The Stata Journal (2016)
16, Number 2, pp. 331–367

Inference in regression discontinuity designs under local randomization

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

Rocío Titiunik
University of Michigan
Ann Arbor, MI
titiunik@umich.edu

Gonzalo Vazquez-Bare
University of Michigan
Ann Arbor, MI
gvazquez@umich.edu

Abstract. We introduce the `rdlocrand` package, which contains four commands to conduct finite-sample inference in regression discontinuity (RD) designs under a local randomization assumption, following the framework and methods proposed in Cattaneo, Frandsen, and Titiunik (2015, *Journal of Causal Inference* 3: 1–24) and Cattaneo, Titiunik, and Vazquez-Bare (2016, Working Paper, University of Michigan, http://www-personal.umich.edu/~titiunik/papers/CattaneoTitiunikVazquezBare2015_wp.pdf). Assuming a known assignment mechanism for units close to the RD cutoff, these functions implement a variety of procedures based on randomization inference techniques. First, the `rdrandinf` command uses randomization methods to conduct point estimation, hypothesis testing, and confidence interval estimation under different assumptions. Second, the `rdwinselect` command uses finite-sample methods to select a window near the cutoff where the assumption of randomized treatment assignment is most plausible. Third, the `rdsensitivity` command uses randomization techniques to conduct a sequence of hypothesis tests for different windows around the RD cutoff, which can be used to assess the sensitivity of the methods and to construct confidence intervals by inversion. Finally, the `rdrbounds` command implements Rosenbaum (2002, *Observational Studies* [Springer]) sensitivity bounds for the context of RD designs under local randomization. Companion R functions with the same syntax and capabilities are also provided.

Keywords: st0435, rdrandinf, rdwinselect, rdsensitivity, rdrbounds, regression discontinuity designs, quasi-experimental techniques, causal inference, randomization inference, finite-sample methods, Fisher's exact p-values, Neyman's repeated sampling approach

1 Introduction

Conventional inference methods for regression discontinuity (RD) designs use nonparametric local polynomial techniques, rely on large-sample approximations, and provide estimators and inference procedures for the (super) population under repeated sampling assumptions. For example, see Hahn, Todd, and van der Klaauw (2001) and Calonico, Cattaneo, and Titiunik (2014b) for treatment-effect estimation and inference and McCrary (2008) and Cattaneo, Jansson, and Ma (2016b) for manipulation testing. Reviews and further references on flexible parametric and nonparametric approaches to analyzing RD designs are given by Imbens and Lemieux (2008), Lee and Lemieux (2010), Calonico, Cattaneo, and Titiunik (2015a), Keele and Titiunik (2015), and Skovron and

Titiunik (2015). These approaches rely on smoothness assumptions leading to nonparametric identification of, and valid inference procedures for, RD treatment effects at the cutoff in the population.

An alternative approach to analyzing RD designs relies on the idea of local randomization, which postulates that treatment assignment may be regarded as (or at least approximated by) a known randomization mechanism near the RD cutoff (Lee 2008). In other words, in this approach, the units closest to the cutoff are viewed as being part of a local randomized experiment. Building on this intuitive idea, Cattaneo, Frandsen, and Titiunik (2015) and Cattaneo, Titiunik, and Vazquez-Bare (2016) develop a framework and methodological tools to analyze RD designs under a local randomization assumption. The approach is based on methods from the classical literature on the analysis of experiments (Rosenbaum 2002, 2010; Imbens and Rubin 2015), where the (possibly transformed) potential outcomes are regarded as fixed, and inference is based on the randomization distribution of the treatment assignment. This alternative inference approach is useful in many cases, including when the dataset of interest is small, the running variable is not continuous, matching techniques are used, or multiple RD cutoffs are analyzed. For a geographic RD example applying some of these ideas, see Keele, Titiunik, and Zubizarreta (2015).

In this article, we introduce the `rdlocrand` package to conduct finite-sample inference in RD designs using randomization inference and related techniques under a local randomization assumption. This package contains the following four commands:

1. `rdrandinf`. This command implements several methods from the literature on the analysis of experiments, including Fisher's exact p-values and tests, Neyman's repeated sampling inference, and related methods. These methods rely on finite-sample exact techniques (that is, randomization inference methods) and on particular large-sample approximations. The command offers point estimation, hypothesis testing, and confidence interval procedures under different assumptions, allowing in particular for regression-based outcome transformations.

2. `rdwinselect`. This command uses finite-sample (but also large-sample) methods to select a window near the cutoff where the local randomization assumption is most plausible, under some intuitive conditions. This command is called by the `rdrandinf` command to select a data-driven window around the cutoff where inference is conducted, but `rdwinselect` can also be used as a stand-alone command for analysis and falsification of RD designs under a local randomization assumption.

3. `rdsensitivity`. This command uses randomization inference methods to conduct a sequence of hypothesis tests for different windows around the RD cutoff, which can be used to assess the sensitivity of the methods and to construct confidence intervals by inversion.

4. `rdrbounds`. This command implements randomization-based sensitivity analysis (Rosenbaum 2002) in the context of RD designs under local randomization.

Altogether, the `rdlocrand` package offers a complete and systematic analysis of RD designs under a local randomization assumption and some related conditions. These finite-sample methods can be used for empirical analysis of RD designs under a set of assumptions that are different from the conventional assumptions in the nonparametric literature (for example, randomization-based methods allow for discrete running variables). In addition, the procedures discussed here can be used as a robustness check on the conventional inference methods that use large-sample approximations.

In the remainder of this article, we provide methodological, practical, and empirical introductions to these commands. First, we briefly review the main theoretical concepts underlying the methods implemented (section 2). Second, we provide syntax and a brief explanation of the functionalities of each of the four commands (sections 3, 4, 5, and 6). Finally, we present a detailed empirical analysis replicating the empirical results originally reported in Cattaneo, Frandsen, and Titiunik (2015), showing some of the main options implemented by these commands (section 7). We briefly conclude in section 8.

For related Stata (and R) commands (`rdrobust`, `rdbwselect`, and `rdplot`) providing graphical presentation, estimation, and inference in RD designs using local polynomial techniques, see Calonico, Cattaneo, and Titiunik (2014a, 2015b). In addition, for Stata (and R) commands (`rddensity` and `rdbwdensity`) implementing manipulation testing based on discontinuity in density using local polynomial techniques, see Cattaneo, Jansson, and Ma (2016a).

2 Overview of methods

This section briefly describes the statistical framework and methods for RD designs under local randomization. The framework and methods considered here were introduced in Cattaneo, Frandsen, and Titiunik (2015) and Cattaneo, Titiunik, and Vazquez-Bare (2016)—we refer the reader to these references for further details. For a review on randomization inference and related experimental and quasi-experimental methods, see Rosenbaum (2002, 2010) and Imbens and Rubin (2015).

2.1 Statistical framework

We start by introducing some notation. The sample consists of n units indexed by $i = 1, \ldots, n$. The running variable, score, or index is denoted by R_i, and the treatment indicator is D_i. In a sharp RD design, $D_i = \mathbb{1}(R_i \geq \overline{r})$, where $\overline{r}$ is a fixed and known cutoff and $\mathbb{1}(\cdot)$ is the indicator function. Let $\mathbf{D}$ be the $n \times 1$ vector collecting the observed treatment assignment for the n observations, and analogously for $\mathbf{R}$; that is, $\mathbf{D} = (D_1, D_2, \ldots, D_n)'$ and $\mathbf{R} = (R_1, R_2, \ldots, R_n)'$. The potential outcome of unit i under a vector of scores $\mathbf{r}$ and a vector of treatment assignments $\mathbf{d}$ is $y_i(\mathbf{d}, \mathbf{r})$ taking values in a set $\mathcal{Y}$. For example, $\mathcal{Y} = \{0, 1\}$ if the potential outcomes are binary, $\mathcal{Y} = [0, \infty)$ if they are nonnegative, and so on. The support of $\mathbf{D}$ is denoted by $\mathsf{supp}(\mathbf{D}) := \Omega$ and, similarly, $\mathsf{supp}(\mathbf{R}) := \mathcal{R}$. We collect these potential outcomes in a

vector $\mathbf{y}(\mathbf{d},\mathbf{r}) = \{y_1(\mathbf{d},\mathbf{r}), y_2(\mathbf{d},\mathbf{r}), \ldots, y_n(\mathbf{d},\mathbf{r})\}'$. Finally, the vector of observed outcomes is $\mathbf{Y} = \mathbf{y}(\mathbf{D},\mathbf{R})$, which is a random variable by virtue of $\mathbf{R}$ and, as a consequence, $\mathbf{D}$ being random. Throughout, we use lowercase to denote fixed, nonrandom variables and uppercase to denote random variables.

The local randomization framework for RD designs is characterized by two features: i) a known randomization (or treatment assignment) mechanism in some region or window around the cutoff and ii) an exclusion restriction on the (transformed) potential outcomes. More precisely, the first condition is related to the existence of a window $W_0 = [\overline{r} - w, \overline{r} + w]$, $w > 0$, around the cutoff in which, as in a randomized controlled experiment, treatment assignment of units with $R_i \in W_0$ follows a known distribution: the probability law $\mathbb{P}(\mathbf{D}_{W_0} = \mathbf{d})$ is known, where $\mathbf{D}_{W_0}$ is the subvector of $\mathbf{D}$ corresponding to the observations with $R_i \in W_0$ and $\mathbf{d} \in \{0,1\}^{n_{W_0}}$ with $n_{W_0} = \sum_{i=1}^{n} \mathbb{1}(R_i \in W_0)$ denoting the number of units in W_0.

For implementation purposes, the `rdlocrand` command considers two types of randomization mechanisms. In the first one, called fixed-margins randomization or complete randomization, each treatment assignment chooses a given number of treated units among the n_{W_0} units in W_0. Thus, in applications, the probability distribution of $\mathbf{D}_{W_0}$ is given by

$$\mathbb{P}(\mathbf{D}_{W_0} = \mathbf{d}) = \binom{n_{W_0}}{m_{W_0}}^{-1} \qquad \forall\, \mathbf{d} \in \Omega_0$$

where $m_{W_0} = \sum_{i=1}^{n} \mathbb{1}(R_i \in W_0) D_i$ is the number of treated units and Ω_0 is the set of n_{W_0}-dimensional vectors with exactly m_{W_0} ones. In the second assignment mechanism, units are assigned to treatment following independent Bernoulli trials with some probability $q \in (0,1)$, so that

$$\mathbb{P}(\mathbf{D}_{W_0} = \mathbf{d}) = \prod_{i=1}^{n_{W_0}} q^{d_i}(1-q)^{1-d_i} \qquad \forall\, \mathbf{d} \in \Omega_0$$

where now $\Omega_0 = \{\mathbf{d} \in \{0,1\}^{n_{W_0}}\}$. The parameter q is often unknown but can be easily estimated, for example, as the proportion of treated units in the window; that is, $\widehat{q} = m_{W_0}/n_{W_0}$.

The second feature of the local randomization framework is a model or, more precisely, an exclusion restriction for the (transformed) potential outcomes. The idea is that the potential outcomes can be transformed to eliminate the direct dependence between potential outcomes and the running variable, that is, that there exists some transformation $\phi(\cdot)$ of the potential outcomes such that

$$\phi(\mathbf{y}(\mathbf{d},\mathbf{r}),\mathbf{d},\mathbf{r}) = \widetilde{\mathbf{y}}(\mathbf{d}_{W_0}) \quad \forall\, \mathbf{r} \in \mathcal{R}$$

In practice, this assumption is likely to be more plausible for units within the window W_0. In the end, the goal is to transform the (potential) outcomes so that $\widetilde{y}_i(\mathbf{d},\mathbf{r}) = \widetilde{y}_i(\mathbf{d}_{W_0})$ for all units with score in W_0, so their potential outcomes are not a function of the score except through the treatment indicator. In some applications, the transformation is not needed [that is, $\phi(\cdot)$ is the identity function and $y_i(\mathbf{d},\mathbf{r}) = y_i(\mathbf{d})$ directly], but

in other cases, the researcher may want to incorporate some parametric relationship. This restriction, for example, could reflect the standard practice in the RD literature of fitting a lower-order polynomial regression at each side of the cutoff.

For the outcome transformation, `rdlocrand` considers polynomial transformations with different centerings. Specifically, all the commands in the `rdlocrand` package allow for a transformation of (potential) outcomes based on a polynomial model of order p of the form

$$y_i(\mathbf{d},\mathbf{r}) = \begin{cases} \alpha_i(\mathbf{d}_{W_0}) + \beta_{01}(r_i - r_0) + \beta_{02}(r_i - r_0)^2 + \cdots + \beta_{0p}(r_i - r_0)^p & \text{if } d_i = 0 \\ \alpha_i(\mathbf{d}_{W_0}) + \beta_{11}(r_i - r_1) + \beta_{12}(r_i - r_1)^2 + \cdots + \beta_{1p}(r_i - r_1)^p & \text{if } d_i = 1 \end{cases}$$

for all i such that $R_i \in W_0$, where r_1 and r_0 are two evaluation points set in advance; natural choices are i) the cutoff point ($r_1 = r_0 = \overline{r}$) and ii) the center of the control ($r_0 = \overline{r} - w/2$) and treatment ($r_1 = \overline{r} + w/2$) regions within W_0. Of course, in the absence of a polynomial transformation (that is, $\beta_{0j} = 0 = \beta_{1j}$ for $j = 1, 2, \ldots, p$), we obtain the natural exclusion restriction

$$y_i(\mathbf{d},\mathbf{r}) = \alpha_i(\mathbf{d}_{W_0}) =: \widetilde{y}_i(\mathbf{d}_{W_0})$$

for all units with their score inside W_0. The polynomial transformation assumption allows for the potential outcomes to have some direct dependence on the score, although this assumption is of course strong and very specific. In this case, the transformed potential outcomes are

$$\widetilde{y}_i(\mathbf{d}_{W_0}) := \begin{cases} y_i(\mathbf{d},\mathbf{r}) - \beta_{01}(r_i - r_0) - \cdots - \beta_{0p}(r_i - r_0)^p = \alpha_i(\mathbf{d}_{W_0}) & \text{if } d_i = 0 \\ y_i(\mathbf{d},\mathbf{r}) - \beta_{11}(r_i - r_1) - \cdots - \beta_{1p}(r_i - r_1)^p = \alpha_i(\mathbf{d}_{W_0}) & \text{if } d_i = 1 \end{cases}$$

for all i such that $R_i \in W_0$. Thus the transformed potential outcomes isolate the portion of the potential outcome that is related to the treatment but unrelated to the particular value taken by the running variable. The coefficients β_{dj} ($d = 0, 1$ and $j = 1, 2, \ldots, p$) are analytically computed by least squares.

In the model above, the transformed potential outcomes depend only on $\mathbf{d}_{W_0}$, and the probability law of $\mathbf{D}_{W_0}$ is known for all units with score in W_0. Therefore, among other possibilities, randomization inference can be used for hypothesis testing and confidence interval construction. As discussed below, we will need to further restrict the model (that is, impose the so-called stable unit treatment value assumption, or SUTVA). Before discussing how to perform inference in this setting, we describe our window-selection method.

2.2 Window selection

Cattaneo, Frandsen, and Titiunik (2015) propose a data-driven method to find the window W_0 where the local randomization assumption is assumed to hold. The idea is that because treatment assignment is "as if random" inside the window, the distribution of preintervention covariates and postintervention unaffected-by-treatment outcomes

should be the same between treated and control units. This observation is closely related to the ideas underlying balance tests in the experimental and nonexperimental literature. For the procedure to be useful, the distribution of these covariates for control and treatment units should be unaffected by the treatment within W_0 but should be affected by the treatment outside the window.

To describe the approach, we will let $\mathbf{x}$ be the $n \times k$ matrix collecting the k covariates and, for an arbitrary window W, $\mathbf{x}_W$ be the subvector corresponding to units with scores inside the window W. Then, the window-selection algorithm is the following:

1. Set an initial "small" window, $\widehat{W}_1$.
2. For each of the k covariates, conduct a test of the null hypothesis of no effect of the treatment on the covariate using some test-statistic $\mathcal{T}(\mathbf{x}_{\widehat{W}_1}, \mathbf{R}_{\widehat{W}_1})$. Take the minimum p-value from the k tests.
3. If the minimum p-value obtained in step 2, p_1, is less than some prespecified level—for example, 0.15—the initial window was too large;[1] decrease the initial window and start over. If the initial window cannot be decreased (for example, because a smaller window would contain too few data points), conclude that W_0 cannot be found. Otherwise, if $p_1 \geq 0.15$, choose a larger window $\widehat{W}_2 \supset \widehat{W}_1$, and go back to step 2 to calculate p_2. Repeat the process until the minimum p-value is less than 0.15. The selected window is the largest window such that the minimum p-value is larger than or equal to 0.15 in that window and in all windows contained in it.

The resulting window, $\widehat{W}$, is the estimate of W_0. The idea of this algorithm is to choose the largest window (or one of the largest windows) in which all the covariates are balanced. This window-selection procedure is implemented by `rdwinselect` using both randomization inference and large-sample methods discussed next. We emphasize that although this procedure performs multiple hypothesis tests, we do not use multiple testing adjustments because the goal is to be conservative and reject the null often so that the chosen window is small.

2.3 Randomization inference

The framework in section 2.1 provides all the necessary information to test the sharp null hypothesis of no treatment effect using randomization inference methods. More precisely, consider the following hypothesis:

$$\mathrm{H}_0: \quad \alpha_i(\mathbf{d}_{W_0}) = \alpha_i(\mathbf{d}'_{W_0}) \qquad \forall i : R_i \in W_0 \quad \text{and} \quad \forall \mathbf{d}_{W_0}, \mathbf{d}'_{W_0}$$

This hypothesis is usually known as the sharp null hypothesis of no treatment effect or Fisher's null hypothesis. In the randomization inference literature, a hypothesis is said

1. In some cases, this could also happen if the initial window was too small because balance tests may be unreliable in these cases. We recommend starting with initial windows of at least 10 observations at each side of the cutoff.

to be "sharp" if it allows the researcher to impute all missing potential outcomes. Under this hypothesis, we have $\widetilde{y}_i(\mathbf{d}_{W_0}) = \alpha_i(\mathbf{d}_{W_0})$ for all units inside the window W_0; thus all the potential outcomes can be observed. Collecting all observed transformed outcomes for units in W_0 in the vector $\widetilde{\mathbf{Y}}_{W_0}(\mathbf{D}_{W_0})$ and all $\alpha_i(\mathbf{d}_{W_0})$ in W_0 in the vector $\boldsymbol{\alpha}^0_{W_0}$, under H_0, we have that $\widetilde{\mathbf{Y}}_{W_0} = \boldsymbol{\alpha}^0_{W_0}$. But $\boldsymbol{\alpha}^0_{W_0}$ is constant under all realizations of $\mathbf{D}$. Therefore, any test-statistic $\mathcal{T}(\mathbf{D}_{W_0}, \widetilde{\mathbf{Y}}_{W_0})$ satisfies $\mathcal{T}(\mathbf{D}_{W_0}, \widetilde{\mathbf{Y}}_{W_0}) = \mathcal{T}(\mathbf{D}_{W_0}, \boldsymbol{\alpha}^0_{W_0})$ under the sharp null hypothesis, implying that its null distribution is known—because the only randomness in $\mathcal{T}(\mathbf{D}_{W_0}, \boldsymbol{\alpha}^0_{W_0})$ originates in the treatment assignment mechanism, $\mathbf{D}$, whose distribution is assumed to be known.

An exact p-value for an observed value of the statistic $\mathcal{T}_{\text{obs}}$ can be easily computed as

$$\mathbb{P}\left\{\mathcal{T}\left(\mathbf{D}_{W_0}, \widetilde{\mathbf{Y}}_{W_0}\right) \geq \mathcal{T}_{\text{obs}}\right\} = \sum_{\mathbf{d}\in\Omega_0} \mathbb{1}\left\{\mathcal{T}\left(\mathbf{D}_{W_0}, \widetilde{\mathbf{Y}}_{W_0}\right) \geq \mathcal{T}_{\text{obs}}\right\} \mathbb{P}\left(\mathbf{D}_{W_0} = \mathbf{d}\right)$$

where $\mathbb{P}(\mathbf{D}_{W_0} = \mathbf{d})$ follows a known distribution, as mentioned above. In practice, the cardinality of Ω_0 will be very large even for relatively small sample sizes; thus the p-value is approximated by drawing treatment assignment vectors from Ω_0 at random. The larger the number of repetitions, the more precise the approximation will be.

A common simplifying assumption in this context is the SUTVA; that is, unit i's (transformed) potential outcome depends only on unit i's score and treatment assignment; that is, $\widetilde{y}_i(\mathbf{d}, \mathbf{r}) = \widetilde{y}_i(d_i, r_i)$ for all i such that $R_i \in W_0$. Although the main ideas and results discussed here do not require this condition, in practice, the outcome transformation model may be hard or impossible to implement without restrictions on the degree of interference between units. Thus our implementations effectively use SUTVA whenever a transformation is used. Section 2.5 discusses ways to relax this assumption when interference between units is suspected. SUTVA implies that each unit has exactly two transformed potential outcomes, $\widetilde{y}_i(1)$ and $\widetilde{y}_i(0)$.

After imposing SUTVA, we can apply the same reasoning above to test any sharp null hypothesis. For example, if one is willing to assume that the treatment effect is an additive constant τ, one can test whether $\tau = \tau_0$:

$$\mathrm{H}_0\colon \quad \alpha_i(1) = \alpha_i(0) + \tau_0 \quad \forall i : R_i \in W_0$$

The idea is that in this case, $\widetilde{y}_i = \widetilde{y}_i(0) + \tau_0 D_i$; hence, $\widetilde{y}_i - \tau_0 D_i$ does not vary with D_i. The null hypothesis is then tested using $\mathcal{T}(\mathbf{D}_{W_0}, \widetilde{\mathbf{Y}}_{W_0} - \tau\mathbf{D}_{W_0})$.

For the rest of this article, we impose SUTVA to simplify the discussion and presentation, except where we note explicitly otherwise.

2.4 Statistics and confidence intervals

As we will illustrate in section 7, the commands presented implement up to four different statistics: difference in means (DM), Kolmogorov–Smirnov (KS), Wilcoxon rank

sum (RS), and Hotelling's T^2 statistics. The last statistic is available only for window selection (`rdwinselect`) but not for RD inference (`rdrandinf`) or sensitivity analysis (`rdsensitivity`, `rdrbounds`). The formulas below are written in terms of the outcome variable (adjusted by the null value τ_0 when required); when implementing `rdwinselect`, one applies the same formulas using each covariate x_{ij} $(j = 1, \ldots, k)$ as an outcome instead of $\widetilde{y}_i - \tau_0 D_i$.

In addition, under appropriate assumptions, different types of confidence intervals can be constructed using randomization inference methods under the local randomization assumption. In this section, we review two methods implemented in the commands `rdsensitivity` (under a treatment-effect model) and `rdrandinf`, allowing for the presence of interference.

DM

The DM statistic is calculated as

$$\mathcal{T}_{\text{DM}} = \frac{1}{m_{W_0}} \sum_{i:R_i \in W_0} \left(\widetilde{Y}_i - \tau_0 D_i\right) D_i - \frac{1}{n_{W_0} - m_{W_0}} \sum_{i:R_i \in W_0} \left(\widetilde{Y}_i - \tau_0 D_i\right) (1 - D_i)$$

with $m_{W_0} = \sum_{i:R_i \in W_0} D_i$ and $n_{W_0} - m_{W_0} = \sum_{i:R_i \in W_0} (1 - D_i)$ as defined above.

Unlike the other estimators that will be discussed, the DM has the appealing feature that it provides a point estimate of the average treatment effect (ATE) without imposing any restrictions on how the treatment effect varies across units. Moreover, one can easily show that this statistic is unbiased for the ATE when the treatment assignment follows a fixed-margin randomization scheme or Bernoulli trials. Hence, $\mathcal{T}_{\text{DM}}$ plays the double role of test statistic and point estimate of the ATE. One should keep in mind, however, that the null hypothesis being tested is that the treatment effect is zero for all units, which is much stronger than the hypothesis that the ATE is zero.

In terms of implementation, `rdlocrand` calculates $\mathcal{T}_{\text{DM}}$ as the estimate of a fully interacted pth-order polynomial regression using observations inside the window, which is simply a regression on the indicator of being above the cutoff D_i and a polynomial on R_i allowed to differ for units above and below the cutoff. Note that this is equivalent to running two separate regressions of the outcome on a polynomial of R_i, one above and the other below the cutoff, and taking the difference in the intercepts. In this case, $p = 0$ would correspond to a simple DM (that is, no model transformation).

KS

The KS statistic is obtained as

$$\mathcal{T}_{\text{KS}} = \sup_{y \in \mathcal{Y}} \left| \widehat{F}(y|D = 1) - \widehat{F}(y|D = 0) \right|$$

where $\widehat{F}(y|D = 1)$ and $\widehat{F}(y|D = 0)$ are estimates of the distribution function of the transformed outcome for the treated and control units, respectively. When $p = 0$, that

is, $Y_i = \widetilde{Y}_i$ (that is, there is no transformation), this statistic is calculated using the `ksmirnov` Stata command directly. Otherwise, when $p > 0$, the statistic is obtained in two steps. First, $Y_i - \tau_0 D_i$ is regressed on a pth-degree polynomial separately for treated and control units; for each regression, the estimate of the intercept is added to the residuals. The resulting values are the transformed outcomes for treated and controls, $\widetilde{Y}_i$. Second, the KS statistic is constructed using $\widetilde{Y}_i - \tau_0 D_i$.

Wilcoxon RS

This statistic is the studentized version of the Wilcoxon RS statistic, computed as

$$\mathcal{T}_{\text{RS}} = \frac{T - \mathbb{E}(T)}{\sqrt{\mathbb{V}(T)}}$$

where T is the sum of ranks for the control group (Wilcoxon's statistic) and $\mathbb{E}(T)$ and $\mathbb{V}(T)$ are the expectation and variance of the Wilcoxon RS statistic. This is calculated using the `ranksum` command. As for the KS statistic, when $p > 0$, the statistic is calculated using the transformed outcomes, $\widetilde{Y}_i$, as mentioned above.

Hotelling's $\mathbf{T^2}$

The window-selection procedure described above involves performing k hypothesis tests, one for each covariate, and choosing the minimum between the k resulting p-values. The `rdwinselect` command also allows the user to perform a joint test using Hotelling's T^2 statistic, which is based on the DM between the whole vector of covariates, instead of each mean individually. The statistic is obtained as

$$\mathcal{T}_{\text{H}} = (\overline{\mathbf{x}}_1 - \overline{\mathbf{x}}_0)\mathbf{S}^{-1}(\overline{\mathbf{x}}_1 - \overline{\mathbf{x}}_0)'$$

where $\overline{\mathbf{x}}_0$ and $\overline{\mathbf{x}}_1$ correspond to the sample means of covariates for control and treatment units, respectively, and $\mathbf{S}$ is the estimated covariance matrix—where all estimates are computed using the units inside the given window. This statistic is obtained using the `hotelling` command. Note that in this case, only one p-value is obtained for each window.

Confidence intervals under parametric model

Once a parametric model for treatment effects is specified, a finite-sample valid confidence interval can be constructed using randomization inference methods. The idea is to invert a sequence of hypothesis tests and to report all the values in the parameter space for which the (sharp) null hypothesis of no treatment effect is not rejected at a given level of significance. This method, which is standard in the literature, can be easily obtained using the `rdrandinf` command or the sensitivity analysis command `rdsensitivity` with the option `ci(·)`, as we discuss and illustrate below.

Confidence intervals under arbitrary interference

The presence of interference is immaterial when testing the sharp null of no effect, and the methods described in section 2.3 can still be applied without modification. Point estimation, however, becomes infeasible because when arbitrary interference is allowed, the definition of a treatment effect is not straightforward. In this case, Rosenbaum (2007) suggests a way to construct confidence intervals for a particular measure of the benefits of the treatment. The `rdrandinf` command implements this procedure via the `interfci()` option, which we briefly summarize here.

Define a placebo trial (or uniformity trial) as a trial in which units are randomly divided into two groups, but then treatment is withheld from all units. Because, by construction, nobody receives treatment, the transformed outcomes in the uniformity trial, $\widetilde{\mathbf{Y}}_{\mathtt{U},W_0}$, satisfy $\widetilde{\mathbf{Y}}_{\mathtt{U},W_0} = \boldsymbol{\alpha}_{W_0}$. If $\mathcal{T}_{\mathtt{U}} := \mathcal{T}(\mathbf{D}_{W_0}, \widetilde{\mathbf{Y}}_{\mathtt{U},W_0})$ is the value of the statistic in this placebo trial, then even though the value of $\mathcal{T}_{\mathtt{U}}$ is unobservable, its distribution is known because in a placebo trial, the sharp null hypothesis holds. Consider the magnitude $\Delta = \mathcal{T} - \mathcal{T}_{\mathtt{U}}$, which measures the difference between the statistic under the experiment under consideration and under the placebo experiment. If the treatment has no effect, $\Delta = 0$. For instance, if $\mathcal{T}$ is the DM, then

$$\Delta = \frac{\sum\limits_{i\in\mathcal{I}_0}\left(\widetilde{Y}_i - \widetilde{Y}_{\mathtt{U}i}\right)D_i}{\sum\limits_{i\in\mathcal{I}_0} D_i} - \frac{\sum\limits_{i\in\mathcal{I}_0}\left(\widetilde{Y}_i - \widetilde{Y}_{\mathtt{U}i}\right)(1-D_i)}{\sum\limits_{i\in\mathcal{I}_0}(1-D_i)}$$

where $\mathcal{I}_0 = \{i : R_i \in W_0\}$. For this choice of $\mathcal{T}$, Δ is the difference between how much the treated and control groups deviate (on average) from the zero-effect case. In other words, Δ measures how much bigger the treatment effect is for the treated group compared with the control group.

A confidence interval for Δ is constructed in the following way. Let κ_1 and κ_2 be some constants satisfying $\kappa_2 < \kappa_1$. Then, it is easy to see that

$$\mathbb{P}\left(\mathcal{T} - \kappa_1 \leq \Delta \leq \mathcal{T} - \kappa_2\right) = \mathbb{P}(\kappa_2 \leq \mathcal{T}_{\mathtt{u}} \leq \kappa_1)$$

Hence, if κ_1 and κ_2 are chosen to be, say, the $\alpha/2$ and $1-\alpha/2$ quantiles of $\mathcal{T}_{\mathtt{u}}$ for some level α, it follows that $\Delta \in [\mathcal{T} - \kappa_1, \mathcal{T} - \kappa_2]$ with probability $1-\alpha$. In practice, the values for κ_1 and κ_2 can be recovered from the randomization distribution of $\mathcal{T}_{\mathtt{u}}$, and $\mathcal{T}$ is replaced by its observed value. (Of course, the level of the confidence interval may not be exactly $1-\alpha$ because the distribution of $\mathcal{T}_{\mathtt{u}}$ is discrete, but the quantiles can be chosen so that the level is at least $1-\alpha$.)

2.5 Sensitivity analysis

We also provide two alternative approaches to sensitivity analysis using randomization inference methods under the local randomization assumption. These methods are implemented in the `rdsensitivity` command for sensitivity to model specification and window length and in the `rdrbounds` command for sensitivity to randomization mechanism and selection bias.

Sensitivity to parametric model and window length

Although the randomization inference framework used in this article suggests an intuitively and appealing window selector, in practice, the user may want to explore how robust the results are to different window lengths and parametric models. The `rdsensitivity` command calculates the p-values for the desired hypothesis test over a range of window lengths, given the parametric model chosen by the researcher. For each window, this command considers a range of treatment effects under the null hypothesis, a feature that offers a way of analyzing the sensitivity of the "point estimates" by carrying out a family of hypothesis tests within and across windows. These methods may also be used to analyze the sensitivity to the specific parametric model by simply using this command with different models and then comparing the results.

In addition, for a given window, the above procedure can be used directly to obtain confidence intervals via inversion of the corresponding hypothesis test, as we mentioned previously. We illustrate empirically all of these features below.

Misspecification of the randomization mechanism: Sensitivity to unobserved confounders

This approach is quite standard in the randomization inference literature (Rosenbaum 2002); thus we include it here as an additional robustness check. The only substantive modification relative to the conventional approach is the explicit use of the different windows around the cutoff, which provides an additional dimension for sensitivity analysis.

Assume that, inside the window, the randomization mechanism follows a Bernoulli experiment where the individual probability of treatment is

$$\mathbb{P}(D_i = 1) = q_i = \frac{\exp(\gamma U_i)}{1 + \exp(\gamma)} \tag{1}$$

and U_i is an unobserved binary variable. This gives

$$q_i = q_L = \frac{1}{1 + \exp(\gamma)} \equiv \frac{1}{1 + \Gamma} \qquad \text{and} \qquad q_i = q_H = \frac{\exp(\gamma)}{1 + \exp(\gamma)} \equiv \frac{\Gamma}{1 + \Gamma}$$

for units with $U_i = 0$ and $U_i = 1$, respectively, where $\Gamma \equiv \exp(\gamma)$. It follows that the odds ratio for units i and j satisfies

$$\frac{1}{\Gamma} \leq \frac{q_i/(1 - q_i)}{q_j/(1 - q_j)} \leq \Gamma$$

or, equivalently,

$$\left| \log\left(\frac{q_i}{1 - q_i}\right) - \log\left(\frac{q_j}{1 - q_j}\right) \right| \leq \gamma$$

In a completely randomized experiment, $\gamma = 0$, so all units have the same probability of treatment. Thus γ measures the degree of departure from a randomized experiment

(under the assumptions imposed). The idea of Rosenbaum's sensitivity analysis is to use (1) to see how the randomization p-value varies over a range of values for γ (or Γ).

To make this method operational, one must assign the unobservable U_i to units in the sample in a way that ensures that the distribution of the statistic of interest can be bounded by two distributions that can be obtained from the data. To this end, following Rosenbaum (2002), the `rdrbounds` command sorts the outcome in decreasing order and performs a search over the sets $\mathcal{U}^+$ and $\mathcal{U}^-$, where $\mathcal{U}^+$ contains all vectors with ones in the first ℓ positions and zeros in the remaining positions, and $\mathcal{U}^-$ contains vectors with zeros in the first ℓ positions and ones in the remaining ones.

2.6 Extension to fuzzy RD designs

In the so-called fuzzy RD design, treatment is no longer deterministically assigned based on the running variable; hence, compliance is imperfect among units below and above the cutoff. This implies that treatment assignment and treatment status can in principle be different. To adapt the previous notation to this context, let $\mathbf{d}(\mathbf{r})$ be the vector of potential treatment assignments, so that the observed treatment status is given by $\mathbf{D} = \mathbf{d}(\mathbf{R})$. Treatment assignment for units i is now given by $Z_i = \mathbb{1}(R_i \geq \overline{r})$, and the vector of treatment assignments is $\mathbf{Z}_{W_0}$ restricted to the units within the window W_0 (using the same notation introduced previously). The usual way to handle fuzzy designs is to use the vector $\mathbf{Z}$ as an instrument for $\mathbf{D}$. We assume that the distribution of $\mathbf{Z}$ is known. Imbens and Rosenbaum (2005) discuss randomization inference methods in the context of IV models.

Under the null hypothesis that the treatment effect is $\tau = \tau_0$, the adjusted responses $\widetilde{y}_i - \tau_0 D_i$ are fixed and unrelated to the instrument. Therefore, the distribution of any statistic $\mathcal{T}(\mathbf{Z}_{W_0}, \widetilde{\mathbf{Y}}_{W_0} - \tau_0 \mathbf{D}_{W_0})$ is known and can be used to perform inference in the same way as it was for sharp designs. The statistic used in this case, which we denote $\mathcal{T}_{\text{AR}}$,[2] is the difference in the adjusted transformed responses between units assigned to treatment and control

$$\mathcal{T}_{\text{AR}} = \frac{\sum\limits_{i \in \mathcal{I}_0} \left(\widetilde{Y}_i - \tau_0 D_i\right) Z_i}{\sum\limits_{i \in \mathcal{I}_0} Z_i} - \frac{\sum\limits_{i \in \mathcal{I}_0} \left(\widetilde{Y}_i - \tau_0 D_i\right) (1 - Z_i)}{\sum\limits_{i \in \mathcal{I}_0} (1 - Z_i)}$$

where as before $\mathcal{I}_0 = \{i : R_i \in W_0\}$. When the outcomes are transformed using a linear model, this statistic is equivalent to the difference in the intercepts between two reduced form regressions of Y_i on r_i, one at each side of the cutoff, then adjusted by τ_0.

For comparison, we provide an implementation of a Wald-type two-stage least-squares statistic (TSLS). Inference for the TSLS statistic relies on asymptotic approximations and is sensitive to weak instruments.

2. AR stands for Anderson–Rubin.

3 The rdrandinf command

This section describes the syntax of `rdrandinf`, which calculates randomization p-values on a specified window under different randomization mechanism and potential-outcomes models.

3.1 Description

`rdrandinf` implements randomization inference and related methods for RD designs, using observations in a specified or data-driven selected window around the cutoff where local randomization is assumed to hold.

3.2 Syntax

`rdrandinf` *outvar runvar* [*if*] [*in*] [`, cutoff(#) wl(#) wr(#)`
`statistic(`*stat*`) p(#) evall(#) evalr(#) kernel(`*kerneltype*`) nulltau(#)`
`ci(`*level* [*tlist*]`) interfci(#) fuzzy(`*fuzzy_var* [*fuzzy_stat*]`) d(#) dscale(#)`
`bernoulli(`*varname*`) reps(#) seed(#) covariates(`*varlist*`) obsmin(#)`
`obsstep(#) wmin(#) wstep(#) nwindows(#) rdwstat(`*stat*`) approximate`
`rdwreps(#) level(#) plot graph_options(`*graphopts*`) quietly`]

outvar is the outcome variable. *runvar* is the running variable (also known as the score or forcing variable).

3.3 Options

`cutoff(#)` specifies the RD cutoff for the running variable *runvar*. The default is `cutoff(0)`.

`wl(#)` specifies the left limit of the window. The default is the minimum of the running variable.

`wr(#)` specifies the right limit of the window. The default is the maximum of the running variable.

`statistic(`*stat*`)` specifies the statistic to be used for randomization inference. *stat* may be `ttest` (DM), `ksmirnov` (KS statistic), `ranksum` (Wilcoxon–Mann–Whitney studentized statistic), or `all`, which specifies all three statistics. The default is `statistic(ttest)`.

`p(#)` specifies the order of the polynomial for the outcome transformation model. The default is `p(0)`.

`evall(#)` specifies the point at the left of the cutoff at which the transformed outcome is evaluated. The default is the cutoff value.

`evalr(#)` specifies the point at the right of the cutoff at which the transformed outcome is evaluated. The default is the cutoff value.

`kernel(`*kerneltype*`)` specifies the type of kernel to use as the weighting scheme. *kerneltype* may be `uniform` (uniform kernel), `triangular` (triangular kernel), or `epan` (Epanechnikov kernel). The default is `kernel(uniform)`.

`nulltau(#)` sets the value of the treatment effect under the null hypothesis. The default is `nulltau(0)`.

`ci(`*level* [*tlist*]`)` calculates a confidence interval for the treatment effect by test inversion, where *level* specifies the level of the confidence interval and *tlist* indicates the grid of treatment effects to be evaluated. This option uses `rdsensitivity` to calculate the confidence interval; type `help rdsensitivity` for details.

`interfci(#)` sets the level for Rosenbaum's confidence interval under arbitrary interference between units (Rosenbaum 2007).

`fuzzy(`*fuzzy_var* [*fuzzy_stat*]`)` specifies the name of the endogenous treatment variable in the fuzzy design. The options for statistics in fuzzy designs are `ar` for an Anderson–Rubin-type statistic and `tsls` for a TSLS statistic. The default *fuzzy_stat* is `ar`.

`d(#)` specifies the effect size for asymptotic power calculation. The default is 0.5 times the standard deviation of the outcome variable for the control group.

`dscale(#)` specifies the fraction of the standard deviation of the outcome variable for the control group used as an alternative hypothesis for asymptotic power calculation. The default is `dscale(.5)`.

`bernoulli(`*varname*`)` specifies that the randomization mechanism follow Bernoulli trials instead of fixed margins randomization. The values of the probability of treatment for each unit are indicated in *varname*.

`reps(#)` specifies the number of replications. The default is `reps(1000)`.

`seed(#)` sets the seed for the randomization test. With this option, the user can manually set the desired seed or can enter the value -1 to use the system seed. The default is `seed(666)`.

Note: When the window around the cutoff is not specified, `rdrandinf` can select the window automatically using the `rdwinselect` command. The following options are available:

`covariates(`*varlist*`)` specifies the covariates used by the `rdwinselect` command.

`obsmin(#)` specifies the minimum number of observations above and below the cutoff in the smallest window used by the `rdwinselect` command. The default is `obsmin(10)`.

`obsstep(#)` specifies the minimum number of observations to be added on each side of the cutoff for the sequence of nested windows constructed by the `rdwinselect` command. The default is `obsstep(2)`.

`wmin(#)` specifies the smallest window to be used (if `minobs()` is not specified) by the `rdwinselect` command. Specifying both `wmin()` and `obsmin()` returns an error.

`wstep(#)` specifies the increment in window length (if `obsstep()` is not specified) by the `rdwinselect` command. Specifying both `obsstep()` and `wstep()` returns an error.

`nwindows(#)` specifies the number of windows to be used by the `rdwinselect` command. The default is `nwindows(10)`.

`rdwstat(`*stat*`)` specifies the statistic to be used by the `rdwinselect` command (see help file for options). The default is `rdwstat(ttest)`.

`approximate` forces the `rdwinselect` command to conduct the covariate balance tests using a large-sample approximation instead of finite-sample exact randomization inference methods.

`rdwreps(#)` specifies the number of replications to be used by the `rdwinselect` command. The default is `rdwreps(1000)`.

`level(#)` specifies the minimum accepted value of the p-value from the covariate balance tests to be used by the `rdwinselect` command. The default is `level(.15)`.

`plot` draws a scatterplot of the minimum p-value from the covariate balance test against window length implemented by the `rdwinselect` command.

`graph_options(`*graphopts*`)` passes the *graphopts* options to the plot. Options such as titles should be written without double quotes.

`quietly` suppresses output from the `rdwinselect` command.

4 The rdwinselect command

This section describes the syntax of the `rdwinselect` command. This command finds the largest window in which a set of covariates is found to be balanced between treated and control groups.

4.1 Description

`rdwinselect` implements the window-selection procedure based on balance tests for RD designs under local randomization. Specifically, it constructs a sequence of nested windows around the RD cutoff and reports binomial tests for the running variable *runvar* and covariate balance tests for covariates *covariates* (if specified). The recommended window is the largest window around the cutoff such that the minimum p-value of the balance test is larger than a prespecified level for all nested (smaller) windows. By default, the p-values are calculated using randomization inference methods.

4.2 Syntax

`rdwinselect` *runvar* [*covariates*] [*if*] [*in*] [`,` `cutoff(#)` `obsmin(#)` `obsstep(#)` `wmin(#)` `wstep(#)` `nwindows(#)` `statistic(`*stat*`)` `approximate` `p(#)` `evalat(`*point*`)` `kernel(`*kerneltype*`)` `reps(#)` `seed(#)` `level(#)` `plot` `graph_options(`*graphopts*`)`]

runvar is the running variable (also known as the score or forcing variable). *covariates* is the list of covariates to be used in the balancing tests. This list is optional, but the recommended window is provided only when at least one covariate is specified.

4.3 Options

`cutoff(#)` specifies the RD cutoff for the running variable *runvar*. The default is `cutoff(0)`.

`obsmin(#)` specifies the minimum number of observations above and below the cutoff in the smallest window. The default is `obsmin(10)`.

`obsstep(#)` specifies the minimum number of observations to be added on each side of the cutoff in all but the first window. The default is `obsstep(2)`.

`wmin(#)` specifies the smallest window to be used (if `minobs()` is not specified). Specifying `wmin()` and `obsmin()` returns an error.

`wstep(#)` specifies the increment in window length (if `obsstep()` is not specified). Specifying both `obsstep()` and `wstep()` returns an error.

`nwindows(#)` specifies the number of windows to be used. The default is `nwindows(10)`.

`statistic(`*stat*`)` specifies the statistic to be used. *stat* may be one of the following: `ttest` (DM), `ksmirnov` (KS statistic), `ranksum` (Wilcoxon–Mann–Whitney studentized statistic), or `hotelling` (Hotelling's T-squared statistic). The default is `statistic(ttest)`.

`approximate` performs the covariate balance test using a large-sample approximation instead of randomization inference.

`p(#)` specifies the order of the polynomial for the outcome adjustment model. The default is `p(0)`.

`evalat(`*point*`)` specifies the point at which the adjusted variable is evaluated. *point* may be `cutoff` or `means`. The default is `evalat(cutoff)`.

`kernel(`*kerneltype*`)` specifies the type of kernel to use as the weighting scheme. *kerneltype* may be `uniform` (uniform kernel), `triangular` (triangular kernel), or `epan` (Epanechnikov kernel). The default is `kernel(uniform)`.

`reps(#)` sets the number of replications for the randomization test. The default is `reps(1000)`.

`seed(#)` sets the initial seed for the randomization test. With this option, the user can manually set the desired seed or can enter the value -1 to use the system seed. The default is `seed(666)`.

`level(#)` specifies the minimum accepted value of the p-value from the covariate balance tests to be used. The default is `level(.15)`.

`plot` draws a scatterplot of the minimum p-value from the covariate balance test against window length.

`graph_options(`*graphopts*`)` passes the *graphopts* options to the plot. Options such as titles should be written without double quotes.

5 The rdsensitivity command

This section describes the syntax of the `rdsensitivity` command, which is used to analyze the sensitivity of randomization p-values and confidence intervals to different window lengths.

5.1 Description

`rdsensitivity` performs sensitivity analysis for RD designs under local randomization.

5.2 Syntax

`rdsensitivity` *outvar runvar* [*if*] [*in*] [`, cutoff(#) wlist(`*numlist*`)`
`tlist(`*numlist*`) saving(`*filename*`) nodots nodraw verbose`
`ci(`*window* [*level*]`) statistic(`*stat*`) p(#) evalat(`*point*`) kernel(`*kerneltype*`)`
`fuzzy(`*fuzzy_var*`) reps(#) seed(#)`]

outvar is the outcome variable. *runvar* is the running variable (also known as the score or forcing variable).

5.3 Options

`cutoff(#)` specifies the RD cutoff for the running variable *runvar*. The default is `cutoff(0)`.

`wlist(`*numlist*`)` specifies the list of window lengths to be evaluated. By default, the program constructs 10 windows around the cutoff, the first one including 10 treated and control observations and adding 5 observations to each group in subsequent windows.

`tlist(`*numlist*`)` specifies the list of values of the treatment effect under the null to be evaluated. By default, the program uses 10 evenly spaced points within the asymptotic confidence interval for a constant treatment effect in the smallest window to be used.

`saving(`*filename*`)` saves the dataset containing the data for the contour plot in *filename*. This allows the user to replicate and modify the appearance of the plot and conduct further sensitivity analysis.

`nodots` suppresses replication dots.

`nodraw` suppresses the contour plot.

`verbose` displays the matrix of results.

`ci(`*window* [*level*]`)` returns the confidence interval corresponding to the window length indicated in *window*. The value in `ci()` needs to be one of the values in `wlist()`. The level of the confidence interval can be specified with the `level()` option. The default level is 0.05, corresponding to a 95% confidence interval.

`statistic(`*stat*`)` specifies the statistic to be used in randomization inference. *stat* may be `ttest` (DM), `ksmirnov` (KS statistic), `ranksum` (Wilcoxon–Mann–Whitney studentized statistic), or `all`, which specifies all three statistics. The default is `statistic(ttest)`.

`p(#)` specifies the order of the polynomial for the outcome transformation model. The default is `p(0)`.

`evalat(`*point*`)` specifies the point at which the adjusted variable is evaluated. *point* may be `cutoff` or `means`. The default is `evalat(cutoff)`.

`kernel(`*kerneltype*`)` specifies the type of kernel to use as the weighting scheme. *kerneltype* may be `uniform` (uniform kernel), `triangular` (triangular kernel), or `epan` (Epanechnikov kernel). The default is `kernel(uniform)`.

`fuzzy(`*fuzzy_var*`)` specifies the name of the endogenous treatment variable in the fuzzy design. This option uses an Anderson–Rubin-type statistic.

`reps(#)` specifies the number of replications for the randomization test. The default is `reps(1000)`.

`seed(#)` sets the initial seed for the randomization test. With this option, the user can manually set the desired seed or can enter the value −1 to use the system seed. The default is `seed(666)`.

6 The rdrbounds command

This section describes the syntax of the `rdrbounds` command, which calculates lower and upper bounds for the randomization p-value under different degrees of departure from a local randomized experiment, as suggested by Rosenbaum (2002).

6.1 Description

`rdrbounds` calculates Rosenbaum bounds for p-values in RD designs under local randomization.

6.2 Syntax

`rdrbounds` *outvar runvar* [*if*] [*in*] [`, cutoff(#) prob(`*varname*`)`
`gammalist(`*numlist*`) expgamma(`*numlist*`) wlist(`*numlist*`) ulist(`*numlist*`)`
`bound(`*bounds*`) fmpval statistic(`*stat*`) p(#) evalat(`*point*`)`
`kernel(`*kerneltype*`) nulltau(#) fuzzy(`*fuzzy_var*`) reps(#) seed(#)`]

outvar is the outcome variable. *runvar* is the running variable (also known as the score or forcing variable).

6.3 Options

`cutoff(#)` specifies the RD cutoff for the running variable *runvar*. The default is `cutoff(0)`.

`prob(`*varname*`)` specifies the name of the variable containing individual probabilities of treatment in a Bernoulli trial when the selection factor gamma is zero. The default is the proportion of treated units in each window (assumed equal for all units).

`gammalist(`*numlist*`)` specifies the list of values of gamma to be evaluated.

`expgamma(`*numlist*`)` specifies the list of values of exp(gamma) to be evaluated. The default is `expgamma(1.5 2 2.5 3)`.

`wlist(`*numlist*`)` specifies the list of window lengths to be evaluated. By default, the program constructs 10 windows around the cutoff, the first one including 10 treated and control observations and then adding 5 observations to each group in subsequent windows.

`ulist(`*numlist*`)` specifies the list of vectors of the unobserved confounder to be evaluated. The default takes all vectors with ones in the first k positions and zeros in the remaining position; see Rosenbaum (2002).

`bound(`*bounds*`)` specifies which bounds the command calculates. *bounds* may be `upper` (upper bound), `lower` (lower bound), or `both` (upper and lower bounds). The default is `bound(both)`.

`fmpval` calculates the p-value under fixed margins randomization in addition to the p-value under Bernoulli trials.

`statistic(`*stat*`)` specifies the statistic to be used in randomization inference. *stat* may be `ttest` (DM), `ksmirnov` (KS statistic), or `ranksum` (Wilcoxon–Mann–Whitney studentized statistic). The default is `statistic(ranksum)`.

`p(`#`)` specifies the order of the polynomial for the outcome transformation model. The default is `p(0)`.

`evalat(`*point*`)` specifies the point at which the transformed variable is evaluated. *point* may be `cutoff` or `means`. The default is `evalat(cutoff)`.

`kernel(`*kerneltype*`)` specifies the type of kernel to use as the weighting scheme. *kerneltype* may be `uniform` (uniform kernel), `triangular` (triangular kernel), or `epan` (Epanechnikov kernel). The default is `kernel(uniform)`.

`nulltau(`#`)` sets the value of the treatment effect under the null hypothesis. The default is `nulltau(0)`.

`fuzzy(`*fuzzy_var*`)` specifies the name of the endogenous treatment variable in the fuzzy design. This option uses an Anderson–Rubin-type statistic.

`reps(`#`)` sets the number of replications for the randomization test. The default is `reps(500)`.

`seed(`#`)` sets the initial seed for the randomization test. With this option, the user can manually set the desired seed or can enter the value -1 to use the system seed. The default is `seed(666)`.

7 Illustration of methods

We illustrate our commands using the dataset from Cattaneo, Frandsen, and Titiunik (2015), which has also been used to illustrate the nonparametric local polynomial methods in Calonico, Cattaneo, and Titiunik (2014a, 2015b) and Cattaneo, Jansson, and Ma (2016a). `rdlocrand_senate.dta` contains information on 1,390 U.S. Senate elections between 1914 and 2010 and was used before to analyze the effect of the incumbent status of a political party on the probability of winning future elections. The running variable in this dataset is `demmv`, the Democratic margin of victory in a statewide Senate election at t, defined as the difference in vote share between the Democratic party and its strongest opponent. A positive value of the running variable indicates that the Democratic party won the election, and the cutoff is therefore $\overline{r} = 0$.

We start by loading the dataset and providing some descriptive statistics:

```
. use rdlocrand_senate
. global covariates presdemvoteshlag1 population demvoteshlag1 demvoteshlag2
> demwinprv1 demwinprv2 dopen dmidterm
. describe $covariates
              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
presdemvotesh~1 float   %9.0g                 Democratic presidential vote share
                                                at t-1
population      long    %10.0g                Population
demvoteshlag1   float   %9.0g                 Democratic vote share at t-1
demvoteshlag2   float   %9.0g                 Democratic vote share at t-2
demwinprv1      float   %9.0g                 =1 if Democratic won at t-1
demwinprv2      float   %9.0g                 =1 if Democratic won at t-2
dopen           float   %9.0g                 =1 if open seat
dmidterm        float   %9.0g                 =1 if midterm election at t
. summarize demmv $covariates
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       demmv |      1,390    7.171159    34.32488       -100        100
presdemvot~1 |      1,387    46.11975    14.31701          0   97.03408
  population |      1,390     3827919     4436950      78000   3.73e+07
demvoteshl~1 |      1,349    52.69048     18.2706          0        100
demvoteshl~2 |      1,308    52.86918    18.23913          0        100
-------------+---------------------------------------------------------
  demwinprv1 |      1,349    .5441067    .4982355          0          1
  demwinprv2 |      1,308     .543578    .4982879          0          1
       dopen |      1,380    .2471014    .4314826          0          1
    dmidterm |      1,390    .5136691     .499993          0          1
```

The running variable ranges from -100 to 100 with an average of 7 percentage points. The outcome of interest is `demvoteshfor2`, the Democratic vote share in the following election for the same Senate seat—which, given the staggered nature of Senate elections in the United States, occurs two elections later, at $t+2$. The set of covariates, described in the output above is exactly the one used in Cattaneo, Frandsen, and Titiunik (2015); thus we can replicate their empirical findings using the new commands introduced here.

The most basic syntax for `rdwinselect` is the following:

```
. rdwinselect demmv $covariates, cutoff(0)

Window selection for RD under local randomization

Cutoff c = 0.00 | Left of c   Right of c        Number of obs  =          1390
----------------+----------------------        Order of poly  =             0
    Number of obs |       640          750       Kernel type    =       uniform
   1st percentile |         6            7       Reps           =          1000
   5th percentile |        32           37       Testing method =     rdrandinf
  10th percentile |        64           75       Balance test   =         ttest
  20th percentile |       128          150

                  |   Bal. test         Var. name    Bin. test
 Window length /2 |    p-value        (min p-value)   p-value     Obs<c   Obs>=c
-----------------+------------------------------------------------------------
            0.529 |      0.210       demvoteshlag2      0.327        10       16
            0.733 |      0.262            dopen         0.200        15       24
            0.937 |      0.132            dopen         0.126        16       27
            1.141 |      0.044            dopen         0.161        20       31
            1.346 |      0.229         dmidterm         0.382        28       36
            1.550 |      0.102         dmidterm         0.728        35       39
            1.754 |      0.075         dmidterm         0.747        41       45
            1.958 |      0.046         dmidterm         0.602        43       49
            2.163 |      0.075         dmidterm         0.480        45       53
            2.367 |      0.132            dopen         0.637        53       59
Variable used in binomial test (running variable): demmv
Covariates used in balance test: presdemvoteshlag1 population demvoteshlag1
> demvoteshlag2 demwinprv1 demwinprv2 dopen dmidterm

Recommended window is [-.733; .733] with 39 observations (15 below, 24 above).
```

Because in this particular application the cutoff is zero, which is the default value, the `cutoff()` option can be omitted. Thus all the remaining examples will not specify this option. In practice, when the cutoff is not zero, the user can simply specify `cutoff()`. Alternatively, it may be easier to simply redefine the running variable by recentering it at the cutoff. By default, `rdwinselect` uses the difference-in-means statistic to perform hypothesis tests—but this can be changed with the `statistic()` option.

The output of `rdwinselect` is divided in three panels. The upper-left panel provides information on sample sizes. The first row gives the total number of observations to the left and to the right of the cutoff and also the total sample size. The following four rows provide the same information but around small neighborhoods around the cutoffs defined by the 1st, 5th, 10th, and 20th percentiles of the running variable.

The upper-right panel indicates the total sample size, the degree of the polynomial used by `rdrandinf`, the type of kernel used for the weighting scheme (`uniform`, `triangular`, or `epan`), the number of replications in the permutation test (whenever this test is performed), the method used to perform the covariate balance tests (`approximate` or `rdrandinf`), and the test statistic used (`test`, `ksmirnov`, or `ranksum`).

Finally, the main panel gives the result of the two balance tests performed at each of the windows considered. The first column provides the window length of each window considered, divided by two. For example, a value of 0.529 in this column refers to the window $[\overline{r} - 0.529; \overline{r} + 0.529]$, where $\overline{r}$ is the cutoff (equal to 0 in this case) and the

window length is $\overline{r}+0.529-(\overline{r}-0.529)=1.058$. The second column, labeled `Bal. test p-value`, provides the minimum p-value of the balancing test; the name of the corresponding variable associated with this p-value is given in column 3, `Var. name (min p-value)`. The p-value is obtained by either randomization inference or an asymptotic approximation, depending on the option specified. The fourth column gives the p-value from a binomial probability test of the hypothesis that the probability of treatment is 0.5. Type `help bitest` for further details. Columns 5 and 6 give the number of observations to the left and right of the cutoff inside each window. As indicated in the last line of the output, the largest recommended window (the largest window for which the second column is equal to or above 0.15) in this case is $[-0.733; 0.733]$ and contains 15 observations below the cutoff and 24 observations above.

By default, `rdwinselect` starts with a window that contains at least 10 observations at each side of the cutoff and increases the length, ensuring that at least 2 observations are added in each successive window. The user can choose these two values using the `obsmin()` and `obsstep()` options, respectively, or can define the windows in terms of their length instead of the number of observations. For instance, Cattaneo, Frandsen, and Titiunik (2015) start from the window $[-0.5; 0.5]$ and increase the width by 0.125 using 10,000 replications in the permutation test. To replicate their results, we can type

```
. rdwinselect demmv $covariates, wmin(.5) wstep(.125) reps(10000)

Window selection for RD under local randomization

Cutoff c = 0.00 | Left of c   Right of c        Number of obs  =          1390
----------------+----------------------        Order of poly  =             0
    Number of obs |       640          750       Kernel type    =       uniform
   1st percentile |         6            7       Reps           =         10000
   5th percentile |        32           37       Testing method =     rdrandinf
  10th percentile |        64           75       Balance test   =         ttest
  20th percentile |       128          150

                  |   Bal. test         Var. name    Bin. test
 Window length /2 |    p-value       (min p-value)   p-value     Obs<c   Obs>=c
-----------------+------------------------------------------------------------
            0.500 |      0.268       demvoteshlag2      0.230         9       16
            0.625 |      0.423              dopen       0.377        13       19
            0.750 |      0.265              dopen       0.200        15       24
            0.875 |      0.153              dopen       0.211        16       25
            1.000 |      0.074              dopen       0.135        17       28
            1.125 |      0.039              dopen       0.119        19       31
            1.250 |      0.063              dopen       0.105        21       34
            1.375 |      0.140           dmidterm       0.539        30       36
            1.500 |      0.092           dmidterm       0.640        34       39
            1.625 |      0.113           dmidterm       0.734        37       41
Variable used in binomial test (running variable): demmv
Covariates used in balance test: presdemvoteshlag1 population demvoteshlag1
> demvoteshlag2 demwinprv1 demwinprv2 dopen dmidterm

Recommended window is [-.875; .875] with 41 observations (16 below, 25 above).
```

Note that the minimum p-value for the largest recommended window, which is $[-0.875; 0.875]$, is slightly above the cutoff value 0.15. To compare our results with the ones in Cattaneo, Frandsen, and Titiunik (2015), we will set the window $[-0.75; 0.75]$ as our preferred choice.[3]

Additionally, observe that the minimum p-value is not necessarily monotonic on the length of the window. The `plot` option allows the user to depict graphically how these values change for different lengths. We will set the number of windows to 80 to have more observations in the plot, and we will specify the `approximate` option to speed up the calculations. With this option, the command uses the large-sample approximation instead of randomization inference. It is useful for illustration because it is much faster, but it can be misleading because the approximation may be poor when the sample is small. The output from `rdwinselect` with 80 windows is a long table and will be omitted. The resulting graph is shown in figure 1.

```
. quietly rdwinselect demmv $covariates, wmin(.5) wstep(.125)
> nwin(80) approximate plot
```

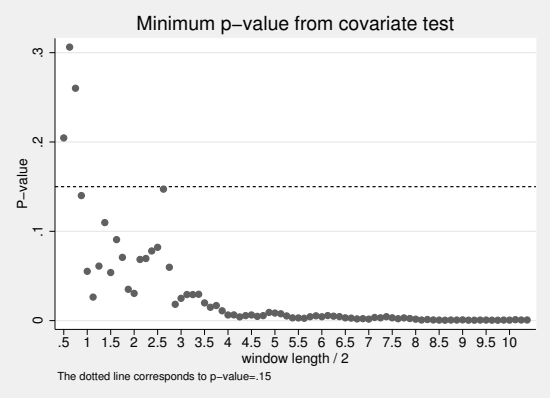

Figure 1. Plot of p-values

The figure shows that the p-values vary widely for very short windows, but the sequence stabilizes once the window length is large enough (around the value 3 in this case).

Once the window has been selected, randomization inference to test the sharp null hypothesis of no treatment effect can be performed using `rdrandinf`. For example, take the window $[-0.75; 0.75]$, which is the one selected by Cattaneo, Frandsen, and Titiunik (2015) and replicated above. The basic syntax for `rdrandinf` is

3. The user trying to replicate this code should be aware that these calculations involve permuting the treatment variable many times for a large set of covariates. Running this command may take about 40 minutes.

```
. rdrandinf demvoteshfor2 demmv, wl(-.75) wr(.75)
Selected window = [-.75 ; .75]
Running randomization-based test...
Randomization-based test complete.

Inference for sharp design

Cutoff c = 0.00 |  Left of c  Right of c        Number of obs =           1390
                |                              Order of poly =              0
     Number of obs |        595         702     Kernel type   =        uniform
Eff. Number of obs |         15          22     Reps          =           1000
   Mean of outcome |     42.808      52.497     Window        =    set by user
   S.D. of outcome |      7.042       7.742     H0:       tau =          0.000
            Window |     -0.750       0.750     Randomization =  fixed margins
Outcome: demvoteshfor2. Running variable: demmv.

                |                Finite sample          Large sample
        Statistic |       T          P>|T|        P>|T|   Power vs d =      3.52
------------------+-------------------------------------------------------------
    Diff. in means |     9.689        0.000         0.000                   0.300
```

Like the output of `rdwinselect`, that of `rdrandinf` is divided in three panels. The upper-left panel provides the number of observations at each side of the cutoff, sample size below and above the cutoff inside the specified window, some descriptive statistics for the outcome inside the window, and the selected window. Note that the first line in this panel displays the number of observations with nonmissing values of the outcome and running variable, so the sample sizes shown can differ from the total sample size. The upper-right panel gives the total sample size, the order of the polynomial, the type of kernel used for the weighting scheme (`uniform`, `triangular`, or `epan`), the number of replications in the permutation test, and whether the window was specified by the user by setting `wl()` and `wr()` or calculated using `rdwinselect`, as will be illustrated shortly.

Finally, the main panel gives the results from the randomization test. The first column, labeled `Statistic`, indicates the statistic used in the randomization test. The second column gives the observed value of the selected statistic, and the third column shows its finite-sample p-value obtained from the randomization test. The fourth column gives the asymptotic p-value, that is, the p-value obtained from the corresponding asymptotic distribution of the chosen statistic. Finally, the fifth column gives the asymptotic power against an alternative value that can be specified using the `d()` or `dscale()` option. The default is `dscale(.5)`, that is, an effect size equal to half the standard deviation of the outcome for the control group inside the window (the critical value for the power calculation is set to 1.96).

As mentioned above, `rdrandinf` uses the DM as the default statistic, but it can also use the KS and the RS statistics. By adding `statistic(all)` as an option, we can obtain the result for all three statistics. These last two statistics are obtained using the `ksmirnov` and `ranksum` Stata commands, respectively. See the corresponding Stata help files for further details. The output is

```
. rdrandinf demvoteshfor2 demmv, wl(-.75) wr(.75) statistic(all)
Selected window = [-.75 ; .75]
Running randomization-based test...
Randomization-based test complete.

Inference for sharp design

Cutoff c = 0.00 |  Left of c  Right of c        Number of obs =           1390
                |                             Order of poly =              0
     Number of obs |        595         702    Kernel type   =        uniform
Eff. Number of obs |         15          22    Reps          =           1000
   Mean of outcome |     42.808      52.497    Window        =    set by user
   S.D. of outcome |      7.042       7.742    H0:       tau =          0.000
            Window |     -0.750       0.750    Randomization =  fixed margins

Outcome: demvoteshfor2. Running variable: demmv.

                |                Finite sample          Large sample
        Statistic |       T          P>|T|        P>|T|   Power vs d =     3.52
------------------+-------------------------------------------------------------
    Diff. in means |     9.689        0.000        0.000                  0.300
Kolmogorov-Smirnov |     0.552        0.002        0.005                      .
   Rank sum z-stat |    -3.217        0.000        0.001                  0.209
```

We can see that the three statistics provide basically the same result in terms of inference; the randomization test rejects the sharp null of no treatment effect at one percent significance level in all three cases. Also note that the `rdrandinf` command does not provide the asymptotic power for the KS statistic.

The window in which to perform the randomization-based tests can be set manually using `wl()` and `wr()`. These options specify the lower and upper limits of the chosen window. Importantly, these are window limits and not lengths, so for instance, if the cutoff is 100 and the user wants a window of ± 5, the correct syntax is `wl(95) wr(105)`. We advise the user to always normalize the cutoff to zero by centering the running variable to avoid confusion.

Alternatively, the user can specify the list of covariates to have `rdrandinf` select the window automatically using `rdwinselect`. All the options allowed in `rdwinselect` can be passed through `rdrandinf`. By default, the output for `rdwinselect` is displayed before the output for `rdrandinf`, but it can be suppressed using the `quietly` option. For example,

```
. rdrandinf demvoteshfor2 demmv, statistic(all) covariates($covariates)
> wmin(.5) wstep(.125) level(0.16) quietly rdwreps(10000)
Calculating window...

Selected window = [-.75 ; .75]

Running randomization-based test...
Randomization-based test complete.

Inference for sharp design

Cutoff c = 0.00 |  Left of c  Right of c        Number of obs =           1390
----------------+----------------------        Order of poly =              0
     Number of obs |       595         702       Kernel type   =        uniform
Eff. Number of obs |        15          22       Reps          =           1000
   Mean of outcome |    42.808      52.497       Window        =    rdwinselect
   S.D. of outcome |     7.042       7.742       H0:       tau =          0.000
            Window |    -0.750       0.750       Randomization =  fixed margins

Outcome: demvoteshfor2. Running variable: demmv.

                |                 Finite sample          Large sample
----------------+-----------------------------------------------------------
        Statistic |         T          P>|T|        P>|T|   Power vs d =     3.52
----------------+-----------------------------------------------------------
    Diff. in means |     9.689          0.000          0.000                 0.300
Kolmogorov-Smirnov |     0.552          0.002          0.005                     .
   Rank sum z-stat |    -3.217          0.000          0.001                 0.209
----------------+-----------------------------------------------------------
```

Note that the reported p-values are slightly different. As explained above, the reason is that the two commands are performing the randomization test starting from different seeds. The user can obtain the exact same results for the two syntaxes by setting the same seed—for example, `seed(9876)`—in both commands.

The `rdrandinf` command allows the user to specify a polynomial transformation model for the outcomes using the `p()` option. By default, the command sets `p(0)`, which means no transformation. When `p()` is set to an integer larger than zero, the slopes (and possibly higher-order terms) are subtracted from the outcomes, leaving a residualized version of the outcome that differs only above and below the cutoff in the intercept. For instance, to perform a linear transformation, we type

```
. rdrandinf demvoteshfor2 demmv, statistic(all) wl(-.75) wr(.75) p(1)
Selected window = [-.75 ; .75]

Running randomization-based test...
Randomization-based test complete.

Inference for sharp design

Cutoff c = 0.00 | Left of c  Right of c        Number of obs =          1390
----------------+-----------------------       Order of poly =             1
   Number of obs |       595         702        Kernel type   =       uniform
Eff. Number of obs |      15          22        Reps          =          1000
  Mean of outcome |   42.808      52.497        Window        =   set by user
  S.D. of outcome |    7.042       7.742        H0:       tau =         0.000
           Window |   -0.750       0.750        Randomization =  fixed margins

Outcome: demvoteshfor2. Running variable: demmv.
--------------------------------------------------------------------------------
                  |             Finite sample            Large sample
                  |             -------------    ----------------------------
        Statistic |        T           P>|T|       P>|T|   Power vs d =     3.52
--------------------------------------------------------------------------------
    Diff. in means|   15.297           0.000       0.066                    0.071
Kolmogorov-Smirnov|    0.797           0.000           .                        .
    Rank sum z-stat|  -4.455           0.000           .                        .
--------------------------------------------------------------------------------
```

When a model for the outcomes is specified—that is, when `p()` is set to a number greater than zero—with the option `statistic(ttest)`, `rdrandinf` fits a regression of the outcome on the treatment dummy interacted with a polynomial of the running variable, and uses the difference in intercepts as the test-statistic. The other test statistics use the residuals described above as outcomes. Note that the command does not provide the asymptotic p-value or the asymptotic power of the KS and RS statistics, because the asymptotic distribution does not account for the model transformation and hence can be misleading.

In the presence of arbitrary interference, a confidence interval for a particular measure of the effects of the program (described in section 2.5) can be obtained with the `interfci()` option. For example, to obtain a 95% confidence interval, we type

```
. rdrandinf demvoteshfor2 demmv, wl(-.75) wr(.75) interfci(.05)
Selected window = [-.75 ; .75]
Running randomization-based test...
Randomization-based test complete.

Inference for sharp design

Cutoff c = 0.00 |  Left of c  Right of c        Number of obs =           1390
                |                             Order of poly =              0
     Number of obs |        595         702    Kernel type   =        uniform
Eff. Number of obs |         15          22    Reps          =           1000
   Mean of outcome |     42.808      52.497    Window        =    set by user
   S.D. of outcome |      7.042       7.742    H0:       tau =          0.000
            Window |     -0.750       0.750    Randomization =  fixed margins
Outcome: demvoteshfor2. Running variable: demmv.

                |                Finite sample          Large sample
        Statistic |       T           P>|T|          P>|T|   Power vs d =     3.52
------------------+-------------------------------------------------------------
    Diff. in means |     9.689         0.000          0.000                  0.300

Confidence interval under interference

        Statistic |  [95% Conf. Interval]
------------------+----------------------
    Diff. in means |     3.963      15.525
```

In terms of interpretation, one should remember that the confidence interval under interference is not a confidence interval for the point estimate (and in fact, it may not even contain the point estimate). As explained above, the interference confidence interval is constructed based on the difference between the observed statistic and the statistic that would be observed if the treatment was withheld from all units. In our application, allowing for arbitrary interference, we can say with 95% confidence that the "excess" benefit of the treated group compared with the control group is roughly between 4 and 15.5. Again, in this particular example, the point estimate under SUTVA happens to be contained in the confidence interval under interference, but this need not be the case and has no clear interpretation.

The `rdlocrand` package provides two types of sensitivity analyses to assess how p-values change with window length. The first one, `rdsensitivity`, calculates and plots a matrix of p-values over a range of values for the treatment effect under the null hypothesis (rows) and window lengths (columns). For instance, we can see how the p-values change by starting from the selected window, increasing the window length by 0.25 and over a range of treatment effects that is roughly the point estimate plus and minus 10:

```
. rdsensitivity demvoteshfor2 demmv, wlist(.75(.25)2) tlist(0(1)20) nodots verbose
Running randomization-based test...
Randomization-based test complete.
```

	.75	1	1.25	1.5	1.75	2
0	0	.001	0	0	0	0
1	0	.001	0	0	0	0
2	.001	.001	.002	0	0	0
3	.013	.004	.003	0	0	0
4	.03	.009	.007	.002	.001	0
5	.068	.023	.018	.013	.003	.005
6	.147	.062	.042	.037	.039	.029
7	.309	.144	.093	.088	.106	.092
8	.518	.306	.173	.239	.233	.262
9	.788	.534	.299	.427	.484	.569
10	.918	.869	.497	.731	.83	.939
11	.608	.844	.756	.907	.839	.668
12	.378	.514	.969	.574	.496	.36
13	.201	.268	.665	.323	.231	.134
14	.102	.139	.428	.154	.09	.036
15	.04	.051	.254	.064	.035	.007
16	.019	.016	.13	.022	.009	.002
17	.008	.006	.073	.004	.003	0
18	.003	0	.032	.001	0	0
19	.001	0	.01	.001	0	0
20	0	0	.002	0	0	0

In the above syntax, `nodots` suppresses the replication dots, and `verbose` indicates that the matrix of p-values will be displayed as part of the output (otherwise, the only output of the command is the plot, unless `nodraw` is specified).

One way to interpret these results is to see the range of values for which the p-value is above, say, 0.05, as a 95% confidence interval for the point estimate (assuming a constant additive treatment effect). Thus the above table shows how the confidence interval for the treatment effect changes as the window length increases. For instance, the 95% confidence interval for the window $[-.75; .75]$ is roughly [5; 14], whereas for the window $[-2; 2]$, it becomes [7; 13]. In this case, the point estimate seems to be relatively stable over the range of windows considered. A more thorough analysis should consider a finer grid of values in `wlist` and `tlist` and a larger number of replications, although these changes can increase computing time considerably. The options `wlist()` and `tlist()` admit the usual Stata *numlist* syntax, such as `wlist(.75 1 1.5 2)`, `wlist(.75(.25)2)`, `wlist(.75(.5)1 2 3)`, etc. The confidence interval for the window $[-.75; .75]$ can be obtained using the `ci()` option:

```
. rdsensitivity demvoteshfor2 demmv, wlist(.75(.25)2) tlist(0(1)20) nodots ci(.75)
Running randomization-based test...
Randomization-based test complete.
Confidence interval for w = .75
```

Statistic	[95% Conf.	Interval]
Diff. in means	5.000	14.000

The `saving()` option saves the dataset with the contour plot data so that the user can replicate and also modify the contour plot. The code to reproduce the default plot is

```
. rdsensitivity demvoteshfor2 demmv, wlist(.75(.25)10) tlist(0(1)20) nodots
> saving(graphdata)
Running randomization-based test...
Randomization-based test complete.
. preserve
. use graphdata, clear
. twoway contour pvalue t w, ccuts(0(0.05)1)
. restore
```

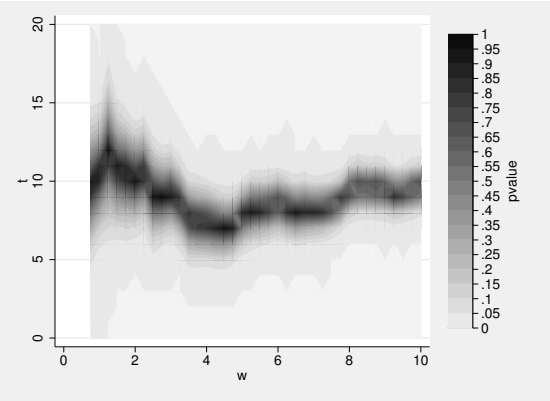

Figure 2. Sensitivity analysis

The resulting graph is shown in figure 2. The plot depicts the grid of window lengths in the horizontal axis and the grid of treatment effects under the null. The color represents the p-value for each pair of window length and treatment effect, where light gray corresponds to zero and black corresponds to one. This is simply a graphical display of the results given by `rdsensitivity`. With the data obtained, and using the `saving()` option, the user can also modify the appearance of the graph or construct different types of plots. For instance, the following command uses a gray scale instead of the default colors, with the resulting plot displayed in figure 3.

```
. preserve
. use graphdata, clear
. twoway contour pvalue t w, ccuts(0(0.05)1) ccolors(gray*0.01 gray*0.05
> gray*0.1 gray*0.15 gray*0.2 gray*0.25 gray*0.3 gray*0.35
> gray*0.4 gray*0.5 gray*0.6 gray*0.7 gray*0.8 gray*0.9 gray
> black*0.5  black*0.6 black*0.7 black*0.8 black*0.9 black)
> xlabel(.75(1.25)10) ylabel(0(2)20, nogrid) graphregion(fcolor(none))
. restore
```

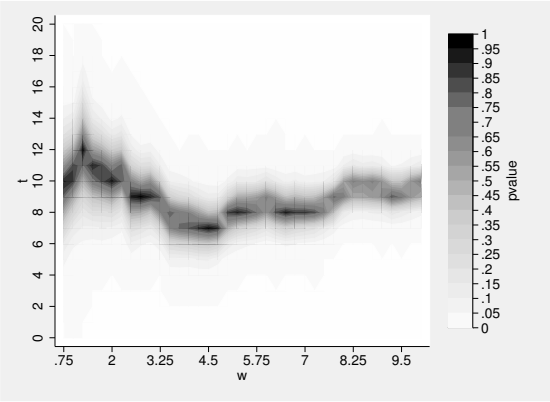

Figure 3. Manually modifying sensitivity plot

Additionally, `rdsensitivity` can be called from within `rdrandinf` to obtain confidence intervals for the point estimates obtained using the `ci()` option. The syntax is the following:

```
. rdrandinf demvoteshfor2 demmv, wl(-.75) wr(.75) ci(.05 3(1)20)
Selected window = [-.75 ; .75]
Running randomization-based test...
Randomization-based test complete.

Inference for sharp design

Cutoff c = 0.00 |  Left of c  Right of c       Number of obs =           1390
----------------+----------------------       Order of poly =              0
     Number of obs |       595         702      Kernel type   =        uniform
Eff. Number of obs |        15          22      Reps          =           1000
   Mean of outcome |    42.808      52.497      Window        =    set by user
   S.D. of outcome |     7.042       7.742      H0:       tau =          0.000
            Window |    -0.750       0.750      Randomization =  fixed margins

Outcome: demvoteshfor2. Running variable: demmv.

                |                 Finite sample          Large sample
        Statistic |       T          P>|T|        P>|T|   Power vs d =      3.52
------------------+-------------------------------------------------------------
    Diff. in means |     9.689        0.000         0.000                  0.300

Calculating confidence interval...
Confidence interval obtained.
```

```
Confidence interval for w = .75
```

Statistic	[95% Conf.	Interval]
Diff. in means	5.000	14.000

The second type of sensitivity analysis is performed with `rdrbounds`. As explained above, this command calculates upper and lower bounds for the randomization p-value under Bernoulli trials for a range of values of a parameter $\Gamma \equiv \exp(\gamma)$ that captures the strength with which an unobservable binary variable U_i affects the probability of selection into treatment $\mathbb{P}[D_i = 1]$. The basic syntax is

```
. rdrbounds demvoteshfor2 demmv, expgamma(1.5 2 3) wlist(.5 .75 1) reps(1000)
Calculating randomization p-values...
```

w =	0.500	0.750	1.000
Bernoulli p-value	0.005	0.000	0.000

```
Running sensitivity analysis...
```

gamma	exp(gamma)	w =	0.500	0.750	1.000
0.41	1.50	lower bound	0.004	0.000	0.000
		upper bound	0.024	0.005	0.002
0.69	2.00	lower bound	0.005	0.000	0.000
		upper bound	0.052	0.025	0.008
1.10	3.00	lower bound	0.005	0.000	0.000
		upper bound	0.194	0.145	0.058

The output from `rdrbounds` is divided in two parts. The first one shows the randomization p-value based on Bernoulli trials for each window. The second panel presents the lower and upper bounds for the p-values for different values of Γ and windows. The wider the distance between the lower and upper bounds, the more sensitive the inference to deviations from a randomized experiment.

There are two directions in which one can look at the second panel. Fixing a column (that is, for a fixed window length) and moving across rows, the table indicates how inference is affected by different degrees of departure from a randomized experiment. This is the type of sensitivity analysis described in Rosenbaum (2002). On the other hand, fixing a row (that is, for a fixed value of Γ) and moving across columns, the matrix shows how sensitive the upper and lower bounds for the p-value are to the choice of the window.

In our application, when the window length is 0.5, the Bernoulli p-value is 0.005. We can see that when $\Gamma = 1.5$, the upper bound is 0.024, so the effect remains significant at 5%. When $\Gamma = 2$, the effect is still significant at the 10% level. On the other hand, when $\Gamma = 3$, the bounds become 0.005 and 0.194, so the evidence for a statistically significant effect is weaker. For the window length selected using `rdwinselect`, $[-0.75; 0.75]$, the p-values look fairly robust to misspecification of the selection mechanism: even if the

unobservable confounder increased the odds ratio of a treated unit compared with a control unit by a factor of 2, the effect would remain significant at the 10% level.

The `fmpval` option adds the fixed margins randomization p-value to the first panel of the output. This allows the user to compare the p-values obtained using each method.

```
. rdrbounds demvoteshfor2 demmv, expgamma(1.5 2 3) wlist(.5 .75 1) reps(1000)
> fmpval
Calculating randomization p-values...
                            w =     0.500      0.750      1.000
------------------------------------------------------------------
              Bernoulli p-value     0.005      0.000      0.000
          Fixed margins p-value     0.007      0.000      0.001
------------------------------------------------------------------

Running sensitivity analysis...
gamma  exp(gamma)          w =     0.500      0.750      1.000
------------------------------------------------------------------
 0.41      1.50    lower bound     0.004      0.000      0.000
                   upper bound     0.024      0.005      0.002
------------------------------------------------------------------
 0.69      2.00    lower bound     0.005      0.000      0.000
                   upper bound     0.052      0.025      0.008
------------------------------------------------------------------
 1.10      3.00    lower bound     0.005      0.000      0.000
                   upper bound     0.194      0.145      0.058
------------------------------------------------------------------
```

We can see that the p-values obtained by both methods are very similar, which we found to be usually true in applications as long as the number of replications is large enough.

Finally, when we use outcome transformation, the `rdrandinf` command allows the user to choose in which point to evaluate the transformed outcomes. By default, the evaluation point is the cutoff, which emulates the idea used in the local polynomial approach of estimating the effect at the cutoff. However, whenever the local randomization assumption is plausible, the cutoff need not be the point of interest. For instance, to set the evaluation points at the means of the running variable inside the window below and above the cutoff, we can type

```
. quietly summarize demmv if abs(demmv)<=.75 & demmv>=0 & demmv!=. &
> demvoteshfor2!=.
. local mt=r(mean)
. quietly summarize demmv if abs(demmv)<=.75 & demmv<0  & demmv!=. &
> demvoteshfor2!=.
. local mc=r(mean)
. rdrandinf demvoteshfor2 demmv, wl(-.75) wr(.75) p(1) evall(`mc´) evalr(`mt´)
Selected window = [-.75 ; .75]
Running randomization-based test...
Randomization-based test complete.
Inference for sharp design
```

Cutoff c = 0.00	Left of c	Right of c		
Number of obs	595	702	Number of obs =	1390
Eff. Number of obs	15	22	Order of poly =	1
Mean of outcome	42.808	52.497	Kernel type =	uniform
S.D. of outcome	7.042	7.742	Reps =	1000
Window	-0.750	0.750	Window =	set by user
			H0: tau =	0.000
			Randomization =	fixed margins

Outcome: demvoteshfor2. Running variable: demmv.

		Finite sample	Large sample	
Statistic	T	P>|T|	P>|T|	Power vs d = 3.52
Diff. in means	9.689	0.000	0.000	0.293

In this case, the user can verify that the point estimate is the same as when no transformation is used: this is because the transformation comes from a linear regression that by construction passes through the means. The p-values, however, can differ. Incidentally, note that the means are taken over the sample inside the window with nonmissing values for the outcome and the running variable. The reason is that `rdrandinf` drops the observations inside the window with missing outcomes or running variables. Similarly, `rdwinselect` drops, at each evaluated window, the observations with missing values of the covariates and running variables.

8 Conclusion

We introduced and discussed the commands `rdrandinf`, `rdwinselect`, `rdsensitivity`, and `rdrbounds`, which together offer a comprehensive and systematic set of tools to analyze RD designs under a local randomization assumption. These methods complement existing procedures based on nonparametric asymptotic approximations by providing an alternative based on exact finite-sample results. These methods should be used only when the local randomization assumption, possibly after transformation of outcomes, is warranted. In these cases, our implementation can be used for both conducting inference in RD designs and offering a robustness check on more conventional methods based on nonparametric local polynomial techniques. Companion R functions are also available from the authors.

9 Acknowledgments

We thank Jake Bowers and David Drukker for useful comments on this project. The authors gratefully acknowledge financial support from the National Science Foundation through grant SES-1357561.

10 References

Calonico, S., M. D. Cattaneo, and R. Titiunik. 2014a. Robust data-driven inference in the regression-discontinuity design. *Stata Journal* 14: 909–946.

———. 2014b. Robust nonparametric confidence intervals for regression-discontinuity designs. *Econometrica* 82: 2295–2326.

———. 2015a. Optimal data-driven regression discontinuity plots. *Journal of the American Statistical Association* 110: 1753–1769.

———. 2015b. rdrobust: An R package for robust nonparametric inference in regression-discontinuity designs. *R Journal* 7: 38–51.

Cattaneo, M. D., B. R. Frandsen, and R. Titiunik. 2015. Randomization inference in the regression discontinuity design: An application to party advantages in the U.S. Senate. *Journal of Causal Inference* 3: 1–24.

Cattaneo, M. D., M. Jansson, and X. Ma. 2016a. rddensity: Manipulation testing based on density discontinuity in R. Working Paper, University of Michigan. http://www-personal.umich.edu/∼cattaneo/software/rddensity/R/rddensity-manual.pdf.

———. 2016b. Simple local regression distribution estimators with an application to manipulation testing. Working Paper, University of Michigan. http://www-personal.umich.edu/∼cattaneo/papers/Cattaneo-Jansson-Ma_2016_LocPolDensity.pdf.

Cattaneo, M. D., R. Titiunik, and G. Vazquez-Bare. 2016. Comparing inference approaches for RD designs: A reexamination of the effect of head start on child mortality. Working Paper, University of Michigan. http://www-personal.umich.edu/∼titiunik/papers/CattaneoTitiunikVazquezBare2015_wp.pdf.

Hahn, J., P. Todd, and W. van der Klaauw. 2001. Identification and estimation of treatment effects with a regression-discontinuity design. *Econometrica* 69: 201–209.

Imbens, G. W., and T. Lemieux. 2008. Regression discontinuity designs: A guide to practice. *Journal of Econometrics* 142: 615–635.

Imbens, G. W., and P. R. Rosenbaum. 2005. Robust, accurate confidence intervals with a weak instrument: Quarter of birth and education. *Journal of the Royal Statistical Society: Series A* 168: 109–126.

Imbens, G. W., and D. B. Rubin. 2015. *Causal Inference for Statistics, Social, and Biomedical Sciences: An Introduction*. New York: Cambridge University Press.

Keele, L., R. Titiunik, and J. R. Zubizarreta. 2015. Enhancing a geographic regression discontinuity design through matching to estimate the effect of ballot initiatives on voter turnout. *Journal of the Royal Statistical Society: Series A* 178: 223–239.

Keele, L. J., and R. Titiunik. 2015. Geographic boundaries as regression discontinuities. *Political Analysis* 23: 127–155.

Lee, D. S. 2008. Randomized experiments from non-random selection in U.S. House elections. *Journal of Econometrics* 142: 675–697.

Lee, D. S., and T. Lemieux. 2010. Regression discontinuity designs in economics. *Journal of Economic Literature* 48: 281–355.

McCrary, J. 2008. Manipulation of the running variable in the regression discontinuity design: A density test. *Journal of Econometrics* 142: 698–714.

Rosenbaum, P. R. 2002. *Observational Studies*. 2nd ed. New York: Springer.

———. 2007. Interference between units in randomized experiments. *Journal of the American Statistical Association* 102: 191–200.

———. 2010. *Design of Observational Studies*. New York: Springer.

Skovron, C., and R. Titiunik. 2015. A practical guide to regression discontinuity designs in political science. Working Paper, University of Michigan. http://www-personal.umich.edu/~titiunik/papers/SkovronTitiunik2015.pdf.

About the authors

Matias D. Cattaneo is an associate professor of economics and an associate professor of statistics at the University of Michigan.

Rocío Titiunik is an associate professor of political science at the University of Michigan.

Gonzalo Vazquez-Bare is a PhD candidate in economics at the University of Michigan.