomial	0
# of smoothness constraints	0
# of bins	21

```
          |      p       s       df
----------+--------------------------
 dots     |      0       0       21
 line     |      3       3       24
 CI       |      3       3       24
 CB       |      3       3       24
 polyreg  |      4      NA        5
----------------------------------------
```

```
Hypothesis tests based on binscatter estimates
Bin selection method: IMSE-optimal plug-in choice
Placement: Quantile-spaced
Derivative: 0
```

```
-------------------------------------------------
# of observations                |        1000
# of distinct values             |        1000
# of clusters                    |           .
-------------------------------------------------
Bin selection:                   |
         Degree of polynomial    |           0
  # of smoothness constraints    |           0
                     # of bins    |          21
-------------------------------------------------
```

```
Shape Restriction Tests:
Degree: 3     # of smoothness constraints: 3
```

```
-----------------------------------------------------
H0: sup mu <=          |  sup T          p value
-----------------------------------------------------
         4             |  -1.920           1.000
-----------------------------------------------------
```

```
Model specification Tests:
Degree: 3     # of smoothness constraints: 3
```

```
-----------------------------------------------------
H0: mu =               |  sup |T|        p value
-----------------------------------------------------
poly. degree  1        |    6.503          0.000
-----------------------------------------------------
```

As a general rule, the implementations within the command `binsreg` is based on the binning scheme either specified by the user via the option `nbins()` or selected in a data-driven procedure given the choice of the degree and smoothness in the option `dots()`. Valid inference results require careful choice of binning or, more specifically, choice of J relative to p. It is recommended to use the data-driven method to select the IMSE-optimal number of bins for a given polynomial degree p, but then inference methods should be implemented with a higher order degree $p+q$, with $q \geq 1$, which corresponds to a simple application of robust bias-corrected inference (Calonico, Cattaneo, and Titiunik 2014; Calonico, Cattaneo, and Farrell 2018, 2019; Cattaneo, Farrell, and Feng 2018).

7 Conclusion

We introduced the `Stata` package Binsreg, which provides general-purpose software implementations of binscatter via three commands `binsreg`, `binsregtest`, and `binsregselect`. A companion `R` package with similar syntax and the same capabilities is also available.

8 Acknowledgments

We thank Michael Droste, John Friedman, Andreas Fuster, Paul Goldsmith-Pinkham, David Lucca, Xinwei Ma and Rocio Titiunik for helpful comments and discussions.

9 References

Calonico, S., M. D. Cattaneo, and M. H. Farrell. 2018. On the Effect of Bias Estimation on Coverage Accuracy in Nonparametric Inference. *Journal of the American Statistical Association* 113(522): 767–779.

———. 2019. Coverage Error Optimal Confidence Intervals for Local Polynomial Regression. arXiv:1808.01398 .

Calonico, S., M. D. Cattaneo, and R. Titiunik. 2014. Robust Nonparametric Confidence Intervals for Regression-Discontinuity Designs. *Econometrica* 82(6): 2295–2326.

Cattaneo, M. D., R. K. Crump, M. H. Farrell, and Y. Feng. 2019. On Binscatter. arXiv:1902.09608 .

Cattaneo, M. D., M. H. Farrell, and Y. Feng. 2018. Large Sample Properties of Partitioning-Based Estimators. arXiv:1804.04916 .

Chetty, R., J. N. Friedman, N. Hilger, E. Saez, D. W. Schanzenbach, and D. Yagan. 2011a. How Does Your Kindergarten Classroom Affect Your Earnings? Evidence from Project STAR. *Quarterly Journal of Economics* 126(4): 1593–1660.

Chetty, R., J. N. Friedman, T. Olsen, and L. Pistaferri. 2011b. Adjustment Costs, Firm Responses, and Micro vs. Macro Labor Supply Elasticities: Evidence from Danish Tax Records. *Quarterly Journal of Economics* 126(2): 749–804.

Chetty, R., A. Looney, and K. Kroft. 2009. Salience and Taxation: Theory and Evidence. *American Economic Review* 99(4): 1145–1177.

Chetty, R., and A. Szeidl. 2006. Marriage, Housing, and Portfolio Choice: A Test of Grossman-Laroque. Working Paper, UC-Berkeley .

Starr, E., and B. Goldfarb. 2018. A Binned Scatterplot is Worth a Hundred Regressions: Diffusing a Simple Tool to Make Empirical Research Easier and Better. SSRN Working paper No. 3257345 .

Stepner, M. 2014. Binned Scatterplots: Introducing -binscatter- and Exploring its Applications. 2014 Stata Conference 4, Stata Users Group .

About the Authors

Matias D. Cattaneo is a Professor of Economics and a Professor of Statistics at the University of Michigan.

Richard K. Crump is a Vice President and the Function Head of the Capital Markets Function at the Federal Reserve Bank of New York.

Max H. Farrell is an Associate Professor of Statistics and Econometrics at the University of Chicago Booth School of Business.

Yingjie Feng in a Ph.D. candidate in Economics at the University of Michigan.