The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–27

Stata Commands for Full-vector and Subvector Inference in Moment Inequality Models

Woosik Gong
University of Wisconsin at Madison
Madison, WI
wgong28@wisc.edu

Gregory F. Cox
National University of Singapore
Singapore
ecsgfc@nus.edu.sg

Xiaoxia Shi
University of Wisconsin at Madison
Madison, WI
xshi@ssc.wisc.edu

Abstract. In this paper, we present two commands – `ccscc` and `sccintreg`. The `ccscc` command implements two tests: the CC test for moment-(in)equality hypotheses and the sCC test for moment-(in)equality hypotheses with nuisance parameters. The CC test can be used to construct joint confidence set for all the parameters in a moment inequality model, while the sCC test for a subset (subvector) of parameters in such a model. The `sccintreg` command applies the subvector submodule in **ccscc** and computes marginal confidence intervals for each parameter in an interval outcome linear regression model. We demonstrate the use of our commands in two simulation examples.

Keywords: **ccscc**, **sccintreg**, moment inequality models, conditional chi-squared test

1 Introduction

Moment-(in)equality models are an important class of models used in econometrics to address challenges such as missing or interval data, multiple equilibria, and large-scale game theoretical models that are infeasible to solve.[1] However, estimation and inference based on such models are often not easy because their parameters may be partially identified and because there are inequality constraints. Many existing methods involve simulated critical values that are nontrivial to compute and tuning parameters that are hard to choose. Moreover, due to partial identification, estimation and inference for parameters are typically done by test inversion, which requires one to conduct (in)equality testing at all points in a dense enough subset of the parameter space. The number of points to be tested increases exponentially with the parameter dimension. For these reasons, Stata implementation of moment inequality models is uncommon. To our knowledge, there are two Stata journal papers that propose Stata commands for such models, namely, Chernozhukov et al. (2015) who implement the estimation and inference procedure developed by Chernozhukov et al. (2013), and Andrews et al. (2017) who implement the methods developed by Andrews and Shi (2013). Both can be used to calculate joint confidence sets for moment inequality models with an infintie number

1. See the review papers: Canay and Shaikh (2017), Ho and Rosen (2017) and Molinari (2020).

of inequalities. Both also incur high computational cost when there are more than a handful of unknown parameters in the model. For a survey of the current empirical practice, see, for example, Canay et al. (2023).

Cox and Shi (2023b) depart from the literature by providing two computationally easy and tuning-parameter-free tests for moment-(in)equality models. The first is a conditional chi-squared (CC) tests for constructing joint confidence sets for all the unknown parameters (full-vector inference), and the second is a subvector CC (sCC) test for constructing marginal confidence sets for a subset of the uknown parameters (subvector inference) when the rest of the parameters enter the moment functions linearly. One still needs to obtain confidence sets by test inversion, but their proposal simplifies the computation greatly: (a) the CC and the sCC tests use test statistics computable by efficient convex quadratic programming algorithms and critical values from a chi-squared distribution (with a data-dependent degree of freedom), and (b) the sCC test allows one to construct confidence sets of the lower dimensional parameter of interest by searching over only the lower dimensional space. Moreover, the sCC test is further simplified by a closed-form formula for the degree-of-freedom (dof) for the chi-squared distribution. The closed-form formula is derived in Cox et al. (2024).

The computationally easy procedures developed in Cox and Shi (2023b) already have been adopted in a number of papers, including Elliot et al. (2022) for detecting p-hacking, Dickstein et al. (2024) for revealing physician's information about patient costs, and Yuan and Barwick (2024) for studying airline competition in a network environment. These applications showcase the versatility of those procedures. We are thus motivated to provide Stata implementation to further facilitate their adoption. First, we design the `ccscc` command which implements both the CC and the sCC test through a unified syntax. The user can indicate whether they are in a CC test situation or a sCC test situation by specifying different options. When implementing the sCC test, the command uses the closed form formula derived in Cox et al. (2024).

Next, we narrow down the focus to a particular moment-(in)equality model, namely, the interval outcome linear regression model. This model has a similar structure as the standard linear regression model except that the outcome variable is measured in a random interval. It finds applications in linear regressions with missing and interval measured data and in certain aggregate demand models with mismeasured market shares (ref. Gandhi et al. (2023)). In this setting, it is useful to report marginal confidence intervals for each of the regression coefficients, similarly to the `reg` command. We design the `sccintreg` command to do that using the sCC module in the source code of `ccscc`.

We note that there is a Stata command `intreg`, which also deals with linear regression models with an interval-measured outcome. This command estimates a Tobit model that relies on a homoskedastic normal error term and a truncation model structure. Our command `sccintreg` does not require one to make such assumptions. Thus, it can be used in heteroskedastic models, in non-normal models, and in models where the interval measurement of the outcome may be caused by reasons other than truncation.

This article is organized as follows. In section 2, we explain our framework for full-vector and subvector inference for moment-(in)equality models. Section 3 explains the

procedures implemented in detail. Section 4 provides instructions for installing the Stata package for our commands. Sections 5 and 6 explain the syntax and options for `ccscc` and `sccintreg`. Section 7 reports Monte Carlo simulation results. Section 8 presents simulation examples to illustrate how to use our commands. Section 9 concludes.

2 Framework

As summarized above, our `ccscc` command implements the CC test for moment-(in)equality models and the sCC test for a special class of linear conditional moment-(in)equality models. We describe the models in Subsections 2.1 and 2.2 below.

Our `sccintreg` command computes marginal confidence intervals for the coefficients in an interval outcome linear regression model. This model is described in Subsection 2.3 below.

2.1 Moment-(in)equality model: full-vector inference

Let $m(W_i, \theta) = (m_1(W_i, \theta), ..., m_{d_m}(W_i, \theta))'$ be a d_m-dimensional moment function known up to the d_θ-dimensional unknown parameter θ, where $\{W_i\}_{i=1}^n$ is the sample of observables for n units. Let Θ be the parameter space of θ. Consider a model defined by the following moment inequalities:

$$B\mathbb{E}\left[\bar{m}_n(\theta_0)\right] \leq d, \tag{1}$$

where θ_0 is the unknown true value of θ, B is a $d_B \times d_m$ known matrix, d is a $d_B \times 1$ known vector, $\bar{m}_n(\theta) = \frac{1}{n}\sum_{i=1}^n m(W_i, \theta)$, and $\mathbb{E}[\cdot]$ is the expectation with respect to distribution of $\{W_i\}_{i=1}^n$. The matrix B is used to absorb linear dependence across the inequalities, for example, to write a moment equality into a pair of opposing moment inequalities. See Cox and Shi (2023b) for details.

The parameter θ may or may not be point identified by the moment inequalities. The identified set of θ is defined as

$$\Theta_0 = \{\theta \in \Theta : B\mathbb{E}\left[\bar{m}_n(\theta)\right] \leq d\}.$$

Since Θ_0 may contain points other than the true value θ_0, it may not be possible to consistently estimate θ_0. However, we can still construct a confidence set for θ_0 by test inversion. That is, at each given value of $\theta \in \Theta$, we test the null hypothesis

$$H_0 : B\mathbb{E}\left[\bar{m}_n(\theta)\right] \leq d, \tag{2}$$

and collect all the θ values at which this hypothesis is not rejected. Mathematically, the $100(1-\alpha)\%$-CS of θ_0 is defined by

$$CS_n(1-\alpha) = \{\theta \in \Theta : T_n(\theta) < c_{n,\alpha}(\theta)\}, \tag{3}$$

where $\alpha \in (0, 1)$ is the nominal significance level, $T_n(\theta)$ is a test statistic, and $c_{n,\alpha}(\theta)$ is the critical value. We say that the confidence set is asymptotically uniformly valid if

the probability that $CS_n(1-\alpha)$ covers θ_0 converges to a value no smaller than $1-\alpha$ uniformly over all $\theta \in \Theta_0$ and over a reasonable set of data generating processes. More discussions of uniform asymptotic validity can be found in Canay and Shaikh (2017) and in Cox and Shi (2023b).

2.2 Conditional moment-(in)equality model: subvector inference

Suppose that in addition to the parameter of interest θ, the model contains another unknown parameter vector $\delta \in \mathbb{R}^{d_\delta}$ that is not of central interest at the moment. Additionally, suppose that there are exogenous variables Z_i which we call instruments. The instruments Z_i are part of the vector of observables W_i.

Let Z denote the n observations of Z_i: $Z := \{Z_i\}_{i=1}^n$.

We consider models defined by the following conditional moment inequalities:

$$B_Z \mathbb{E}\left[\bar{m}_n(\theta_0)|Z\right] - C_Z\delta_0 \leq d_Z, \text{ a.s,} \tag{4}$$

where B_Z is a $d_B \times d_m$ matrix, C_Z is a $d_B \times d_\delta$ matrix, d_Z is a $d_B \times 1$ vector, $\bar{m}_n(\theta) = \frac{1}{n}\sum_{i=1}^n m(W_i,\theta)$, $\mathbb{E}[\cdot|Z]$ denotes expectation with respect to $\{W_i\}_{i=1}^n$ conditional on $Z = \{Z_i\}_{i=1}^n$, $(\theta_0', \delta_0')'$ is the unknown true parameter value, and "a.s." in (4) means that the inequality in (4) holds almost surely with respect to the distribution of Z. The quantities B_Z, C_Z, and d_Z are known and may depend on $\{Z_i\}_{i=1}^n$ and θ_0.

Comparing to the model in (1), the model in (4) is more restrictive in two ways: First, the moment inequalities in (4) are restricted to be linear in a subset of the unknown parameters–the δ parameters, and the δ parameters have to be coefficients of the exogenous variables $\{Z_i\}$, while those in (1) can depend on the whole set of parameters arbitrarily. Second, there needs to be exogenous instruments Z_i and the moment inequalities need to hold conditionally given $\{Z_i\}_{i=1}^n$ in model (4), while the moment inequalities only need to hold unconditionally in model (1).

Similar to the full-vector inference case, the parameter $(\theta_0', \delta_0')'$ may be partially identified. Here since we want to construct a marginal confidence set (MCS) for θ, we do so by inverting a test for

$$H_0 : \exists \delta \in \mathbb{R}^{d_\delta} \text{ s.t. } B_Z \mathbb{E}\left[\bar{m}_n(\theta)|Z\right] - C_Z\delta \leq d_Z, \text{ a.s,} \tag{5}$$

for a given value $\theta \in \Theta$. The MCS can be written as

$$MCS_n(1-\alpha) = \{\theta \in \Theta : T_n(\theta) < c_{n,\alpha}(\theta)\}, \tag{6}$$

where $T_n(\theta)$ is a test statistic for the null hypothesis (5), and $c_{n,\alpha}(\theta)$ is a critical value of nominal significance level α. When the quantities B_Z, C_Z, and d_Z depend on θ_0, one simply plugs the given θ value in them to form the null hypothesis (5).

2.3 Interval Outcome Linear Regression Model

We now describe the interval outcome linear regression model, which is an example of the conditional moment inequality model in (4). Consider the linear regression model

$$Y_i^* = X_i'\beta_0 + \epsilon_i,\ \mathbb{E}[\epsilon_i|\{X_i\}_{i=1}^n] = 0, \tag{7}$$

where $Y_i^* \in \mathbb{R}$ is a dependent variable, and $X_i \in \mathbb{R}^{d_\beta}$ is a vector of exogenous regressors, and $\beta_0 \in \mathbb{R}^{d_\beta}$ is the unknown true parameter value. Suppose that the outcome Y_i^* is not observed. Instead, we observe an upper bound variable Y_{Ui} and a lower bound variable Y_{Li}, and we have the background knowledge that

$$\mathbb{E}[Y_i^*|X_i] \in [\mathbb{E}[Y_{Li}|X_i], \mathbb{E}[Y_{Ui}|X_i]]. \tag{8}$$

The restrictions in (7) and (8) imply an infinite number of moment inequalities if X_i contains continuous variables and imply k moment inequalities if X_i is discrete and can take k values. Constructing tests using only the moment functions $Y_{Li} - X_i'\beta$ and $Y_{Ui} - X_i'\beta$ would be inefficient. Instead, we boost the efficiency by choosing a vector of instrumental functions $\mathcal{I}(X_i)$, which is a nonnegative vector-valued function of X_i. We discuss the construction of $\mathcal{I}(X_i)$ in a later section. Using the instrumental functions, we obtain the moment inequalities:

$$\mathbb{E}\left[n^{-1}\sum_{i=1}^n \begin{pmatrix} Y_{Li}\mathcal{I}(X_i) \\ -Y_{Ui}\mathcal{I}(X_i) \end{pmatrix} - n^{-1}\sum_{i=1}^n \begin{pmatrix} \mathcal{I}(X_i)X_i' \\ -\mathcal{I}(X_i)X_i' \end{pmatrix}\beta_0 \middle| \{X_i\}_{i=1}^n\right] \leq \mathbf{0}. \tag{9}$$

We can use the sCC test to construct marginal confidence intervals for each element of β. For example, suppose we want to construct a confidence interval for the jth element of β, β_j. Let β_{-j} denote β with its jth element removed. Then, β_j is the parameter of interest and β_{-j} the nuisance parameter, and they are respectively denoted θ and δ in the notation of Section 2.2. Now let $X_{j,i}$ denote the jth element of X_i, and let $X_{-j,i}$ denote X_i with its jth element removed. Then, (9) can be written as

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^n \begin{pmatrix} (Y_{Li}-\theta_0 X_{j,i})\mathcal{I}(X_i) \\ -(Y_{Ui}-\theta_0 X_{j,i})\mathcal{I}(X_i) \end{pmatrix} - \frac{1}{n}\sum_{i=1}^n \begin{pmatrix} (\mathcal{I}(X_i)X_{-j,i}') \\ -(\mathcal{I}(X_i)X_{-j,i}') \end{pmatrix}\delta_0 \middle| \{X_i\}_{i=1}^n\right] \leq \mathbf{0}. \tag{10}$$

It is easy to see that this is a special case of (4) with $\{Z_i\} = \{X_i\}$, $W_i = (Y_{Li}, Y_{Ui}, X_i')'$, $B_Z = I$, $m(W_i,\theta) = \begin{pmatrix} (Y_{Li}-\theta X_{j,i})\mathcal{I}(X_i) \\ -(Y_{Ui}-\theta X_{j,i})\mathcal{I}(X_i) \end{pmatrix}$, $C_Z = \frac{1}{n}\sum_{i=1}^n \begin{pmatrix} (\mathcal{I}(X_i)X_{-j,i}') \\ -(\mathcal{I}(X_i)X_{-j,i}') \end{pmatrix}$, and $d_Z = \mathbf{0}$. Therefore, a confidence interval of β_j can be obtained by inverting a test for (5).

3 Description of the Procedures

In this section, we explain the CC test and the sCC test implemented by the command `ccscc` and explain the confidence interval construction implemented in `sccintreg`.

In models other than the interval outcome linear regression model, one cannot use `sccintreg` to calculate the marginal confidence interval for each unknown parameter.

Instead, a joint confidence set of θ can be obtained by constructing a grid of θ to run `ccscc` over and collecting θ's that the test does not reject. Marginal confidence intervals for an element of θ can be obtained by finding the lowest and the highest values of that element that correspond to a θ value in the joint confidence set.

3.1 CC test: full-vector test

The test statistic for the null hypothesis in (2) is

$$T_n(\theta) = \min_{\mu: B\mu \leq d} n(\bar{m}_n(\theta) - \mu)' \widehat{\Sigma}_n(\theta)^{-1} (\bar{m}_n(\theta) - \mu), \tag{11}$$

where $\widehat{\Sigma}_n(\theta)$ is an estimator of $\text{Var}(\sqrt{n}\bar{m}_n(\theta))$. When $\{W_i\}_{i=1}^n$ are independent and identically distributed (i.i.d.) across i, a natural choice is

$$\widehat{\Sigma}_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} (m(W_i, \theta) - \bar{m}_n(\theta))(m(W_i, \theta) - \bar{m}_n(\theta))'. \tag{12}$$

When $\{W_i\}_{i=1}^n$ are not i.i.d. across i, users can plug in $\widehat{\Sigma}_n(\theta)$ obtained by a different formula to take into account cluster dependence, autocorrelation, or other types of dependence.

Let $\hat{\mu}$ be the solution to the minimization problem (11). Let b_j' denote the jth row of B and d_j denote the jth element of d for $j = 1, ..., d_B$. Define the set of indices for active constraints of the minimization problem (11) to be[2]

$$\widehat{J} = \{j \in \{1, ..., d_B\} : b_j' \hat{\mu} = d_j\}.$$

For any subset $J \subset \{1, ..., d_B\}$, let B_J be the submatrix of B that is formed by all the rows of B corresponding to the indices in J. Let $\hat{r} = \text{rk}(B_{\widehat{J}})$ be the rank of the matrix $B_{\widehat{J}}$. Then, the critical value $c_{n,\alpha}(\theta)$ in (3) is defined to be $\chi^2_{\hat{r},1-\alpha}$ which is the $100(1-\alpha)\%$ quantile of the chi-squared distribution with $\hat{r}$ degrees of freedom. Note that $\hat{\mu}$, $\widehat{J}$, and $\hat{r}$ are obtained for each θ and thus they depend on θ. The dependence is suppressed for simpler notation.

The critical value $c_{n,\alpha}(\theta)$ sometimes can be refined (reduced) when $\hat{r} = 1$ so that the test has more accurate size. Section A.1 of Cox and Shi (2023b) describes the refinement in full details. We implement the refinement step in the `ccscc` command but omit the description here.

3.2 SCC test: subvector test

The test statistic for the null hypothesis in (5) is

$$T_n(\theta) = \min_{\mu, \delta: B_Z \mu - C_Z \delta \leq d_Z} n(\bar{m}_n(\theta) - \mu)' \widehat{\Sigma}_n(\theta)^{-1} (\bar{m}_n(\theta) - \mu), \tag{13}$$

2. Typically, due to numerical imprecision of computer algorithms, a tolerance level (*tol*) is used when determining which inequality is active. If $b_j' \hat{\mu} - d_j \geq -tol$, the jth inequality is considered active.

where $\widehat{\Sigma}_n(\theta)$ is an estimator of $\text{Var}(\sqrt{n}\bar{m}_n(\theta)|Z)$. When the data are i.i.d. across i and Z_i is a discrete random variable with a finite support set $\mathcal{Z}$, a natural estimator of $\text{Var}(\sqrt{n}\bar{m}_n(\theta)|Z)$ is

$$\widehat{\Sigma}_n(\theta) = \sum_{\ell \in \mathcal{Z}} \frac{n_\ell}{n} \frac{1}{n_\ell - 1} \sum_{i=1}^{n} (m(W_i, \theta) - \bar{m}_n^\ell(\theta))(m(W_i, \theta) - \bar{m}_n^\ell(\theta))' \mathbb{I}(Z_i = \ell), \tag{14}$$

where $\mathbb{I}(E)$ is the dummy variable that equals 1 if event E occurs and equals 0 otherwise, $n_\ell = \sum_{i=1}^n \mathbb{I}(Z_i = \ell)$, and $\bar{m}_n^\ell(\theta) = \frac{1}{n_\ell} \sum_{i=1}^n m(W_i, \theta)\mathbb{I}(Z_i = \ell)$. The proof of the consistency of this estimator is in Theorem 6(b) in Section B of Cox and Shi (2023a).

When the data are i.i.d. across i and Z_i is a continuous random variable, we can use a nearest neighbor matching estimator similar to the one adopted by Abadie et al. (2014). To define this estimator, let $\widehat{\Sigma}_{Z,n} = n^{-1} \sum_{i=1}^n (Z_i - \bar{Z}_n)(Z_i - \bar{Z}_n)'$ where $\bar{Z}_n = n^{-1} \sum_{i=1}^n Z_i$. For each i, let the nearest neighbor be $\ell_Z(i) = \arg\min_{j \in \{1,\ldots,n\}, j \neq i} (Z_i - Z_j)' \widehat{\Sigma}_{Z,n}^{-1} (Z_i - Z_j)$, and $\ell_Z(i)$ picks one randomly when the argmin is not unique. Then, the nearest neighbor matching estimator of $\text{Var}(\sqrt{n}\bar{m}_n(\theta)|Z)$ is

$$\widehat{\Sigma}_n(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (m(W_i, \theta) - m(W_{\ell_Z(i)}, \theta))(m(W_i, \theta) - m(W_{\ell_Z(i)}, \theta))'. \tag{15}$$

The proof of the consistency of this estimator is in Theorem 6(c) in Section B of Cox and Shi (2023a). Users can also plug in $\widehat{\Sigma}_n(\theta)$ obtained by a different formula to take into account clustering, autocorrelation, or other types of dependence.

Let $(\hat{\mu}', \hat{\delta}')'$ be the solution to the minimization problem (13). Let $b_{j,Z}'$ and $c_{j,Z}'$ be the jth rows of B_Z and C_Z, respectively, and $d_{j,Z}$ be the jth element of d_Z. Define

$$\widehat{J} = \{j \in \{1, ..., d_B\} : b_{j,Z}' \hat{\mu} - c_{j,Z}' \hat{\delta} = d_{j,Z}\},$$

which is set of indices of the inequalities that are active for the minimization problem (13).[3] Let $B_{\widehat{J}}$ and $C_{\widehat{J}}$ be the submatrices of B_Z and C_Z, respectively, which contain the jth rows of B_Z and C_Z if and only if $j \in \widehat{J}$. To obtain the critical value of the sCC test, let

$$\hat{r} = \text{rk}\left(\left[\begin{array}{c|c} B_{\widehat{J}} & C_{\widehat{J}} \end{array}\right]\right) - \text{rk}\left(C_{\widehat{J}}\right), \tag{16}$$

where $\left[\begin{array}{c|c} B_{\widehat{J}} & C_{\widehat{J}} \end{array}\right]$ is the matrix obtained by concatenating the matrices $B_{\widehat{J}}$ and $C_{\widehat{J}}$ side-by-side. The critical value is then

$$c_{n,\alpha}(\theta) = \chi^2_{\hat{r}, 1-\alpha}, \tag{17}$$

which is the $100(1-\alpha)\%$ quantile of the chi-squared distribution with $\hat{r}$ degrees of freedom. Note that $\widehat{J}$ and $\hat{r}$ depend on θ, although the dependence is suppressed for simpler notation.

3. Due to the numerical imprecision of quadratic programming algorithms, we use a tolerance level(tol) to determine which inequalities are active. We say that the jth inequality is active if $b_{j,Z}'\hat{\mu} - c_{j,Z}'\hat{\delta} - d_{j,Z} \geq -tol$.

3.3 Marginal Confidence Intervals for Interval Outcome Regression

Now we specialize to the interval outcome linear regression model defined in (7) and (8) and describe the construction of marginal confidence intervals for each regression coefficient.

In broad strokes, the task is straightforward. To construct a marginal confidence interval for the jth element, β_j, of β, we first realize that the parameter of interest θ is β_j and the nuisance parameter δ is β_{-j}. The following steps will give us the desired confidence interval:

Step 1. Define B_Z, $m(W_i, \theta)$, C_Z, and d_Z according to the two lines below (10).

Step 2. Define the test statistic $T_n(\theta)$ and the critical value $c_{n,\alpha}(\theta)$ as functions of θ according to (13) and (17).

Step 3. Find $\hat{\theta}_U(\alpha) = \max\{\theta \in \mathbb{R} : T_n(\theta) \leq c_{n,\alpha}(\theta)\}$ and $\hat{\theta}_L(\alpha) = \min\{\theta \in \mathbb{R} : T_n(\theta) \leq c_{n,\alpha}(\theta)\}$.

Then, $[\hat{\theta}_L(\alpha), \hat{\theta}_U(\alpha)]$ is the marginal confidence interval for β_j with nominal confidence level $1 - \alpha$.

The broad-stroke steps are applicable to all settings to which the sCC test is applicable. However, important implementation details are left unspecified. Filling in those details is a model-specific task, which we do for the interval outcome linear regression model now:

- First, to define $m(W_i, \theta)$ and C_Z, one needs the vector of instrumental functions $\mathcal{I}(X_i)$. We allow different ways to generate $\mathcal{I}(X_i)$ from X_i. By default, we generate binarized variable $\widetilde{X}_i = (\mathbb{I}(X_{1,i} > x_1^{med}), ..., \mathbb{I}(X_{d_\beta,i} > x_{d_\beta}^{med}))'$, where x_j^{med} is the sample median of $\{X_{j,i}\}_{i=1}^n$ for $j \in \{1, ..., d_\beta\}$. Then, we let $\mathcal{I}(X_i) = (\mathbb{I}(\widetilde{X}_i = \tilde{x}_1), ..., \mathbb{I}(\widetilde{X}_i = \tilde{x}_\mathcal{X}))' \in \mathbb{R}^\mathcal{X}$, where $\{\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_\mathcal{X}\}$ is the support of $\widetilde{X}_i$.

  If X_i takes only finite number of values, say $\{x_1, x_2, \ldots, x_M\}$, the user can specify the "discrete" option. When this option is specified, $\mathcal{I}(X_i)$ is defined to be $(\mathbb{I}(X_i = x_1), \ldots, \mathbb{I}(X_i = x_M))' \in \mathbb{R}^M$.

  We also allow the user to directly input their preferred $\mathcal{I}(X_i)$ as a variable list in the option "instrument". The user should make sure that $\mathcal{I}(X_i)$ are nonnegative-valued functions of X_i.

  The options are explained in more details in Section 6.3 below. See Cox and Shi (2023b) for more guidance on constructing $\mathcal{I}(X_i)$.

- Second, in Step 3, the command carries out a constrained maximization and a constrained minimization to obtain $\hat{\theta}_U(\alpha)$ and $\hat{\theta}_L(\alpha)$, respectively. This could be done accurately in a variety of ways since θ is one-dimensional. By default, we use a bisection algorithm, which goes as follows (for a MCI with nominal coverage probability $1 - \alpha$):

1. Set a lower bound $\underline{\theta}$ and an upper bound $\overline{\theta}$ for θ. They are set to prevent the iterations below from going on indefinitely. One should set them reasonably wide so that they do not meaningfully affect the result. By default, $\overline{\theta} = 100$, and $\underline{\theta} = -100$ in our command.

2. Compute $\hat{\theta}_0 = \arg\min_{\theta \in R} \min_{\mu,\delta:\mu - C_Z\delta \leq d_Z} n(\bar{m}_n(\theta) - \mu)' \widehat{\Sigma}_n^{-1} (\bar{m}_n(\theta) - \mu)$. This is our first guess for a point inside the marginal confidence interval (MCI). Note that $\widehat{\Sigma}_n(\theta)$ does not depend on θ in this model as it is an estimator of the conditional variance-covariance of $\frac{1}{\sqrt{n}} \sum_{i=1}^{n} \begin{pmatrix} \mathcal{I}(X_i) Y_{Li} \\ -\mathcal{I}(X_i) Y_{Ui} \end{pmatrix}$ given $\{X_i\}_{i=1}^n$. If $\hat{\theta}_0$ is not between $\underline{\theta}$ and $\overline{\theta}$, we set $\hat{\theta}_0$ to be the midpoint $(\underline{\theta} + \overline{\theta})/2$.

   We verify whether $\hat{\theta}_0$ is in fact inside the MCI by testing (5) with $\theta = \hat{\theta}_0$ using the sCC test with nominal level α. If the test does not reject, we have found $\hat{\theta}_0$ to be a point inside the MCI. Then we proceed to the next step to find the point below the lower end point and that above the upper end point of the MCI.

   If the sCC test rejects (5) with $\theta = \hat{\theta}_0$, we attempt again to find a point in the MCI via a grid search. That is, we carry out a nominal level α sCC test for (5) at $\theta = \hat{\theta}_0 + q \times 10^{-1} \cdot c_{rel}(\overline{\theta} - \hat{\theta}_0)$ and $\theta = \hat{\theta}_0 - q \times 10^{-1} \cdot c_{rel}(\hat{\theta}_0 - \underline{\theta})$ for $q = 1, 2, \ldots, \lfloor 10(c_{rel})^{-1} \rfloor$ until we find a point that is not rejected or q reaches $\lfloor 10(c_{rel})^{-1} \rfloor$ whichever comes first. Here $\lfloor x \rfloor$ stands for the largest integer that is no greater than x. If a point of non-rejection is found, we replace $\hat{\theta}_0$ with the value of this point and proceed to the next step. Otherwise, our command reports that the MCI is empty and proceeds no further, in which case, one can try again with a wider $[\underline{\theta}, \overline{\theta}]$ or a reduced c_{rel}.

   Here, c_{rel} is a scaling factor for the step size of the search and we call it the relative step size. By default, our command sets $c_{rel} = 0.1$. The user can specify these values in options as described in Section 6.3 below.

3. To find a point below the lower end point of the MCI. We test (5) with $\theta = \hat{\theta}_0 - (q \times c_{rel} |\underline{\theta} - \hat{\theta}_0|)$ sequentially for $q = 1, 2, ...$ until we find a point where the sCC test rejects, or until we reach the lower bound, $\underline{\theta}$ of the parameter space for θ, whichever comes first. Denote the value of θ where we stop by $\hat{\theta}_{LL}$. Let $\hat{\theta}_{LU} = \hat{\theta}_{LL} + c_{rel} |\underline{\theta} - \hat{\theta}_0|$, which is an upper bound of $\hat{\theta}_L(\alpha)$ since the test is not rejected at $\theta = \hat{\theta}_{LU}$ and hence $\hat{\theta}_{LU}$ is inside the MCI.

4. If $\hat{\theta}_{LL} = \underline{\theta}$, let $\hat{\theta}_L(\alpha) = \underline{\theta}$.

5. Otherwise, test (5) at $\theta = (\hat{\theta}_{LU} + \hat{\theta}_{LL})/2$ using the sCC test.

6. If the sCC test rejects $\theta = (\hat{\theta}_{LU} + \hat{\theta}_{LL})/2$, replace the value of $\hat{\theta}_{LL}$ with $(\hat{\theta}_{LU} + \hat{\theta}_{LL})/2$, which is a new point below $\hat{\theta}_L(\alpha)$. Note that each time the value of $\hat{\theta}_{LL}$ is changed, it moves closer to $\hat{\theta}_L(\alpha)$.

   If the sCC test does not reject $\theta = (\hat{\theta}_{LU} + \hat{\theta}_{LL})/2$, replace the value of $\hat{\theta}_{LU}$ with $\theta = (\hat{\theta}_{LU} + \hat{\theta}_{LL})/2$, which is a new point inside the MCI. Note that each time the value of $\hat{\theta}_{LU}$ is changed, it also moves closer to $\hat{\theta}_L(\alpha)$.

7. Repeat the last two steps until $|\hat{\theta}_{LU} - \hat{\theta}_{LL}|$ is below a precision level (*prec*, by default, $prec = 10^{-8}$), at which time let $\hat{\theta}_L(\alpha) = \hat{\theta}_{LL}$. Note that each time we repeat steps 4 and 5, the distance $|\hat{\theta}_{LU} - \hat{\theta}_{LL}|$ is halved. Thus, the algorithm converges exponentially fast.

8. Analogously work on the upper end point and find $\hat{\theta}_U(\alpha)$.

The interval $[\hat{\theta}_L(\alpha), \hat{\theta}_U(\alpha)]$ is the calculated MCI of nominal coverage probability $1-\alpha$.

Another method for constructing the confidence intervals of the coefficients is doing grid search. An equidistant grid of the parameter $\{\theta_1, \theta_2, ..., \theta_M\}$, $\underline{\theta} = \theta_1 \leq \theta_2 \leq ... \leq \theta_M = \overline{\theta}$, is constructed while M is set by users. Check Section 6.3 for more details on how the grid is constructed. An MCI can be obtained by implementing the sCC test at each θ value on the grid and collect the values at which the test does not reject. The grid search approach takes longer time than the bisection one above to reach the same precision. On the other hand, grid search computes $T_n(\theta)$ and $c_{n,\alpha}(\theta)$ on an equidistant grid, which can be useful for plotting, as shown in Figure 2 of Section 8.3.

4 Installation of the ccscc Package

The Stata commands proposed in this paper are available in the Statistical Software Components (SSC) archive. Our Stata package, `ccscc`, can be installed from within Stata by typing `ssc install ccscc` or `ssc install ccscc, all`. The former installs the commands and the help files, while the latter installs those as well as the ancillary data file that allows the user to run the examples in the help files.

The commands `ccscc` and `sccintreg` use Stata/Python integration for quadratic programming. Python and its CVXOPT (2023) package should be installed before using our commands because there currently is no Mata package that solves a convex quadratic programming problem such as that in (13), and we use the python package CVXOPT for that task. The included help files explain how to install python and the CVXOPT (2023) package. The instructions on how to set up the python environment can be found in the Stata manual [P] **PyStata integration** or on the official website of Stata.

5 The ccscc command

5.1 Syntax

The syntax of `ccscc` is as follows:

`ccscc` (*instruments*) *moments* [*if*] [*in*] [, `alpha(`*real* 0.05`)` `matineq(`*matname*`)`
`vecineq(`*matname*`)` `matnuisa(`*matname*`)` `ivdiscrete` `refined`
`sigma(`*matname*`)` `activetol(`*real 1e-05*`)` `qpabstol(`*real 1e-08*`)` `qpreltol(`*real*

1e-08) `qpfeastol(`*real 1e-08*`)`]

5.2 Description

To implement `ccscc` for the CC test described in Section 3.1, one first generates Stata variables $m_1(W_i, \theta)$, ..., $m_{d_m}(W_i, \theta)$ for observations $i = 1, ..., n$ and input the names of these variables in the variable list *moments*. The variable list *instruments* should be empty. The options `matnuisa` and `ivdiscrete` should not be specified.

To implement `ccscc` for the sCC test described in Section 3.2, one also generates Stata variables $m_1(W_i, \theta)$, ..., $m_{d_m}(W_i, \theta)$ for observations $i = 1, ..., n$, and input the names of these variables in the variable list *moments*. The variable list *instruments* should not be empty. Instead, it should contain the variable names for the variables in Z_i. The option `matnuisa` should be specified and its argument should be the matrix C_Z. The option `refined` should not be specified.

5.3 Options

`alpha(`*real* 0.05`)` specifies the size of the test. By default, it is set to 0.05.

`matineq(`*matname*`)` specifies the matrix B in (1) and B_Z in (4). Inside the parentheses, *matname* should be the name of a Stata $d_B \times d_m$ matrix. If this option is not specified, B or B_Z is taken to be the $d_m \times d_m$ identity matrix.

`vecineq(`*matname*`)` specifies the vector d in (1) and d_Z in (4). Inside the parentheses, *matname* should be the name of a Stata $d_B \times 1$ matrix. If this option is not specified, d or d_Z is taken to be a d_B-dimension zero vector.

`matnuisa(`*matname*`)` specifies C_Z in (4). It needs and only needs to be specified for subvector inference. Inside the parentheses, *matname* should be the name of a Stata $d_B \times d_\delta$ matrix.

`ivdiscrete` specifies that the instruments are discrete. This option affects how $\widehat{\Sigma}_n(\theta)$ is calculated as explained below. It should not be specified for full-vector inference.

`refined` adjusts the critical value to make the size of the CC test more accurate. Details of it is provided in Section A.1 of Cox and Shi (2023a). The refined sCC test is not implemented in the present code package, and thus this option should not be used when doing subvector inference.[4]

`sigma` specifies a user-defined $\widehat{\Sigma}_n(\theta)$. Its argument *matname* should be the name of a Stata $d_m \times d_m$ positive-definite matrix. If this option is not specified, the command calculate $\widehat{\Sigma}_n(\theta)$ according to (12) for full-vector inference, and according to (15) for subvector inference if `ivdiscrete` is not specified and according to formula (14) otherwise.

4. The reason that the refined sCC test is not implemented is that implementing it would require a vertex enumeration algorithm for polyhedrons, which is currently not available in MATA.

`activetol` specifies the tolerance level used to determine which inequalities are active as explained in Sections 3.1 and 3.2. By default, it is set to 10^{-5}.

`qpabstol`, `qpreltol`, `qpfeastol` sets 'abstol', 'reltol' and 'feastol' for quadratic programming via CVXOPT (2023). See its documentation for precise descriptions.

5.4 Saved Results

`ccscc` stores the following results in `r()`:

Scalars

`r(N)`	number of observations	`r(alpha)`	nominal significance level of the test
`r(reject)`	1 if the test is rejected, 0 if not	`r(rank)`	$\hat{r}$ that determines the critical value of the test
`r(cv)`	critical value for the test, $\chi^2_{\hat{r},1-\alpha}$	`r(t_stats)`	test statistic, $T_n(\theta)$
`r(pval)`	p-value for the test		

Matrix

`r(Sigmainv)`	the inverse of estimated (conditional) variance -covariance matrix of moments, $(\widehat{\Sigma}_n(\theta))^{-1}$

Macros

`r(cmd)`	`ccscc`	`r(title)`	"Conditional Moment Inequalities Test"

6 The sccintreg command

6.1 Syntax

The syntax of `sccintreg` is as follows:

`sccintreg` *y_low y_up exog* [*if*] [*in*] [, `alpha(`*real* 0.05`)` `instrument(`*varname*`)`
`subset(`*varname*`)` `discrete` `noconstant` `sigma(`*matname*`)` `ub(`*matname*`)`
`lb(`*matname*`)` `precision(`*real 1e-08*`)` `relstep(`*real 0.1*`)` `activetol(`*real 1e-05*`)`
`qpabstol(`*real 1e-08*`)` `qpreltol(`*real 1e-08*`)` `qpfeastol(`*real 1e-08*`)`
`gridsearch(`*integer 1*`)`]

6.2 Description

To implement `sccintreg`, one needs to have ready the bounding variables Y_{Li}, Y_{Ui}, and the regressors in X_i, and input the variable names of these variables in the variable lists *y_low*, *y_up*, and *exog*, respectively.

6.3 Options

`alpha(`*real* 0.05`)` sets the coverage rate of each confidence interval to $100(1-\alpha)\%$. By default, it is set to 0.05.

`instrument` specifies the user-defined nonnegative instrument functions $\mathcal{I}(X_i)$ in (10). Inside the parentheses, one should input a Stata variable list. For example, one can use `instrument("Z1 Z2 Z3")` if "Z1", "Z2", and "Z3" are the Stata variable names of the instruments $\{\mathcal{I}_1(X_i)\}_{i=1}^n$, $\{\mathcal{I}_2(X_i)\}_{i=1}^n$, and $\{\mathcal{I}_3(X_i)\}_{i=1}^n$, respectively, when $\mathcal{I}(X_i) = (\mathcal{I}_1(X_i), \mathcal{I}_2(X_i), \mathcal{I}_3(X_i))'$. If this option is not specified, default instruments are constructed according to the description in Section 3.3.

`subset(`*varnames*`)` specifies the subset of regressors that the user wants to construct MCIs for. Inside the parentheses, one inputs a variable list that is a subset of the regressor list *exog* unless the MCI of the intercept is of interest. For example, `subset("X1 X2")` instructs Stata to report MCIs of the coefficients for variables "X1" and "X2". To include intercept, add `"cons"` to `subset`. For instance, `subset("cons X1")` returns MCIs of intercept and the coefficient for "X1". The order of variable list in `subset` is irrelevant in a sense that MCIs in the outcome table are always sorted in the same order as *exog* and the intercept always come first. For example, if *exog* is `X1 X2 X3` and `subset("X3 cons X1")` is specified as option, then MCIs in the outcome table is ordered as: intercept, coefficient for "X1", and coefficient for "X3". If this option is not specified, MCIs for all the coefficients are reported.

`discrete` specifies that regressors are discrete. Specifying this option may have two effects: First, $\widehat{\Sigma}_n(\theta)$ will be computed according to (14) unless the user manually sets the variance-covariance matrix using the option `sigma`; Second, $\mathcal{I}(X_i)$ will be constructed according to the discrete case described in Section 3.3 unless the user manually sets the instruments using the option `instrument`.

`noconstant` instructs Stata that the constant term be excluded from the regression.

`sigma` specifies a user-defined $\widehat{\Sigma}_n(\theta)$. Its argument *matname* should be the name of a Stata $d_m \times d_m$ positive-definite matrix. If this option is not specified, the command calculates $\widehat{\Sigma}_n(\theta)$ according to (15) if `discrete` is not specified and according to (14) otherwise.

`ub(`*matname*`)` is a vector consisting of $\overline{\theta}$ in Section 3.3. It specifies an upper bound vector for β_{sub}, where β_{sub} is a subvector of the full-coefficients β that includes and only includes the coefficients for variables specified by the option `subset`. If users do not specify `subset`, β_{sub} is just β. Be mindful that the full-vector β includes an intercept unless the `noconstant` option is specified, and hence its dimension is the number of variables in *exog* added by 1 in this case. The argument *matname* should be the name of a Stata $1 \times d_{\beta_{\text{sub}}}$ matrix, where $d_{\beta_{\text{sub}}}$ is the dimension of β_{sub}. By default, it is a $d_{\beta_{\text{sub}}}$-dimensional vector whose elements are all 100.

`lb(`*matname*`)` is a vector consisting of $\underline{\theta}$ in Section 3.3. It specifies a lower bound vector for β_{sub}, where β_{sub} is explained above. The argument *matname* should be a Stata

$1 \times d_{\beta_{\text{sub}}}$ matrix. By default, it is a $d_{\beta_{\text{sub}}}$-dimensional vector whose elements are all -100.

`precision(`*real* 1e-08`)` specifies the constant *prec* in the bisection algorithm explained in Section 3.3. By default, it is 10^{-8}.

`relstep` specifies the relative step size c_{rel} in the bisection algorithm explained in Section 3.3. It should be a real number between 0 and 1. By default, it is set to 0.1.

`activetol` specifies the tolerance level used to determine which inequalities are active as explained in Section 3.2. By default, it is set to 10^{-5}.

`qpabstol`, `qpreltol`, `qpfeastol` sets 'abstol', 'reltol' and 'feastol' for quadratic programming via CVXOPT (2023). See its documentation for precise description.

`gridsearch(`*int* 1`)` instructs Stata to use the grid search algorithm instead of the bisection algorithm to find the end points of the CI. Its value should be the number of grid points. If $\underline{\beta}$ and $\overline{\beta}$ denote `lb` and `ub`, respectively, and n_g is the number of grid points specified by `gridsearch`, then the resulting grid for β_j, the jth coefficient of β_{sub}, becomes $\{\underline{\beta}_j + k(\overline{\beta}_j - \underline{\beta}_j)/(n_g - 1) : k = 0, ..., n_g - 1\}$, where $\underline{\beta}_j$ and $\overline{\beta}_j$ are the jth elements of $\underline{\beta}$ and $\overline{\beta}$, respectively. If the user accidentally sets the argument of this option to 1, then the bisection algorithm is used instead of the grid search.

6.4 Saved Results

`sccintreg` stores the following in `r()`:

Scalars			
`r(N)`	number of observations	`r(alpha)`	nominal significance level
Matrix			
`r(table)`	table of the marginal CIs	`r(Tns)`	test statistics evaluated at each grid point (available only if `gridsearch` is used)
`r(CVs)`	critical values evaluated at each grid point (available only if `gridsearch` is used)	`r(TESTs)`	test results at each grid point, 1 if rejected and 0 otherwise (available only if `gridsearch` is used)
`r(GRIDs)`	grid of parameter that is used (available only if `gridsearch` is used)		
Macros			
`r(cmd)`	`sccintreg`	`r(title)`	"Conditional Moment Inequalities Tests for Linear Regression with Interval Outcome"

7 Monte Carlo simulations

In this section, we test the command on the simulation examples from Cox and Shi (2023b). Specifically, we run Monte Carlo experiments using both our `ccscc` command and the Matlab code of Cox and Shi (2023b), and then compare the results. For the

sCC test, we adjusted the Matlab code to compute critical values using the $\hat{r}$ formula derived in Cox et al. (2024) because this is how $\hat{r}$ is calculated in the `ccscc` command. The number of Monte Carlo repetitions is 1000, and the significance level for the CC and sCC tests is 5%.

For full-vector inference, we use the simulation example from Andrews and Barwick (2012), which is also implemented in Cox and Shi (2023b). In this example, the inequalities tested are

$$\mathbb{E}[Z_i - \theta] \leq 0,$$

where $\{Z_i\}_{i=1}^n$ is n draws from the d_m dimensional random vector Z_i. We consider three cases of d_m: $d_m = 2, 4, 10$. We consider the sample size $n = 100$.

To generate the simulated data set, we let $Z_i \sim N(0, \Omega)$ and be i.i.d. across i for a variance-covariance matrix Ω. We consider three cases of Ω: Ω_{Zero}, Ω_{Neg}, Ω_{Pos}, respectively representing the cases were the moment inequalities are independent, have negative correlations, and are positive correlated. The specific values of these matrices are the same as those set in Andrews and Barwick (2012) and Cox and Shi (2023b).

We consider one null value of θ, which satisfies $\theta \geq 0$, and three alternative parameter values of θ which do not satisfy $\theta \geq 0$. The null parameter is set to $\theta = \mathbf{0}$, while the alternative parameter values are set to the first three elements of $\mathcal{M}_{d_m}(\Omega)$ that appears in Section S7.1 of the Supplemental Material of Andrews and Barwick (2012), which depends on d_m and Ω.

For subvector inference, we use the interval outcome regression example similar to the example in Cox and Shi (2023b). We modify this example by replacing its endogenous variable with an exogenous covariate. Specifically, instead of generating the regressors according to Cox and Shi (2023b), we let the regressors be $X_i = (1, X_{1i}, X_{2i}, \ldots, X_{d_c i})'$ where the non-constant elements of X_i are independent Bernoulli random variables with success probability 0.5.[5] We consider $d_c = 2$ and 4. The full parameter is $\beta = (\beta_{cons}, \beta_1, \beta_2, \ldots, \beta_{d_c})'$, where the parameter of interest is β_1. Hence, $\theta = \beta_1$ while $\delta = (\beta_{cons}, \beta_2, \ldots, \beta_{d_c})'$ in the notation of Section 2.2. We set the true parameters to $\theta_0 = -1$ and $\delta_0 = (0, -1, \mathbf{0}'_{d_c-2})'$. Other elements of the model remain the same to the description of the model in Cox and Shi (2023b). We consider two sample sizes: $n = 500$ and $n = 1000$. We implement the sCC test for θ at the hypothesized values: $\{-1.5, -1.25, -1, -0.75, -0.5\}$, where -1 is the true value and the others are values outside the identified set for θ.

Table 1 reports the results of the CC test. It shows that the empirical null rejection probability is close to the nominal level 5% as expected. The CC test also exhibits nontrivial power against alternative parameter values. The results from our Stata `ccscc` command are almost identical to those from the Matlab code of Cox and Shi (2023b), with the maximum difference between the number of rejections by the two codes not exceeding 1 out of 1000 trials in any case.

Table 2 reports the results of the sCC test. The empirical rejection probabilities do

5. Note that in Cox and Shi (2023b), X_{1i} in X_i defined above is endogenous. We change it to exogenous because `sccintreg` does not apply to regression models with endogenous regressors.

not exceed 5% for the true hypothesis $\theta = -1$ while the rejection probabilities increases as θ deviates from the true value. In most cases, the rejection probabilities for the two different commands do not exceed 0.002, except when $d_c = 4$ and $n = 1000$ with $\theta = -0.75$, where the difference is 0.006.

The small differences between the Matlab results and the Stata results are not unexpected. They occur because the quadratic programming implementation by Python's CVXOPT package is not exactly the same as that by the `quadprog` package of Matlab. Due to the different implementation, they can display different numerical imprecision even when their tolerance levels are all set to be the same. In Tables 1 and 2, the differences in rejection probabilities are all in the third decimal point. The CC and the sCC tests implemented by both programs display excellent finite sample size and power properties.

8 Examples

We illustrate how to use the **ccscc** command to do inference for moment inequality models and the **sccintreg** command for an interval regression.

8.1 Full-vector inference: ccscc

For illustration of the CC test, we use the first simulation example in Section 7. Let $Z_i \sim i.i.d.N(0, I_2)$ for $i = 1, ..., n$, with sample size $n = 100$, where I_2 is the identity matrix of size 2. For this example, $m(Z_i, \theta) = Z_i - \theta$.

First, we use the CC test to test the value $\theta = (0, 0)'$. That is, we set $\theta = (0, 0)'$, and use `ccscc` to test: $\mathcal{H}_0 : E[\bar{m}_n(\theta)] \leq 0$. We first import the data set `dataex1`, which contains the variables $Z1$ and $Z2$, then generate the matrices A and b which in this case are $A = I_2$ and $b = (0, 0)'$, then generate the input variable lists: $m1 = Z1 - \theta(1)$ and $m2 = Z2 - \theta(2)$, and finally run the command `ccscc` with the inputs generated. Below is the the log file that shows the command lines implemented and the output of the `ccscc` test:

```
. use dataex1
.
. matrix A = (1, 0 \ 0, 1)
. matrix b = (0 \ 0)
.
. // Testing H0 : theta=(0,0)
. generate m1 = Z1-0
. generate m2 = Z2-0
. ccscc () m1 m2, matineq(A) vecineq(b)
Number of obs : 100
Method : CC
Accept the hypothesis
test statistic : .13656579
critical value : 3.8414588
rank           : 1
```

Table 1: Empirical rejection probabilities for the CC test. Nrp denotes rejection rate for the test at the null parameter $\theta = \mathbf{0}$. Pwr1-3 denote empirical powers for the three alternative parameter values explained in this section. `stata` denotes testing by the Stata `ccscc` command while `matlab` denotes testing by the Matlab code from Cox and Shi (2023b).

		$d_m = 2$		$d_m = 4$		$d_m = 10$	
		`stata`	`matlab`	`stata`	`matlab`	`stata`	`matlab`
		CC test					
Ω_{Zero}	Nrp	0.037	0.036	0.034	0.033	0.049	0.049
	Pwr1	0.630	0.630	0.649	0.649	0.551	0.551
	Pwr2	0.639	0.639	0.666	0.666	0.575	0.575
	Pwr3	0.664	0.664	0.666	0.666	0.634	0.634
Ω_{Neg}	Nrp	0.050	0.049	0.039	0.039	0.055	0.055
	Pwr1	0.571	0.571	0.663	0.663	0.569	0.569
	Pwr2	0.589	0.589	0.663	0.663	0.603	0.603
	Pwr3	0.598	0.598	0.663	0.663	0.605	0.605
Ω_{Pos}	Nrp	0.058	0.058	0.041	0.041	0.063	0.063
	Pwr1	0.564	0.564	0.551	0.551	0.624	0.624
	Pwr2	0.673	0.673	0.563	0.563	0.676	0.675
	Pwr3	0.694	0.694	0.566	0.566	0.687	0.687
		RCC test					
Ω_{Zero}	Nrp	0.049	0.048	0.048	0.047	0.051	0.051
	Pwr1	0.656	0.656	0.726	0.726	0.551	0.551
	Pwr2	0.693	0.693	0.754	0.754	0.576	0.576
	Pwr3	0.745	0.745	0.757	0.757	0.637	0.637
Ω_{Neg}	Nrp	0.053	0.052	0.056	0.056	0.055	0.055
	Pwr1	0.571	0.571	0.710	0.710	0.569	0.569
	Pwr2	0.591	0.591	0.710	0.710	0.603	0.603
	Pwr3	0.600	0.600	0.710	0.710	0.605	0.605
Ω_{Pos}	Nrp	0.058	0.058	0.051	0.051	0.063	0.063
	Pwr1	0.564	0.564	0.592	0.592	0.624	0.624
	Pwr2	0.676	0.676	0.611	0.611	0.676	0.675
	Pwr3	0.698	0.698	0.614	0.614	0.687	0.687

```
p-value        : .71171924
```

As we can see, the test does not reject $\theta = (0, 0)'$ at the default significance level $\alpha = 5\%$. The CC test statistic and critical value are respectively .13656579 and 3.8414588. And the p-value is .71171924.

Next, we illustrate the use of the `refined` option. This option could make a difference to the critical value in this case because $\hat{r} = 1$:

```
. // Refined CC test at theta=(0,0)
. ccscc () m1 m2, matineq(A) vecineq(b) r
```

Table 2: Empirical rejection probabilities for the sCC test at different values of θ. The true value $\theta_0 = -1$. `stata` denotes testing by the Stata `ccscc` command while `matlab` denotes testing by the modification of the Matlab code from Cox and Shi (2023b).

	$n = 500$				$n = 1000$			
	$d_c = 2$		$d_c = 4$		$d_c = 2$		$d_c = 4$	
θ	`stata`	`matlab`	`stata`	`matlab`	`stata`	`matlab`	`stata`	`matlab`
-1.50	0.949	0.949	0.832	0.830	0.998	0.998	0.987	0.987
-1.25	0.245	0.244	0.187	0.184	0.407	0.406	0.313	0.312
-1.00	0.005	0.005	0.016	0.015	0.002	0.002	0.007	0.007
-0.75	0.162	0.162	0.108	0.106	0.235	0.234	0.142	0.136
-0.50	0.923	0.923	0.774	0.772	0.997	0.997	0.972	0.972

```
Number of obs : 100
Method : RCC
Accept the hypothesis
test statistic : .13656579
critical value : 3.2282258
rank           : 1
p-value        : .49166099
```

Note that the refinement did make a difference by reducing the critical value from 3.8414588 to 3.2282258 and the p-value from .71171924 to .49166099.

Finally, we illustrate how to use the CC test to test different θ values. We try two additional θ values: $\theta = (1, -1)'$ and $\theta = (1, 1)'$. The value $\theta = (1, 1)'$ is in the interior of the null parameter space. At this value, both moment inequalities are slack in population. It is likely that both the test statistic and the critical value are zero, in which case the sCC test does not reject the null hypothesis.

```
. // H0 : theta=(1,-1)
. replace m1 = Z1-(1)
(100 real changes made)
. replace m2 = Z2-(-1)
(100 real changes made)
. ccscc () m1 m2, matineq(A) vecineq(b)
Number of obs : 100
Method : CC
Reject the hypothesis
test statistic : 71.579337
critical value : 3.8414588
rank           : 1
p-value        : 0
.
. // H0 : theta=(1,1)
. replace m1 = Z1-(1)
(0 real changes made)
. replace m2 = Z2-(1)
(100 real changes made)
. ccscc () m1 m2, matineq(A) vecineq(b)
Number of obs : 100
Method : CC
```

```
Accept the hypothesis
test statistic : 0
critical value : 0
rank           : 0
p-value        : 1
```

8.2 Subvector inference: ccscc

For illustration of the sCC test, we use the interval-outcome linear regression example, which is the second simulation example in Section 7. In addition to the binary regressor set up considered in Section 7, we also consider a case where X_{1i} and X_{2i} are uniform random variables with support $[0, 1]$. We let the two regressors be independent to each other in both the discrete and the continuous cases.

Set the number of observations $n = 100$ and the true parameters $\beta_1 = \beta_2 = -1$ and $\beta_{cons} = 0$, where $X_i = (1, X_{1i}, X_{2i})'$ and $\beta = (\beta_{cons}, \beta_1, \beta_2)'$. Other elements of the model remains the same with the one in Section 7.

Testing a given value of the parameter

We first run the 5%-size sCC test for the null hypothesis $\mathcal{H}_0 : \beta_1 = -1$. The nuisance parameter is $\delta = (\beta_0, \beta_2)'$.

First, we load a dataset that contains the interval measurements Yl, Yu of the dependent variable as well as the continuous regressors X1 and X2.

```
. use dataex2
. // Summary of continuous variables X1 and X2
. summarize X1 X2
```

Variable	Obs	Mean	Std. dev.	Min	Max
X1	100	.5102962	.2887167	.0037044	.9995728
X2	100	.5091578	.2874893	.0254002	.9978425

The summary statistics reported by the command `summarize X1 X2` tell us that X_{1i} and X_{2i} are continuous random variables.

We define the instrumental function by

$$\mathcal{I}(X_i) = \mathcal{I}(X_{1i}, X_{2i}) = \begin{pmatrix} \mathbb{I}(X_{1i} \leq 0.5, X_{2i} \leq 0.5) \\ \mathbb{I}(X_{1i} \leq 0.5, X_{2i} > 0.5) \\ \mathbb{I}(X_{1i} > 0.5, X_{2i} \leq 0.5) \\ \mathbb{I}(X_{1i} > 0.5, X_{2i} > 0.5) \end{pmatrix}.$$

Each element of $\mathcal{I}(X_i)$ should be a variable in Stata. These variables are generated as follows:

```
. // Generate I(X), instrument function
. generate iv1 = 1*(X1<=0.5) & (X2<=0.5)
. generate iv2 = 1*(X1<=0.5) & (X2>0.5)
. generate iv3 = 1*(X1>0.5) & (X2<=0.5)
```

```
. generate iv4 = 1*(X1>0.5) & (X2>0.5)
```

Let $X_{-1,i} = (1, X_{2i})'$. In the following code, we compute elements of $\begin{pmatrix} (\mathcal{I}(X_i)X'_{-1,i}) \\ -(\mathcal{I}(X_i)X'_{-1,i}) \end{pmatrix}$, which are needed for computing C_Z.

```
. // Generate (I(X)´,-I(X)´)´*(1,X2), which are 8 x 2 matrix.
. // Sample average of it becomes the matrix Cz,
. // which is multiplied before nuisance parameter in the moment inequality
. generate c11 = iv1
. generate c21 = iv2
. generate c31 = iv3
. generate c41 = iv4
. generate c51 = -iv1
. generate c61 = -iv2
. generate c71 = -iv3
. generate c81 = -iv4
. generate c12 = iv1*X2
. generate c22 = iv2*X2
. generate c32 = iv3*X2
. generate c42 = iv4*X2
. generate c52 = -iv1*X2
. generate c62 = -iv2*X2
. generate c72 = -iv3*X2
. generate c82 = -iv4*X2
```

We define the 8×2 matrix $C_Z = \frac{1}{n}\sum_{i=1}^{n} \begin{pmatrix} (\mathcal{I}(X_i)X'_{-1,i}) \\ -(\mathcal{I}(X_i)X'_{-1,i}) \end{pmatrix}$ in the following code:

```
. // Compute Cz using the data we generated above
. matrix C = J(8,2,.)
. qui mean c11 c12 c21 c22 c31 c32 c41 c42 c51 c52 c61 c62 c71 c72 c81 c82
. matrix meanC = r(table)[1,1..16]
.
. forvalues i = 1/8{
  2.         forvalues j = 1/2{
  3.                 matrix C[`i´,`j´] = meanC[1,2*(`i´-1)+`j´]
  4.         }
  5. }
```

We also need to compute $\begin{pmatrix} (\mathcal{I}(X_i)X_{1i}) \\ -(\mathcal{I}(X_i)X_{1i}) \end{pmatrix}$ that are multiplied by the parameter of interest β_1.

```
. // Generate (I(X)´*X1,-I(X)´*X1)´
. generate zv1 = iv1*X1
. generate zv2 = iv2*X1
. generate zv3 = iv3*X1
. generate zv4 = iv4*X1
. generate zv5 = -iv1*X1
. generate zv6 = -iv2*X1
```

```
. generate zv7 = -iv3*X1
. generate zv8 = -iv4*X1
```

In the following code, we compute $\begin{pmatrix} (\mathcal{I}(X_i)Y_{Li}) \\ -(\mathcal{I}(X_i)Y_{Ui}) \end{pmatrix}$.

```
. // Generate (I(X)´*Yl,-I(X)´*Yu)´
. generate zy1 = iv1*Yl
. generate zy2 = iv2*Yl
. generate zy3 = iv3*Yl
. generate zy4 = iv4*Yl
. generate zy5 = -iv1*Yu
. generate zy6 = -iv2*Yu
. generate zy7 = -iv3*Yu
. generate zy8 = -iv4*Yu
```

To test $\beta_1 = -1$, we need to construct $m(W_i, \beta_1) = \begin{pmatrix} (\mathcal{I}(X_i)(Y_{Li} - \beta_1 X_{1i})) \\ -(\mathcal{I}(X_i)(Y_{Ui} - \beta_1 X_{1i})) \end{pmatrix}$ as follows:

```
. // Generate (I(X)´*(Yl-(-1)*X1),-I(X)´*(Yu-(-1)*X1))´
. // Its moment is used to test H0: beta1 = (-1)
. generate m1 = zy1 - (-1)*zv1
. generate m2 = zy2 - (-1)*zv2
. generate m3 = zy3 - (-1)*zv3
. generate m4 = zy4 - (-1)*zv4
. generate m5 = zy5 - (-1)*zv5
. generate m6 = zy6 - (-1)*zv6
. generate m7 = zy7 - (-1)*zv7
. generate m8 = zy8 - (-1)*zv8
```

Then, all inputs for the sCC test are ready. We can run the 5%-size sCC test using the `ccscc` command as follows:

```
. matrix dz = J(8,1,0)
. matrix B = I(8)
.
. ccscc (X1 X2) m1 m2 m3 m4 m5 m6 m7 m8, matineq(B) matnuisa(C) vecineq(dz)
Number of obs : 100
Accept the hypothesis
test statistic : .02629131
critical value : 3.8414588
rank           : 1
p-value        : .87119092
```

Note that the test for $\mathcal{H}_0 : \beta_1 = -1$ is not rejected at the default significance level 5%. The test statistic and critical value are .02629131 and 3.8414588, and the p-value is .87119092.

We now show how to run the sCC test when the exogenous variables are discrete. In the following example, we use a dataset where X_1 and X_2 are binary variables. We need to add the `ivdiscrete` option to the `ccscc` command so that $\widehat{\Sigma}_n$ is efficiently obtained.

```
. clear
. use dataex3
. // Summary of discrete variables X1 and X2
. summarize i.X1 i.X2
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
          X1 |
          0  |        100         .47    .5016136          0          1
          1  |        100         .53    .5016136          0          1
             |
          X2 |
          0  |        100         .43    .4975699          0          1
          1  |        100         .57    .4975699          0          1
.
. generate iv1 = 1*(X1==0) & (X2==0)
. generate iv2 = 1*(X1==0) & (X2==1)
. generate iv3 = 1*(X1==1) & (X2==0)
. generate iv4 = 1*(X1==1) & (X2==1)
.
. generate c11 = iv1
. generate c21 = iv2
. generate c31 = iv3
. generate c41 = iv4
. generate c51 = -iv1
. generate c61 = -iv2
. generate c71 = -iv3
. generate c81 = -iv4
. generate c12 = iv1*X2
. generate c22 = iv2*X2
. generate c32 = iv3*X2
. generate c42 = iv4*X2
. generate c52 = -iv1*X2
. generate c62 = -iv2*X2
. generate c72 = -iv3*X2
. generate c82 = -iv4*X2
.
. matrix C = J(8,2,.)
. qui mean c11 c12 c21 c22 c31 c32 c41 c42 c51 c52 c61 c62 c71 c72 c81 c82
. matrix meanC = r(table)[1,1..16]
.
. forvalues i = 1/8{
  2.         forvalues j = 1/2{
  3.                 matrix C[`i´,`j´] = meanC[1,2*(`i´-1)+`j´]
  4.         }
  5. }
.
. generate zv1 = iv1*X1
. generate zv2 = iv2*X1
. generate zv3 = iv3*X1
. generate zv4 = iv4*X1
```

```
. generate zv5 = -iv1*X1
. generate zv6 = -iv2*X1
. generate zv7 = -iv3*X1
. generate zv8 = -iv4*X1
.
. generate zy1 = iv1*Yl
. generate zy2 = iv2*Yl
. generate zy3 = iv3*Yl
. generate zy4 = iv4*Yl
. generate zy5 = -iv1*Yu
. generate zy6 = -iv2*Yu
. generate zy7 = -iv3*Yu
. generate zy8 = -iv4*Yu
.
. generate m1 = zy1 - (-1)*zv1
. generate m2 = zy2 - (-1)*zv2
. generate m3 = zy3 - (-1)*zv3
. generate m4 = zy4 - (-1)*zv4
. generate m5 = zy5 - (-1)*zv5
. generate m6 = zy6 - (-1)*zv6
. generate m7 = zy7 - (-1)*zv7
. generate m8 = zy8 - (-1)*zv8
.
. matrix dz = J(8,1,0)
. matrix B = I(8)
.
. ccscc (X1 X2) m1 m2 m3 m4 m5 m6 m7 m8, matineq(B) matnuisa(C) vecineq(dz) ivd
Number of obs : 100
Accept the hypothesis
test statistic : .38010528
critical value : 5.9914645
rank           : 2
p-value        : .8269156
```

Applying test inversion to obtain a confidence set

We now illustrate how to apply test inversion to construct confidence intervals using our `ccscc` command. We continue using the interval outcome linear regression example with the discrete dataset `dataex3` we loaded in the last section and consider β_1.

We employ the grid search method here, which involves constructing a grid of parameter values and conducting a sequence of tests for each value in the grid. We construct the grid by $\{-2 + 0.05j : 0 \leq j \leq 40\}$.

```
. // Construct CS using test inversion
.
. matrix grid = J(41,1,0)
. matrix rejects = J(41,1,0)
```

```
. matrix tstats = J(41,1,0)
. matrix cvs = J(41,1,0)
.
. forvalues j = 1/41{
  2.         matrix grid[`j´,1] = 0.05*(`j´-1)-2
  3. }
.
. forvalues j = 1/41{
  2.         qui replace m1 = zy1 - grid[`j´,1]*zv1
  3.         qui replace m2 = zy2 - grid[`j´,1]*zv2
  4.         qui replace m3 = zy3 - grid[`j´,1]*zv3
  5.         qui replace m4 = zy4 - grid[`j´,1]*zv4
  6.         qui replace m5 = zy5 - grid[`j´,1]*zv5
  7.         qui replace m6 = zy6 - grid[`j´,1]*zv6
  8.         qui replace m7 = zy7 - grid[`j´,1]*zv7
  9.         qui replace m8 = zy8 - grid[`j´,1]*zv8
 10.
.         qui ccscc (X1 X2) m1 m2 m3 m4 m5 m6 m7 m8, matineq(B) matnuisa(C) vecineq(dz) ivd
 11.
.         matrix rejects[`j´,1] = `r(reject)´
 12.         matrix tstats[`j´,1] = `r(t_stat)´
 13.         matrix cvs[`j´,1] = `r(cv)´
 14. }
.
. svmat grid
. svmat tstats
. svmat cvs
.
. // Plot test statistics and critical values over the grid
. line tstats grid, legend(order(1 "Test statistics" 2 "Critical values")) xtitle("") ///
> || line cvs grid
```

The Stata matrix `grid` in the code above contains the values of the grid points. The Stata matrix `rejects` stores results indicating whether the test rejected the parameter values in the grid. The Stata matrices `tstats` and `cvs` hold the test statistic and critical value for each parameter value in the grid. The first loop in the code constructs the grid, while the second loop performs the sCC test for each hypothesized parameter value within the grid. The final line of the code returns Figure 1, which displays the test statistics and critical values for hypothesized parameter values across the grid. In Figure 1, the parameter values where the test statistics are lower than their corresponding critical values are not rejected. The collection of these parameter values forms the 95% confidence interval. Figure 1 shows that lower and upper bounds of the 95% confidence interval are approximately -1.5 and -0.5. More precise bounds can be obtained either by considering a finer grid or by a more refined search algorithm such as the bisection algorithm implemented in `sccintreg`.

8.3 sccintreg

For the interval outcome linear regression model, the construction of marginal confidence intervals is automated by the command `sccintreg`, so that the user does not have to manually write the loops for the bisection or grid-search.

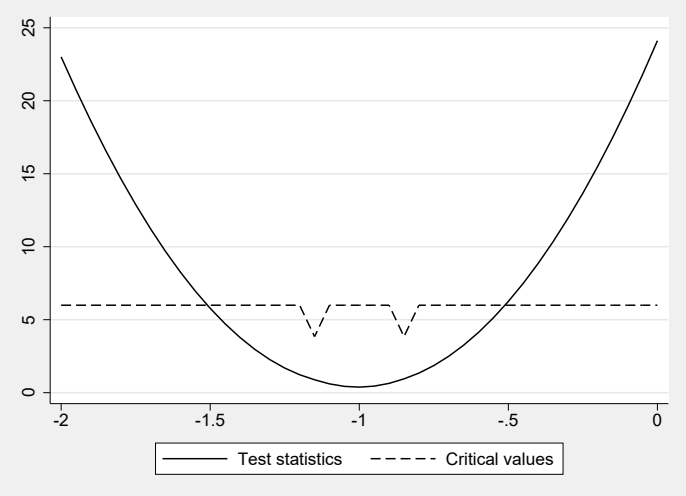

Figure 1: x-axis is the grid of hypothesized parameter values. The solid line represents the test statistics. The dashed line represents the critical values.

To illustrate the command `sccintreg`, we again use the dataset `dataex2`, which has the interval measures Yl and Yu and continuous regressors X1 and X2. We start with running the command without any options:

```
. use dataex2 // Use continuous X dataset
.
. sccintreg Yl Yu X1 X2
Number of obs : 100
Table of confidence intervals
```

Variable	[95 % Conf.	Interval]
constant	-.42101443	1.1744011
X1	-2.0398755	.03558808
X2	-2.6442282	-.55284578

The command without options set default values to all options, which are explained in Section 7 above. In this example, the command reports a table of 95% confidence intervals for all 3 coefficients in the model.

We use the data set `dataex3` where the regressors are discrete variables to illustrate the option `discrete`.

```
. clear
. use dataex3 // Use discrete X dataset
. sccintreg Yl Yu X1 X2, d
Number of obs : 100
Table of confidence intervals
```

```
Variable |    [95 % Conf.      Interval]
---------+--------------------------------
constant |     -.41889411      .50473672
X1       |     -1.5094282      -.5111461
X2       |     -1.5798855     -.60470784
```

Users can specify options to perform the grid search instead, which typically returns cruder confidence intervals but has the advantage of saving the test statistics and critical values at each grid point for plotting. Now we implement the command with the grid search option using the discrete dataset we loaded:

```
. // Use grid search for the coefficient of "X1"
. // Grid is from -2 to 0 and has 201 points.
. matrix lbound = (-2)
. matrix ubound = (0)
. sccintreg Yl Yu X1 X2, d lb(lbound) ub(ubound) subset("X1") grid(201)
Calculate CIs for (  X1 )
Bounds are set
X1     : [-2, 0]
Number of obs : 100
Table of confidence intervals
---------+--------------------------------
Variable |    [95 % Conf.     Interval]
---------+--------------------------------
X1       |          -1.51          -.51
---------+--------------------------------
```

The following code shows how to make the plot of the test statistics and critical values, Figure 2, after running `sccintreg`. Figure 2 provides a more detailed view compared to Figure 1, featuring a finer grid of points.

```
. matrix grid = r(GRIDs)[1,1..201]´
. matrix Tns = r(Tns)[1,1..201]´
. matrix CVs = r(CVs)[1,1..201]´
. svmat grid
number of observations will be reset to 201
Press any key to continue, or Break to abort
Number of observations (_N) was 100, now 201.
. svmat Tns
. svmat CVs
.
. // Plot test statistics and critical values over the grid
. line Tns grid, legend(order(1 "Test statistics" 2 "Critical values")) xtitle("") ///
> || line CVs grid
```

9 Conclusion

Inequality testing arises in a variety of economic applications. In this article, we introduce the new `ccscc` command that implements computationally simple and fast inequality tests proposed by Cox and Shi (2023b). We also conduct Monte Carlo simulations to compare the results obtained from the `ccscc` command with those from Cox and Shi (2023b). Additionally, we demonstrate how to use the `ccscc` command for full-

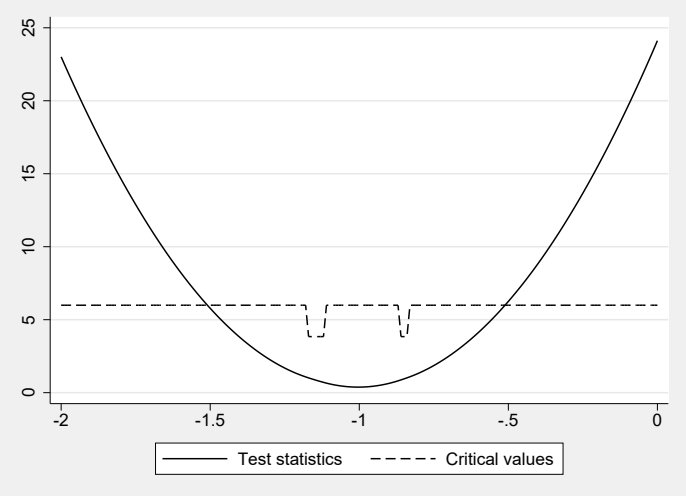

Figure 2: x-axis is the grid of hypothesized parameter values. The solid line represents the test statistics. The dashed line represents the critical values.

vector and subvector inferences with simulated datasets. We additionally introduce the `sccintreg` command which is convenient for linear regression with an interval outcome and show how to use the command through applications using simulated datasets. A limitation of the commands developed in this article is that they do not directly apply to the generalized CC test in Cox et al. (2024), but the package developed here can be a useful basis for developing a more general command that covers the GCC test.

10 References

Abadie, A., G. W. Imbens, and F. Zheng. 2014. Inference for Misspecified Models With Fixed Regressors. *Journal of the American Statistical Association* 109(508): 1601–1614.

Andersen, M. S., J. Dahl, and L. Vandenberghe. 2023. CVXOPT: A Python package for convex optimization, version 1.1.6. Available at cvxopt.org. Technical report.

Andrews, D. W. K., and P. J. Barwick. 2012. Inference for Parameters Defined by Moment Inequalities: A Recommended Moment Selection Procedure. *Econometrica* 80(6): 2805–2826.

Andrews, D. W. K., W. Kim, and X. Shi. 2017. Commands for Testing Conditional Moment Inequalities and Equalities. *The Stata Journal* 17(1): 56–72.

Andrews, D. W. K., and X. Shi. 2013. Inference Based on Conditional Moment Inequalities. *Econometrica* 81(2): 609–666.

Canay, I. A., G. Illanes, and A. Velez. 2023. A User's guide for inference in models defined by moment inequalities. *Journal of Econometrics* .

Canay, I. A., and A. Shaikh. 2017. Practical and Theoretical Advances for Inference in Partially Identified Models In B. Honoré, A. Pakes, M. Piazzesi, and L. Samuelson (Eds) *Advances in Economics and Econometrics*: Volume 2: Eleventh World Congress, (Econometric Society Monographs, pp. 271-306). Cambridge University Press.

Chernozhukov, V., W. Kim, S. Lee, and A. M. Rosen. 2015. Implementing Intersection Bounds in Stata. *The Stata Journal* 15(1): 21–44.

Chernozhukov, V., S. Lee, and A. M. Rosen. 2013. Intersection Bounds: Estimation and Inference. *Econometrica* 81(2): 667–737.

Cox, G., and X. Shi. 2023a. Supplemental Appendix for "Simple Adaptive Size-Exact Testing for Full-Vector and Subvector Inference in Moment Inequality Models," The Review of Economic Studies 90, 201-228. Available on academic.oup.com/restud/article/90/1/201/6548809. Technical report.

———. 2023b. Simple Adaptive Size-Exact Testing for Full-Vector and Subvector Inference in Moment Inequality Models. *The Review of Economic Studies* 90: 201–228.

Cox, G., X. Shi, and Y. Shimizu. 2024. Testing Inequalities Linear In Nuisance Parameters unpublished manuscript.

Dickstein, M. J., J. Jeon, and E. Morales. 2024. Patient Costs and Physicians Information NBER working paper.

Elliot, G., N. Kudrin, and K. Wuthrich. 2022. Detecting p-Hacking. *Econometrica* 90: 887–906.

Gandhi, A. K., Z. Lu, and X. Shi. 2023. Estimating Demand for Differentiated Products with Zeroes in Market Share Data. *Quantitative Economics* 14: 381–418.

Ho, K., and A. Rosen. 2017. Partial Identification in Applied Research: Benefits and Challenges In B. Honoré, A. Pakes, M. Piazzesi, and L. Samuelson (Eds) *Advances in Economics and Econometrics*: Volume 2: Eleventh World Congress, (Econometric Society Monographs, pp. 307-359). Cambridge University Press.

Molinari, F. 2020. Econometrics with Partial Identification In S. Durlauf, L. Hansen, J. Heckman, and R. Matzkin (Eds) *Handbook of Econometrics*: Volume 7A, 1st Edition. North Holland.

Yuan, Z., and P. J. Barwick. 2024. Network Competition in the Airline Industry: An Empirical Framework Working paper, University of Wisconsin at Madison.