Broad Tasks Involved in Implementing the Procedure

1. Elicit a specification of the propensity score, estimate it, and save the predicted values.
2. Elicit a set $\mathcal{S}$. For each $s \in \mathcal{S}$, construct the function s itself and β_s.
3. Elicit a specification of m_0 and m_1, including shape restrictions.
4. Elicit a parameter of interest, $\beta^\star$.
5. Use $\mathcal{S}$, $\beta^\star$, m_0, and m_1 to construct the constants γ_{ds} and $\gamma_d^\star$.
6. Minimize the observational equivalence criterion. [**New step, not in the papers; see below.**]
7. Minimize and maximize $\beta^\star$ subject to the constraint that the observational equivalence condition is not violated by too much more than the minimum found in the previous step.

I believe the ideal structure would be to have each task as a separate function. Then towards the end, we can make a single driver/wrapper function that allows a user to run only a single command. The driver/wrapper would basically just unpack the inputs and then call the separate functions above. The idea would be to make the code simple enough so that unsophisticated practitioners can use it, while still leaving it flexible enough so that power users can intervene in the different steps if they want to. (I believe that this is how most R packages are organized, but that is just from my very casual and selected observation — could be completely off.)

1 Task 1 — Propensity Score

This part should be straightforward I think. Some key points:

1a) I don't believe we need the propensity score per se, just the fitted values.

1b) It would be good to allow the user some built-in flexibility with how the propensity score is estimated so that logit/probit/linear probability model are all possibilities. The syntax should be as close to the usual R syntax as possible.

1c) We also want to let the user bypass this step and construct their own predicted propensity score values. Perhaps they could do that by referring to a variable in the dataset that has these predicted values.

1d) There needs to be some error checking that all predicted values of the propensity score are in $[0, 1]$. Perhaps this is done at the next stage (since the user can input their own predicted values).

2 Task 2 — Elicit $\mathcal{S}$

2a) I think for the sake of sanity we want to limit ourselves to accepting one type of IV–like estimand, which is anything that can be specified in terms of a two-stage least squares (TSLS) estimand. This includes OLS and IV/Wald as special cases. It also means we can use the usual TSLS syntax for R, and then from that construct the implied s function. So the input is then a list of TSLS specifications.

2b) We would also want to allow the user to optionally select a subset of the TSLS coefficients to use — this could be specified through a variable list I think.

2c) Perhaps as a slight compromise to the above we also allow the user to directly input s as a function?

2d) I believe it should be possible to do the sharp bounds (when D, Z and X are all discrete) through a correct set of TSLS specifications. This should be double-checked. I don't think we need an option for sharp bounds, but we just want to make sure that our focus on only TSLS specifications does not rule this out, since it is important conceptually.

3 Task 3 — Specifying m_0 and m_1

This is the task I am struggling with the most.

Imposing Boundedness

The key problem is that we generally need to require that m_0 and m_1 are bounded (e.g. in $[0, 1]$) in order to get non-trivial bounds. The fact that we need this is fine. The problem is how to *enforce* these bounds through linear constraints.

This problem is much simpler when there are no covariates, so that m_0 and m_1 have one-dimensional domains. For this case, the solution we used was to code the polynomials through the Bernstein basis. This allows one to enforce bounds simply by bounding the coefficients on the Bernstein basis, as discussed in (e.g.) Chak, Madras, and Smith (2005).

However, even here there is a subtle issue that is somewhat unsatisfactory. The issue is that the set of all Bernstein polynomials of a given order with coefficients in $[0, 1]$ is a strict subset of the set of all (power basis) polynomials of the same order that are bounded in $[0, 1]$. That is, keeping the polynomial order fixed, all Bernstein polynomials with coefficients in $[0, 1]$ are polynomials that are bounded in $[0, 1]$, but the converse is not true. The reason I find this somewhat unsatisfactory is that regular polynomials are easier for people to interpret than Bernstein polynomials.

Perhaps this is not that big of a problem. In any case, to the extent that it is a problem, the problem diminishes as the order of the polynomial increases in the sense that both the set of Bernstein and power-basis polynomials with coefficients in $[0, 1]$ becomes dense in the set of continuous functions bounded in $[0, 1]$. An alternative and perhaps more elegant solution is to use a fixed-order Bernstein polynomial *spline* basis with fixed knot points, and adjust the knot points instead of the order of the polynomial. A paper by Papp and Alizadeh (2014) provides a good justification and some examples (see the supplemental code).

The case with covariates is where this gets more difficult, because m_0 and m_1 have larger domains. The types of specifications we want to handle are:

$$m_1(u, x) = g_1(u) + x'\alpha_1 + g_2(u)x'\alpha_2$$
$$\text{where} \quad g_j(u) = \sum_{k=1}^{K_j} \theta_{jk} b_j(u) \tag{1}$$

with something similar for m_0. Here g_1 and g_2 are K_j–dimensional linear polynomial bases with unknown parameters θ_{kj}. The second term, $x'\alpha_1$, allows for the usual linear regression type of flexibility in how x enters, i.e. it allows for nonlinear functions of x as long as they are linear-in-parameters. The final term $g_2(u)x'\alpha_2$ allows for some interactions between u and x, but some or all of the interactions could be set to zero by requiring some or all of the components of α_2 to be 0.

The reason that (1) is difficult is that we still need some way to ensure that m_1 is bounded. One possibility is to view $m_1(u, x)$ as a large multivariate Bernstein polynomial with the coefficients bounded. In that case, (1) would mean choosing a low order polynomial for the x components, and then zero'ing out many of the cross terms between different components. That is, typically one wouldn't want to have $x_l x_k$ for all possible l and k, but these terms can be eliminated by setting their coefficients to zero. The potential problem with this approach is the same as in the univariate case — the set of Bernstein polynomials with coefficients in $[0, 1]$ will be a strict subset of the set of polynomials with range in $[0, 1]$. My intuition, however, is that this problem becomes more severe with polynomials in multiple dimensions, which makes this a bigger problem in the multivariate case. (I might be wrong about this!)

I have been communicating with Dávid Papp (of the paper referenced above) about his work and potential other methods that we could use for this problem. So there may be a graceful solution. I haven't read his dissertation (Papp (2011)) yet, but it looks like there may be some additional insight there. I also have not done a very extensive literature review since the related literature is vast. But this could be useful to do.

One simple possibility is just to add constraints like

$$m_d(u_i, x_i) \in [0, 1] \quad \text{for } d = 0, 1 \tag{2}$$

on a grid of (u_i, x_i). This seems to be fairly common in the literature, e.g. Villalobos and Wahba (1987). However it is obviously an inelegant solution and creates the possibility that the results may depend on how the grid is chosen. We could use this as a fall-back option, however. Perhaps if we put some thought into constructing the grid intelligently we could make this work pretty well.

Syntax and Restrictions

There are a few concerns here that are potentially conflicting:

3a) It would be great if we could use the usual R equation syntax. Something like:

```
m0 ~ u + u^2 + var1 + var2 + var1*var2
```

A likely challenge here is in parsing/interpreting the u terms, but this should be surmountable. This interacts with whether we try to do something elegant with Bernstein polynomials (or some other solution), or go for the less elegant solution (2).

3b) We also want to be able to impose monotonicity constraints on $m_0(u, x)$, $m_1(u, x)$, and/or $m_1(u, x) - m_0(u, x)$. Ideally we would be able to impose these on either u or any component of x. Note that monotonicity is like a boundedness constraint on the derivative, so the solution here will be related to how we solve the bigger boundedness question discussed in the previous section.

3c) Regardless of how we do this, the way that u enters will interact with Task 5. In particular, we probably want to ensure that u only enters as a piecewise polynomial.

3d) How do we allow for this while still allowing the user to be nonparametric in u? Suppose I want to use a constant spline with knot points `K` for u. Is there a way to say something like

```
m0 ~ Spline('constant', K) + var1 + var2 + var1*var2
```

Or perhaps a way to generate a spline with some preliminary function and then insert this into the usual equation? Perhaps consulting some module that does spline estimators would be helpful for syntax.

Grid-Based Approach

The grid-based approach does seem pretty attractive due to its simplicity. The downside is that there is no guarantee that imposing boundedness/monotonicity on a grid necessarily imposes this everywhere. However, in this application maybe this is perhaps not such a big problem, since not imposing boundedness/monotonicity everywhere leads to bounds that are too loose.[1] There is also a worry that creating a sufficiently fine grid will become very difficult with a large number of covariates. Here's a procedure that could be used to ameliorate both problems.

3a) Start with some initial grid. One way to do this is to take a fairly fine grid for u, say 20 (or more) evenly-spaced points from $[0, 1]$ including the endpoints — call this set

[1]At least in the population. When adding the $\widehat{Q}$ step it is ambiguous.

$\mathcal{U}$. Then take a subsample of M (unique) points from the empirical support of the variables that the user inputs (via an R–style equation) into the specifications of m_0 and m_1. Call this subsample $\mathcal{X}_M$. Then the initial grid would be:

$$\mathcal{U} \times \mathcal{X}_M. \tag{3}$$

Intuitively, it would be best to pick X points that cover the empirical support pretty well. One way to do this would be to find an R package that computes some concept of a multivariate quantile, and then take the M points to be evenly spaced quantiles.

3b) Continue with the rest of the procedure, imposing boundedness and monotonicity (if the user requests it) on the grid.

Note that Gurobi/CPLEX can handle a huge number of these types of constraints, since they are very sparse. So one can be fairly aggressive with M and the fineness of $\mathcal{U}$.

3c) Perform an audit step. The idea here is to take the solution (m_0, m_1) and check boundedness (and monotonicity) on a large number of (off-grid) points (u, x). This should be extremely fast to perform since it is just evaluating a polynomial multiple times. Collect all points that fail the audit.

The audit points can be drawn from elements of $[0, 1]$ that are not in $\mathcal{U}$ and elements of the empirical support of X that are not included in the initial grid $\mathcal{X}_M$.

3d) Add the points that fail the audit to the grid and repeat. If there were no points that failed, then stop auditing. If $\widehat{Q}$ did not change appreciably, then also stop auditing.

3e) Repeat the audit step a set number of times. Obviously there should be some default cap on this (say 5?) so that the program can't go into a semi-infinite loop.

I think it will be sufficient to do the audit only on the solutions to the $\widehat{Q}$ step. So we could just loop over the $\widehat{Q}$ step and then when we are done with the auditing, go on to Task 7 and only do that once.

4 Task 4 — Elicit $\beta^\star$

We want to allow for a nice balance between flexibility and ease-of-use here.

4a) Providing some pre-set parameter names for the user is good for ease-of-use. I think we want these to be: ATE, ATT, ATU and LATE$[\underline{u}, \overline{u}]$ where $\underline{u}, \overline{u}$ are parameters input by the user.

4b) For flexibility, the user could input the functions $\omega_0^\star$ and $\omega_1^\star$ directly.

5 Task 5 — Construct the $\gamma_d^\star$ and γ_{ds} Constants

This should follow directly from the paper as long as the specification of m_0, m_1 and $\beta^\star$ are such that the integral involved in the definition of $\gamma_d^\star$ and γ_{ds} can be computed analytically. We will likely restrict m_0 and m_1 such that this is the case, which then means that the integral γ_{ds} will always be computable analytically.

It is possible, however, that the user could input a choice of $\beta^\star$ (through a custom function) for which these integrals have no closed form solution. Detecting this seems rather difficult. Probably it is best to just resort to numerical integration if the user inputs a custom $\omega_d^\star$. It is only one-dimensional numerical integration, and it only needs to be done once on the outset, so this should be relatively easy to do. However, the weight functions are often discontinuous, so we want to make sure that the numerical integration approach we use is reasonably robust to this. Probably we want to leverage some existing R package to do this.

6 Task 6 — Minimize Observational Equivalence

When using this procedure with actual data, it is typically not going to be the case that the observational equivalence conditions can be satisfied exactly. This will result in infeasibility of the linear program. While this suggests misspecification, it could also just reflect some tolerable statistical noise. That is, a formal misspecification test would be needed to address this formally. The situation is exactly analogous to overidentified GMM.

In order to deal with this we add an additional step relative to what it is in the paper right now. (Actually, these steps might be in the revision, but we haven't decided yet.) The first step is to solve for

$$\widehat{Q} \equiv \min_{m \in \mathcal{M}} \sum_{s \in \mathcal{S}} |\Gamma_s(m) - \beta_s|, \tag{4}$$

where we replace $m \in \mathcal{M}$ by the finite basis $\theta \in \Theta$ with the estimated γ_{ds} terms as usual. This program finds the smallest violation of the observational equivalence condition in terms of L_1 loss.

Using L_1 loss is nice because it means we can rephrase (4) as a linear program. In particular, (4) is equivalent to

$$\begin{aligned} \widehat{Q} = \min_{m \in \mathcal{M}, w^-, w^+ \in \mathbb{R}^{|\mathcal{S}|}} & \sum_{s \in \mathcal{S}} w_s^+ + w_s^- \\ \text{s.t.} \quad & w_s^+ - w_s^- = \Gamma_s(m) - \beta_s \\ & w_s^+, w_s^- \geq 0 \text{ for all } s \end{aligned} \tag{5}$$

This is one of those OR tricks — it just follows from the observation that for any real number x one has $x = x^+ - x^-$ and $|x| = x^+ + x^-$ where $x^+ \equiv \max\{0, x\}$ and $x^- \equiv \max\{0, -x\}$. (There's a bit more too the formal argument than that, but that's the essential observation.) We are then going to use $\widehat{Q}$ in Task 7.

7 Task 7 — Construct Bounds

Now we construct estimates of the bounds as

$$\begin{aligned} \widehat{\beta}^{\star} \equiv \max_{m \in \mathcal{M}} \; & \Gamma^{\star}(m) \\ \text{s.t.} \; & \sum_{s \in \mathcal{S}} |\Gamma_s(m) - \beta_s| \leq \widehat{Q} + \epsilon. \end{aligned} \tag{6}$$

Here $\epsilon > 0$ is a tuning parameter that we can probably take to be small for now; say $1e^{-2}$. (In principle, it can be set to 0 subject to small numerical/rounding errors, given the way that $\widehat{Q}$ was constructed, although this requires somewhat strong conditions to justify the asymptotic theory.) We are working on some sort of procedure to get more guidance on a default choice of ϵ. Letting $\widehat{\beta}_{\star}$ be the corresponding minimum, the bounds we return to the user are then $[\widehat{\beta}_{\star}, \widehat{\beta}^{\star}]$.

Note that (5) can also be turned into a linear program using a similar trick as in (5). In particular, a similar argument shows that

$$\begin{aligned} \widehat{\beta}^{\star} = \max_{m \in \mathcal{M}, w^-, w^+ \in \mathbb{R}^{|\mathcal{S}|}} \; & \Gamma^{\star}(m) \\ \text{s.t.} \; & \sum_{s \in \mathcal{S}} \left(w_s^+ + w_s^- \right) \leq \widehat{Q} + \epsilon \\ & w_s^+ - w_s^- = \Gamma_s(m) - \beta_s \\ & w_s^+, w_s^- \geq 0 \text{ for all } s \end{aligned} \tag{7}$$

8 Auxiliary Tasks The User Should be Able to do Easily

This could either be through providing a function, or coding the right returns that the user can then use to achieve one of these tasks fairly easily.

8a) Plotting the (average) weights

8b) Comparing the different TSLS outputs in a table

8c) . . . ?

References

Chak, P. M., N. Madras, and B. Smith (2005): "Semi-nonparametric estimation with Bernstein polynomials," *Economics Letters*, 89, 153–156. 4

Papp, D. (2011): "Optimization Models for Shape-Constrained Function Estimation Problems Involving Nonnegative Polynomials and their Restrictions," Ph.D. thesis, Rutgers, The State University of New Jersey. 5

Papp, D. and F. Alizadeh (2014): "Shape-Constrained Estimation Using Nonnegative Splines," *Journal of Computational and Graphical Statistics*, 23, 211–231. 4

Villalobos, M. and G. Wahba (1987): "Inequality-Constrained Multivariate Smoothing Splines with Application to the Estimation of Posterior Probabilities," *Journal of the American Statistical Association*, 82, 239–248. 5