The Political Methodologist

Newsletter of the Political Methodology Section
American Political Science Association
Volume 23, Number 2, Spring 2016

Editor:
Justin Esarey, Rice University
justin@justinesarey.com

Associate Editors:
Randolph T. Stevenson, Rice University
stevenso@rice.edu

Editorial Assistant:
Ahra Wu, Rice University
ahra.wu@rice.edu

Rick K. Wilson, Rice University
rkw@rice.edu

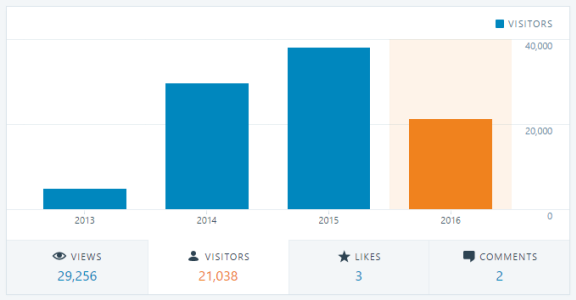
Contents

Notes From the Editors

Welcome to the Spring 2016 edition of *The Political Methodologist*! This issue has special resonance for the editorial team because it is the last issue that will be produced at Rice University.

Since Rice took over *TPM* in 2013 and initiated its blog component, our readership has grown dramatically. Figure 1 shows statistics produced by the blog's WordPress hosting software. As the Figure indicates, *TPM* had nearly 40,000 unique visitors last year and is on track to surpass this threshold in 2016. It's worth noting that a surprising amount of this traffic is driven by just one article: "Making High-Resolution Graphics for Academic Publishing" by Thomas Leeper. In 2015, this article accounted for about 50% of our views! However, even setting this outlier aside, we are proud to have achieved such a large readership within and beyond the community of political methodologists.

Figure 1: Growth in *TPM* readership over time



The search for a new editorial team for *The Political Methodologist* has now begun and is being spearheaded by the current Society President, Jeff Lewis. Those who are interested in taking over editorship of *TPM* should contact Jeff via e-mail at jblewis@polisci.ucla.edu. The Rice team is also happy to answer any questions about editing *TPM*; please direct inquiries to the Editor, Justin Esarey, at justin@justinesarey.com.

-*The Editor*s

Conjoint Analysis Tools for Developing Country Contexts

Alexander Meyer
acomeyer@gmail.com

Leah R. Rosenzweig
Massachusetts Institute of Technology
lrosenzw@mit.edu

Abstract: Conjoint analysis has recently become popular in political science as a tool to understand how respondents choose between multidimensional alternatives. Originally developed for marketing research, political scientists have recently applied this method to understand how citizens evaluate policies (Horiuchi, Smith, and Yamamoto 2015), candidates (Carlson 2015; Rosenzweig and Tsai N.d.) and immigrants (Hainmueller, Hopkins, and Yamamoto 2014; Berinsky et al. 2015). With its growing popularity, conjoint analysis is increasingly being employed across a variety of contexts and respondent samples. Researchers are now applying this method to study attitudes and behaviors in developing countries. However, these contexts pose problems for existing tools and implementation practices. We developed tools to overcome these obstacles that can be used to implement the conjoint method in contexts where surveys are conducted offline and with illiterate respondents.

1. Introduction

Conjoint analysis has long been used in marketing research, but has recently become popular in political science. Originally developed by Luce and Tukey in 1964, conjoint analysis serves as a useful tool for understanding preferences over multidimensional alternatives. This method presents respondents with profiles – for example of candidates (Carlson 2015; Rosenzweig and Tsai N.d.) or immigrants (Hainmueller and Hopkins 2014; Berinsky et al. 2015) – that have randomly assigned attributes and asks respondents to evaluate and choose between them. The random assignment of profile characteristics allows researchers to identify the causal influence of attributes on a person's decision to vote for a candidate or allow an immigrant into the country.[1]

Conjoint analysis is advantageous for researchers interested in observing respondents' choice-making behaviors and attitudes. Using this method, researchers can identify interaction effects as well as analyze particular aspects of treatments. For example, not only can it be used to identify the effect of a candidate's past performance on the probability that respondents will vote for her, but we can also analyze the influence of past performance with respect to the candidate's ethnic identity (Carlson 2015). In addition, conjoint analysis allows us to investigate subgroup effects based on shared attributes between profiles and respondents, which can influence respondent attitudes (Berinsky et al. 2015). Thus, we are able to implement more realistic 'bundled' treatments, testing multiple hypotheses simultaneously (Hainmueller, Hopkins, and Yamamoto 2014).

As with all survey experiments, external validity is always a concern. Hainmueller, Hangartner, and Yamamoto (2015) test the external validity of conjoint analysis by comparing results to a real-world behavioral benchmark in Switzerland. The authors find strong evidence that conjoint experiments can help to explain the preferences and behaviors of people in the real-world. From a paired conjoint design "estimates are within 2% percentage points of the effects in the behavioral benchmark" (Hainmueller, Hangartner, and Yamamoto 2015, p. 2395). Not only is conjoint analysis useful for investigating multiple hypotheses at once, but it can also achieve reliable results.

Until very recently, conjoint analysis had been relegated to online surveys. However, this method presents an excellent opportunity for researchers to understand preferences and behaviors across a host of different contexts. Researchers have begun to take advantage of this method in developing countries (Carlson 2015; Hartman and Morse 2015) but lack widely available resources for easy implementation and standardized best practices. Here we present the tools we developed to help researchers conduct conjoint experiments offline among respondents with little or no education.

2. Application Areas

As described above, conjoint analysis is a useful technique for a variety of settings and questions. The tools that we present here allow researchers to easily apply this method in conditions where surveys are conducted in-person, offline, and respondents have low levels of education or are illiterate. We believe these tools will prove useful for researchers wishing to conduct conjoint experiments in settings such as these–mainly developing countries.

In many developing countries it is difficult to conduct surveys online. The commonly used companies that recruit online samples in the US and other developed nations, such as YouGov, SSI and Mturk, do not yet operate in many developing countries. Part of the problem may be that our target respondents are not yet online. While 82.2% of individuals in developed countries use the internet, only 35.3% of individuals in developing countries do.[2] Although Facebook is becoming a valuable resource for recruiting online samples in developing countries (Rosenzweig and Tsai 2016), there is not yet a reliable, convenient and low-cost way to obtain a representative sample online in many countries that we wish to study. Therefore, in many cases recruiting respon-

[1] See Hainmueller, Hopkins, and Yamamoto (2014) for an analytical discussion of estimation techniques.
[2] ICT Facts and Figures 2015.

dents requires traveling to find them at home. Sometimes respondents reside in rural villages that are disconnected from data networks. In 2015, only 29% of the world's *rural* population had 3G coverage.[3] In these instances the existing tools (Hainmueller, Hopkins, and Yamamoto 2014) to set up a conjoint design will not work.

Figure 1: Sample conjoint profile of hypothetical immigrants (Source: Berinsky et al. (2015)

	Immigrant 1	Immigrant 2
Gender	Male	Male
Country of Origin	Nigeria	France
Education Level	Equivalent to completing two years of college in the US	Equivalent to completing two years of college in the US
Language Skills	During admission interview, this applicant spoke broken English	During admission interview, this applicant spoke through an interpreter
Religion	Atheist	Catholic
Attends Religious Services	Never	More than once a week

Please choose one:

Immigrant 1 ○ Immigrant 2 ○

On a scale from 1 to 7, where 1 indicates that the United States should absolutely not admit the immigrant and 7 indicates that the United States should definitely admit the immigrant, how would you rate Immigrant 1 and Immigrant 2?

	Absolutely not admit the immigrant	2	3	4	5	6	Definitely admit the immigrant
Immigrant 1	○	○	○	○	○	○	○
Immigrant 2	○	○	○	○	○	○	○

Another instance in which the tools we present apply is conducting conjoint experiments with less educated or illiterate respondents. Generally, conjoint profiles look akin to Figure 1. To present conjoint profiles in an intelligible way for many respondents we must abandon the practice of using text. Translating the text into the local language is insufficient in environments where respondents are illiterate or have not received years of education and are unaccustomed to engaging in these types of cognitive tasks. We would moreover advise against relying on enumerators to present the information verbally and respondents retaining this information without the aid of visual cues. As others have done before us (Hartman and Morse 2015), we propose using images to represent profile characteristics. The tools we developed address each of these instances–working with survey software offline and creating conjoint profiles in image form.

3. The Resources

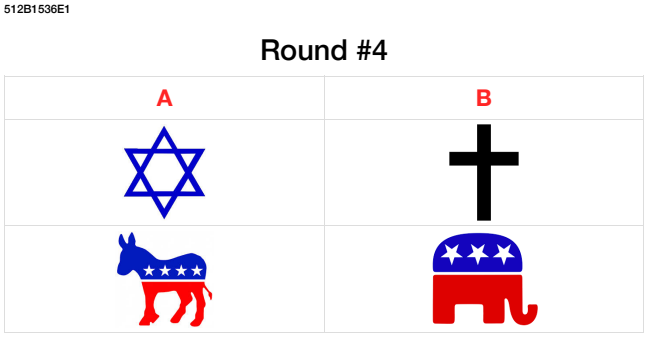
To contribute to the community of researchers wishing to conduct these experiments in similar contexts we have developed tools and made them open source and available for immediate use. The first of these resources is JavaScript code that works in conjunction with the Qualtrics offline application to store attribute variables and randomly assign those variables without relying on a web scripts source. The second tool is an application that allows users to assign images or text to attribute levels.[4] The application produces randomized conjoint profiles with these images in a PDF. Each PDF has a unique ID, and a csv file recording the attribute-levels presented in every round of each PDF. In this section we describe these resources in greater detail.

3.1. Conjoint Offline

This resource is appropriate for use in the instances described above, where internet access is unreliable and standard tools cannot be used. The code we developed allows you to customize a conjoint design for offline use. In the instructions, we explain how to integrate the code for use with the Qualtrics offline application. Following standard practices, the code randomizes the order of the attributes once for every respondent. In each round the attribute-levels are randomly assigned. This tool presents conjoint profiles in text form. We recommend using this tool on a tablet or other mobile device. In some rare cases, with a literate and highly educated sample, respondents may be able to view the text profile tables on the tablets directly and then respond. In most cases, where respondents are less educated, enumerators will need to read the randomized conjoint profiles on the tablet and then use visual aids to present the profiles in image form to respondents. In either case, researchers need not rely on enumerators to implement the randomization themselves. This code can also be adapted for other offline survey applications that are JavaScript compatible. In a following section we illustrate how this tool was used in rural Tanzania.

3.2. Conjoint with Images

Figure 2: Sample candidate profiles that include party and religion attributes pictorially represented



[3] ICT Facts and Figures 2015.

[4] A hosted app is ready for immediate use on the MIT GOV/LAB website and all code/files referred to in this publication are available at: Meyer and Rosenzweig (2016) http://dx.doi.org/10.7910/DVN/DURDD3.

The second tool we developed is open source code that builds an app which generates conjoint profiles using images (or text) to represent attribute-levels.[5] After uploading the desired images, designating the number of rounds, and names of choices, it produces PDFs that have two profiles per round. These profiles are comprised of randomly assigned attribute-levels in image form. These PDFs are for use in one-on-one surveys with respondents. The PDF packets produce unique IDs for each conjoint packet and are printed on each page. The profile data (attribute-levels) presented in each round are stored in a csv file that can be downloaded from the application. We recommend using these profiles on paper with respondents and easily connecting them to the stored data from the application and other survey questions recorded in tablets with the unique IDs. Figure 2 provides an example of one round of conjoint profiles. In the next section we describe common challenges to implementing conjoint experiments in developing countries and illustrate, using an example from Tanzania, how to overcome these challenges with the tools we created.

4. Challenges of Conjoint: Evidence from Tanzania

In this section we describe some common challenges to implementing conjoint experiments in developing countries. We illustrate how the tools we created helped to overcome these problems for a conjoint experiment in Tanzania.

4.1. Randomizing conjoint profiles without an internet connection

In many developing countries cell network is sparse and data coverage rare. In these cases implementing a conjoint experiment using resources that require an internet connection is infeasible. However, researchers often want to collect data electronically, and therefore require a way to create randomized conjoint profiles and store these data electronically without the use of web-scripts. Following standard practice for creating conjoint profiles, the order of the attributes should be randomized for each respondent and the levels of each attribute randomly assigned each round (Hainmueller, Hopkins, and Yamamoto, 2014). Relying on enumerators to randomize conjoint profiles in this way and trusting that they record the correct attributes shown is risky and cumbersome. The tool we developed works in conjunction with the Qualtrics offline app to randomize the order of the attributes and each attribute-level, and stores these variables in the data that can be downloaded from the Qualtrics site. This tool makes implementing conjoint experiments and data collection on tablets or other hand-held devices without an internet connection feasible.

Just prior to the October 2015 general elections in Tanzania, Rosenzweig and Tsai, in collaboration with Twaweza, ran a conjoint experiment in 82 rural villages. The researchers wanted to know what candidate attributes influence the probability that voters will support that candidate? "The hypothetical candidate profiles varied on the basis of six attributes: religion, party, tribe, past performance towards the community, past performance towards individuals, and credibility of promises. Each of the six attributes are binary and take on one of two values" (see Table 1) (Rosenzweig and Tsai, N.d.).

Table 1: Attributes and levels of hypothetical candidates used in Tanzania

Attribute	**Level 1**	**Level 2**
Religion	Muslim	Christian
Party	CCM	Opposition
Tribe	Chagga	Sukuma
Past Performance --Community	Gave community nothing	Gave community social services
Past Performance --Individuals	Gave you nothing	Gave you money for social services
Promises	Has promises and a plan	Has promises but no plan

Using our tool with the Qualtrics offline app, the researchers in Tanzania were able to create randomized candidate profiles on tablets. "In one treatment condition in Tanzania the enumerator conducted a conjoint experiment with a group of five participants simultaneously. The enumerator sees the randomized conjoint profiles, in text form, on the tablet and then presents laminated index cards, with images to represent the candidate attributes, to the group. The enumerator uses these visual aids to tell the story of each candidate and presents the choice task as a 'game'. By referencing specific features of the images the respondents are better able to remember what each picture represents. The enumerator uses the tablet to record how each participant voted using the numeric ID tags participants wear. The tool records the profiles the group saw and any other variables included in the survey. Each night, back at the hotel, the supervisor sets up a wifi hotspot and uploads the data from the tablets to the server" (Rosenzweig and Tsai N.d.).

4.2. Self-administered surveys with illiterate respondents

In general, self-administered surveys are difficult to conduct in developing countries for several reasons, including low levels of education. When we want surveys to mirror the real-world processes and behaviors they are trying to pre-

[5] The app can also produce conjoint profiles with text that are easily printed, if that is desired.

dict, we need respondents to answer questions in private. In the case of voting, ideally respondents should use secret ballots instead of reporting their answers directly to enumerators, since enumerators can affect responses (Adida et al. 2016) and social desirability bias is a constant concern.

In Tanzania approximately 32% of the population is illiterate.[6] "Of the sampled survey respondents 6% had no formal schooling, 8% had only some primary school education, and 62% finished primary school but did not pursue further education" (Rosenzweig and Tsai N.d.).. Although it was impractical to have respondents self-administer the entire survey in Tanzania, the researchers were able to have respondents record their voting preferences in the conjoint experiment in secret using the tools we created.

Using the tool that produces conjoint profiles using images to represent attributes, respondents were able to use paper and pen to record their voting decisions in secret.[7] After presenting the candidate profiles and explaining how to fill out the ballot to the respondent, the enumerator leaves the room while the respondent votes by circling their preferred candidate ("A" or "B"), or "X" if they want to abstain.

In addition to measuring if respondents voted and for which candidate, respondents were also asked to rate both candidates. Instead of the traditional feeling thermometers and Likert scales that prove confusing for innumerate respondents, the researchers developed a 'bucket scale' measure. "Respondents indicate how much water they would fill a bucket with to represent how well they think the candidate would be at getting things done if elected. Respondents are instructed on this measure and then asked to draw their water line before putting their secret ballots in the envelop" (Rosenzweig and Tsai N.d.).[8] Obtaining multiple measures of voting preferences in these contexts allows researchers to verify respondent comprehension with the task.

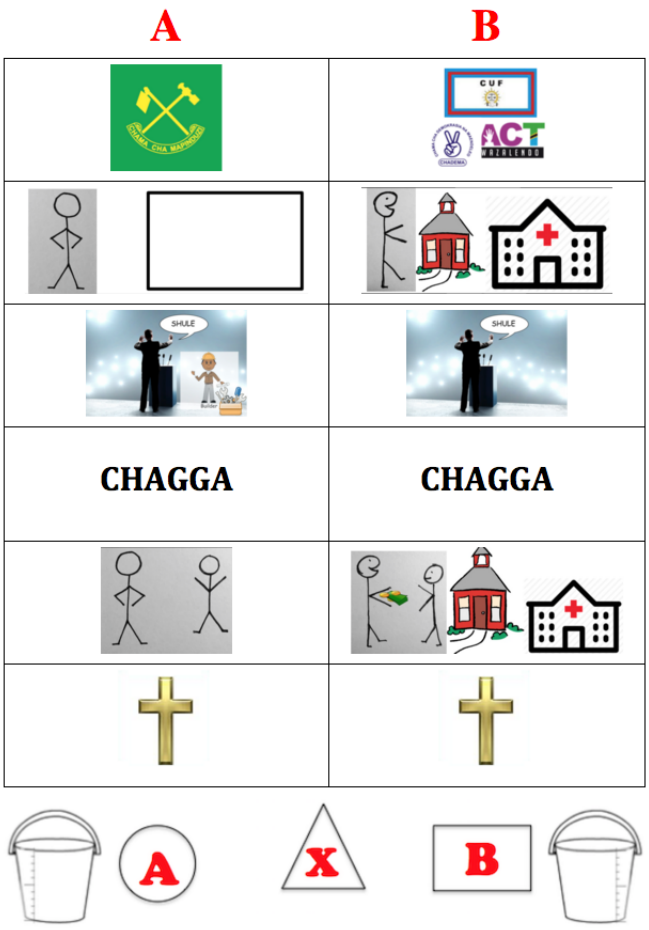
The tool we created allows researchers to easily combine survey data collected with the tablet and the conjoint voting responses written on paper. The app produces a unique *packet ID* and round number on each sheet of paper, making it easy to connect respondents' secret ballot responses, with pre-populated conjoint profiles produced by the app and survey questions recorded in the tablet.[9] The enumerator can also honestly tell the respondent that they will never know how they vote since the data entry of voting responses and data analysis is conducted by the researcher, using the packet ID as a unique identifier.

5. Conclusion

Conjoint analysis is gaining popularity in the social sciences and being applied in a variety of contexts. We propose that many of these contexts require adaptation of the standard tools used to implement conjoint experiments. We provide two such tools that allow researchers to design a conjoint choice task using offline survey software and a tool that allows easy randomization and presentation of conjoint profiles using images to represent attribute-levels.

Appendix

Figure 3: Sample conjoint profile created with the app and used in Tanzania



Acknowledgments

[6] Unicef 2013.

[7] The researchers tested using the tablet to record respondent votes but found that this did not work well. Respondents were confused by and often suspicious of the tablets—they even thought the tablets were being used to capture their fingerprints so that they would not be able to vote in the upcoming general election. This is likely context specific, since Tanzania had just introduced a new biometric voter registration system using fingerprints, but something of which to be wary.

[8] See Figure 3 in the Appendix for an example of the secret ballot used.

[9] Enumerators record the packet ID in the tablet before the beginning of the survey.

We thank Lily L. Tsai and Teppei Yamamoto for their helpful comments and suggestions.

References

Adida, L. Claire, Karen E. Ferree, Daniel N. Posner, and Amanda Lea Robinson. 2016. "Who's Asking? Interviewer Coethnicity Effects in African Survey Data." *Comparative Political Studies*. Forthcoming. `http://dx.doi.org/10.1177/0010414016633487` (August 1, 2016).

Berinsky, Adam, Tesalia Rizzo, Leah R. Rosenzweig, and Elisha Heaps. 2015. "Attribute Affinity: US Natives' Attitudes Toward Immigrants." Working paper.

Carlson, Elizabeth. 2015. "Ethnic Voting and Accountability in Africa: A Choice Experiment in Uganda." *World Politics* 67(2): 353-85.

Hainmueller, Jens, and Daniel J. Hopkins. 2014. "The Hidden American Immigration Consensus: A Conjoint Analysis of Attitudes toward Immigrants." *American Journal of Political Science* 59(3): 529-48.

Hainmueller, Jens, Daniel J. Hopkins, and Teppei Yamamoto. 2014. "Causal Inference in Conjoint Analysis: Understanding Multidimensional Choices via Stated Preference Experiments." *Political Analysis* 22(1): 1-30.

Hainmueller, Jens, Dominik Hangartner, and Teppei Yamamoto. 2015. "Validating Vignette and Conjoint Survey Experiments against Real-world Behavior." *Proceedings of the National Academy of Sciences* 112(8): 2395-400.

Hartman, Alexandra, and Ben Morse. 2015. "Wartime Violence, Empathy, and Inter-group Altruism: Theory and evidence from the Ivoirian refugee crisis in Liberia." Working paper.

Horiuchi, Yusaku, Daniel M Smith, and Teppei Yamamoto. 2015. "Identifying Multidimensional Policy Preferences of Voters in Representative Democracies: A Conjoint field experiment in Japan." *SSRN* 2627907.

Meyer, Alexander, and Leah R. Rosenzweig. 2016. "Replication Data for: Conjoint Analysis Tools for Developing Country Contexts." `http://dx.doi.org/10.7910/DVN/DURDD3` Harvard Dataverse, V1. (August 1, 2016).

Rosenzweig, Leah R., and Lily L. Tsai. 2016. "Using Facebook for Online Experiments." Working paper.

Rosenzweig, Leah R., and Lily L. Tsai. N.d. "Evaluating Candidates: A Conjoint Experiment in Tanzania." Working paper.

Visualize Dynamic Simulations of Autoregressive Relationships in R

Christopher Gandrud
City University London and Hertie School of Governance
christopher.gandrud@city.ac.uk

Laron K. Williams
University of Missouri
williamslaro@missouri.edu

Guy D. Whitten
Texas A&M University
g-whitten@tamu.edu

Two recent trends in the social sciences have drastically improved the interpretation of statistical models. The first trend is researchers providing substantively meaningful quantities of interest when interpreting models rather than putting the burden on the reader to interpret tables of coefficients (King et al. 2000). The second trend is a movement to more completely interpret and present the inferences available from one's model. This is seen most obviously in the case of time-series models with an autoregressive series, where the effects of an explanatory variable have both short- and a long-term components. A more complete interpretation of these models therefore requires additional work ranging from the presentation of long-term multipliers (De Boef and Keele 2008) to dynamic simulations (Williams and Whitten 2012).

These two trends can be combined to allow scholars to easily depict the long-term implications from estimates of dynamic processes through simulations. Dynamic simulations can be employed to depict long-run simulations of dynamic processes for a variety of substantively-interesting scenarios, with and without the presence of exogenous shocks. We introduce dynsim (Gandrud et al. 2016) which makes it easy for researchers to implement this approach in R.[1]

1. Dynamic simulations

Assume that we estimate the following partial adjustment model: $Y_t = \alpha_0 + \alpha_1 Y_{t-1} + \beta_0 X_t + \epsilon_t$, where Y_t is a continuous variable, X_t is an explanatory variable and ϵ_t is a random error term. The short-term effect of X_1 on Y_t is simple, β_0. This is the inference that social science scholars most often make, and unfortunately, the only one that they usually make (Williams and Whitten 2012). However, since

[1] There is also a Stata (StataCorp. 2015) implementation documented in Williams and Whitten (2011).

the model incorporates a lagged dependent variable (Y_{t-1}), a one-unit change in X_t on Y_t also has a long-term effect by influencing the value of Y_{t-1} in future periods. The appropriate way of calculating the long-term effect is with the long-term multiplier, or $\kappa_1 = \frac{\beta_0}{(1-\alpha_1)}$. We can then use the long-term multiplier to calculate the total effect that X_t has on Y_t distributed over future time periods. Of course, the long-term multiplier will be larger as β_0 or α_1 gets larger in size.

We can use graphical depictions to most effectively communicate the dynamic properties of autoregressive time series across multiple time periods. The intuition is simple. For a given scenario of values of the explanatory variables, calculate the predicted value at time t: $\tilde{y} = \boldsymbol{X_C}\tilde{\beta} + \tilde{\epsilon}$, where $\tilde{\beta}$ is a vector of simulated effect coefficients, $\boldsymbol{X_C}$ is a matrix of user-specified values of variables, including y_{t-1}, and $\tilde{\epsilon}$ is one draw from $N(0, \tilde{\sigma}^2)$. At each subsequent observation of the simulation, the predicted value of the previous scenario ($\tilde{y}|\mathbf{X_C}$) replaces the value of y_{t-1} to calculate $\tilde{y}_t$. Inferences such as long-term multipliers and dynamic simulations are based on *estimated* coefficients that are themselves uncertain. It is therefore very important to also present these inferences with the necessary measures of uncertainty (such as confidence intervals).

Dynamic simulations offer a number of inferences that one cannot make by simply examining the coefficients. First, one can determine whether or not the confidence interval for one scenario overlaps across time, suggesting whether or not there are significant changes over time. Second, one can determine whether the confidence intervals of different scenarios overlap at any given time period, indicating whether the scenarios produce statistically different predicted values. Finally, if one includes exogenous shocks, then one can determine the size of the effect of the exogenous shock as well as how quickly the series then returns to its pre-shock state. These are all invaluable inferences for testing one's theoretical expectations.

2. dynsim process and syntax

Use the following four step process to simulate and graph autoregressive relationships with dynsim:

1. Estimate the linear model using the core R function `lm`.
2. Set up starting values for simulation scenarios and (optionally) shock values at particular iterations (e.g. points in simulated time).
3. Simulate these scenarios based on the estimated model using the `dynsim` function.
4. Plot the simulation results with the `dynsimGG` function.

Before looking at examples of this process in action, let's look at the `dynsim` and `dynsimGG` syntax.

The `dynsim` function has seven arguments. The first–`obj`–is used to specify the model object. The lagged dependent variable is identified with the `ldv` argument. The object containing the starting values for the simulation scenarios is identified with `scen`. `n` allows you to specify the number of simulation iterations. These are equivalent to simulated 'time periods'. `scen` sets the values of the variables in the model at 'time' $n = 0$. To specify the level of statistical significance for the confidence intervals use the `sig` argument. By default it is set at `0.95` for 95 percent significance levels. Note that `dynsim` currently depicts uncertainty in the systematic component of the model, rather than forecast uncertainty. The practical implication of this is that the confidence intervals will not grow over the simulation iteration. The number of simulations drawn for each point in time–i.e. each value of `n`–is adjusted with the `num` argument. By default 1,000 simulations are drawn. Adjusting the number of simulations allows you to change the processing time. There is a trade-off between the amount of time it takes to draw the simulations and the resulting information you have about about the simulations' probability distributions (King et al. 2000, p. 349). Finally the object containing the shock values is identified with the `shocks` argument.

Objects for use with `scen` can be either a list of data frames–each data frame containing starting values for a different scenario – or a data frame where each row contains starting values for different scenarios. In both cases, the data frames have as many columns as there are independent variables in the estimated model. Each column should be given a name that matches the names of a variable in the estimation model. If you have entered an interaction using `*`[2] then you only need to specify starting values for the base variables not the interaction term. The simulated values for the interaction will be found automatically.

`shocks` objects are data frames with the first column called `times` containing the iteration number (as in `n`) when a shock should occur. Note each shock must be at a unique time that cannot exceed `n`. The following columns are named after the shock variable(s), as they are labeled in the model. The values will correspond to the variable values at each shock time. You can include as many shock variables as there are variables in the estimation model. Again only values for the base variables, not the interaction terms, need to be specified.

Once the simulations have been run you will have a `dynsim` class object. Because `dynsim` objects are also data frames you can plot them with any available method in R. They contain at least seven columns:

- `scenNumber`: The scenario number.

[2]For example, an interaction between `Var1` and `Var2` entered into the model as `Var1*Var2`.

- `time`: The time points.
- `shock. ...`: Optional columns containing the values of the shock variables at each point in `time`.
- `ldvMean`: Mean of the simulation distribution.
- `ldvLower`: Lower bound of the simulation distribution's central interval set with `sig`.
- `ldvUpper`: Upper bound of the simulation distribution's central interval set with `sig`.
- `ldvLower50`: Lower bound of the simulation distribution's central 50 percent interval.
- `ldvUpper50`: Upper bound of the simulation distribution's central 50 percent interval.

The `dynsimGG` function is the most convenient plotting approach. This function draws on ggplot2 (Wickham and Chang 2015) to plot the simulation distributions across time. The distribution means are represented with a line. The range of the central 50 percent interval is represented with a dark ribbon. The range of the interval defined by the `sig` argument in `dynsim`, e.g. 95%, is represented with a lighter ribbon.

The primary `dynsimGG` argument is `obj`. Use this to specify the output object from `dynsim` that you would like to plot. The remaining arguments control the plot's aesthetics. For instance, the size of the central line can be set with the `lsize` argument and the level of opacity for the lightest ribbon[3] with the `alpha` argument. Please see the ggplot2 documentation for more details on these arguments. You can change the color of the ribbons and central line with the `color` argument. If only one scenario is plotted then you can manually set the color using a variety of formats, including hexadecimal color codes. If more than one scenario is plotted, then select a color palette from those available in the RColorBrewer package (Neuwirth 2014).[4] The plot's title, y-axis and x-axis labels can be set with the `title`, `ylab`, and `xlab` arguments, respectively.

There are three arguments that allow you to adjust the look of the scenario legend. `leg.name` allows you to choose the legend's name and `leg.labels` lets you change the scenario labels. This must be a character vector with new labels in the order of the scenarios in the `scen` object. `legend` allows you to hide the legend entirely. Simply set `legend = FALSE`.

Finally, if you included shocks in your simulations you can use the `shockplot.var` argument to specify *one* shock variable's fitted values to include in a plot underneath the main plot. Use the `shockplot.ylab` argument to change the y-axis label.

The output from `dynsimGG` is generally a ggplot2 `gg` class object.[5] Because of this you can further change the aesthetic qualities of the plot using any relevant function from ggplot2 using the `+` operator. You can also convert them to interactive graphics with `ggplotly` from the plotly (Sievert et al. 2016) package.

3. Examples

The following examples demonstrate how dynsim works. They use the Grunfeld (1958) data set. It is included with dynsim. To load the data use:

```
data(grunfeld, package = "dynsim")
```

The linear regression model we will estimate is:

$$I_{it} = \alpha + \beta_1 I_{it-1} + \beta_2 F_{it} + \beta_3 C_{it} + \mu_{it} \quad (1)$$

where I_{it} is real gross investment for firm i in year t. I_{it-1} is the firm's investment in the previous year. F_{it} is the real value of the firm and C_{it} is the real value of the capital stock.

In the `grunfeld` data set, real gross investment is denoted `invest`, the firm's market value is `mvalue`, and the capital stock is `kstock`. There are 10 large US manufacturers from 1935-1954 in the data set (Baltagi 2001). The variable identifying the individual companies is called `company`. We can easily create the investment one-year lag within each `company` group using the `slide` function from the DataCombine package (Gandrud 2016). Here is the code:

```
library(DataCombine)

grunfeld <- slide(grunfeld, Var = "invest", GroupVar = "company",
                  TimeVar = "year", NewVar = "InvestLag")
```

The new lagged variable is called `InvestLag`. The reason we use `slide` rather than R's core `lag` function is that the latter is unable to lag a grouped variable. You could of course use any other appropriate function to create the lags.

3.1. Dynamic simulation without shocks

Now that we have created our lagged dependent variable, we can begin to create dynamic simulations with dynsim by estimating the underlying linear regression model using `lm`, i.e.:

```
M1 <- lm(invest ~ InvestLag + mvalue + kstock, data = grunfeld)
```

The resulting model object–`M1`–is used in the `dynsim` function to run the dynamic simulations. We first create a list object containing data frames with starting values for each simulation scenario. Imagine we want to run three contrasting scenarios with the following fitted values:

- **Scenario 1**: mean lagged investment, market value and capital stock held at their 95th percentiles,

[3] The darker 50 percent central interval ribbon is created by essentially doubling the opacity set by `alpha`.
[4] To see the full list, after loading RColorBrewer in your R session, simply enter `brewer.pal.info` into your console.
[5] This is not true if you include a shock plot.

- **Scenario 2**: all variables held at their means,
- **Scenario 3**: mean lagged investment, market value and capital stock held at their 5th percentiles.

We can create a list object for the `scen` argument containing each of these scenarios with the following code:

```
> attach(grunfeld)
Scen1 <- data.frame(InvestLag = mean(InvestLag, na.rm = TRUE),
                    mvalue = quantile(mvalue, 0.95),
                    kstock = quantile(kstock, 0.95))
Scen2 <- data.frame(InvestLag = mean(InvestLag, na.rm = TRUE),
                    mvalue = mean(mvalue),
                    kstock = mean(kstock))
Scen3 <- data.frame(InvestLag = mean(InvestLag, na.rm = TRUE),
                    mvalue = quantile(mvalue, 0.05),
                    kstock = quantile(kstock, 0.05))
detach(grunfeld)

ScenComb <- list(Scen1, Scen2, Scen3)
```

To run the simulations without shocks use:

```
library(dynsim)

Sim1 <- dynsim(obj = M1, ldv = "InvestLag", scen = ScenComb, n = 20)
```

3.2. Dynamic simulation with shocks

Now we include fitted shock values. In particular, we will examine how a company with capital stock in the 5th percentile is predicted to change its gross investment when its market value experiences shocks compared to a company with capital stock in the 95th percentile. We will use market values for the first company in the `grunfeld` data set over the first 15 years as the shock values. To create the shock data use the following code:

```
# Keep only the mvalue for the first company for the first 15 years
grunfeldsub <- subset(grunfeld, company == 1)
grunfeldshock <- grunfeldsub[1:15, "mvalue"]

# Create data frame for the shock argument
grunfeldshock <- data.frame(times = 1:15, mvalue = grunfeldshock)
```

Now add `grunfeldshock` to the `dynsim` `shocks` argument.

```
Sim2 <- dynsim(obj = M1, ldv = "InvestLag", scen = ScenComb, n = 15,
               shocks = grunfeldshock)
```

Interactions between the shock variable and another exogenous variable can also be simulated for. To include, for example, an interaction between the firm's market value (the shock variable) and the capital stock (another exogenous variable) we need to rerun the parametric model like so:

```
M2 <- lm(invest ~ InvestLag + mvalue*kstock, data = grunfeld)
```

We then use `dynsim` as before. The only change is that we use the fitted model object `M2` that includes the interaction.

```
Sim3 <- dynsim(obj = M2, ldv = "InvestLag", scen = ScenComb, n = 15,
               shocks = grunfeldshock)
```

3.3. Plotting simulations

The easiest and most effective way to communicate `dynsim` simulation results is with the package's built-in plotting capabilities, e.g.:

```
dynsimGG(Sim1)
```

We can make a number of aesthetic changes. The following code adds custom legend labels, the 'orange-red' color palette–denoted by `OrRd`–, and relabels the y-axis to create Figure 1.

```
Labels <- c("95th␣Percentile", "Mean", "5th␣Percentile")

dynsimGG(Sim1, leg.name = "Scenarios", leg.labels = Labels,
               color = "OrRd",
               ylab = "Predicted␣Real␣Gross␣Investment\n")
```

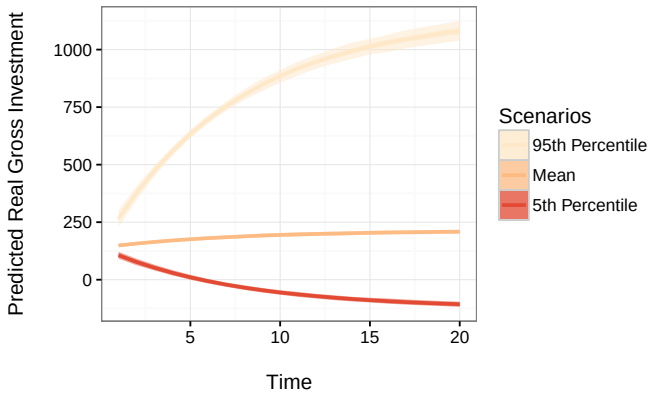


Figure 1: Three Dynamic Simulations Plotted with Custom Scenario Labels and Color Palette

When plotting simulations with shock values another plot can be included underneath the main plot showing one shock variable's fitted values. To do this use the `shockplot.var` argument to specify which variable to plot. Use the `shockplot.ylab` argument to change the y-axis label. For example, the following code creates Figure 2:

```
dynsimGG(Sim2, leg.name = "Scenarios", leg.labels = Labels,
               color = "OrRd",
               ylab = "Predicted␣Real␣Gross␣Investment\n",
               shockplot.var = "mvalue",
               shockplot.ylab = "Firm␣Value")
```

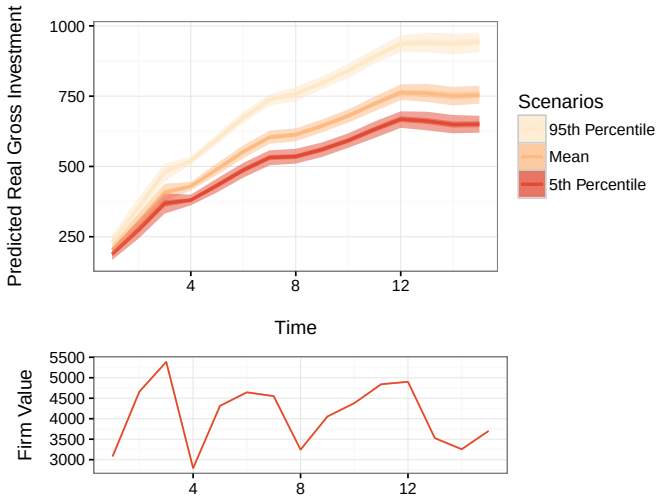


Figure 2: An Example of a Dynamic Simulation with the Inclusion of a Shock Variable

4. Summary

We have demonstrated how the R package dynsim makes it easy to implement Williams and Whitten's (2012) approach to more completely interpreting results from autoregressive time-series models where the effects of explanatory variables have both short- and long-term components. Hopefully, this will lead to more meaningful investigations and more useful presentations of results estimated from these relationships.

References

Baltagi, Badi H. 2001. *Econometric Analysis of Panel Data.* Chichester, UK: Wiley and Sons.

De Boef, Suzanna, and Keele, Luke. 2008. "Taking Time Seriously." *American Journal of Political Science* 52(1): 184-200.

Gandrud, Christopher. 2016. *DataCombine: Tools for Easily Combining and Cleaning Data Sets.* R package version 0.2.21. `https://CRAN.R-project.org/package=DataCombine` (August 1, 2016).

Gandrud, Christopher, Williams, Laron K., and Whitten, Guy D. 2016. *dynsim: Dynamic Simulations of Autoregressive Relationships.* R package version 1.2.2. `https://CRAN.R-project.org/package=dynsim` (August 1, 2016).

Grunfeld, Yehuda. (1958). "The Determinants of Corporate Investment." Ph.D.diss. The University of Chicago.

King, Gary, Tomz, Michael, and Wittenberg, Jason. 2000. "Making the Most of Statistical Analyses: Improving Interpretation and Presentation." *American Journal of Political Science* 44(2): 347-61.

Neuwirth, Erich. 2014. *RColorBrewer: ColorBrewer palettes.* R package version 1.1-2. `https://CRAN.R-project.org/package=RColorBrewer` (August 1, 2016).

Sievert, Carson, Chris Parmer, Toby Hocking, Scott Chamberlain, Karthik Ram, Marianne Corvellec, and Pedro Despouy. 2016. *plotly: Create Interactive Web Graphics via 'plotly.js'.* R package version 3.4.13. `https://cran.r-project.org/package=plotly` (August 1, 2016).

StataCorp. 2015. *Stata Statistical Software: Release 14.* College Station, TX: StataCorp LP.

Wickham, Hadley, and Winston Chang. 2015. *ggplot2: An implementation of the Grammar of Graphics.* R package version 1.0.1. `https://cran.r-project.org/package=ggplot2` (August 1, 2016).

Williams, Laron K., and Guy D. Whitten. 2011. "Dynamic Simulations of Autoregressive Relationships." *The Stata Journal* 11(4): 577-88.

Williams, Laron K., and Guy D. Whitten. 2012. "But Wait, There's More! Maximizing Substantive Inferences from TSCS Models." *Journal of Politics* 74(3): 685-93.

Embrace your Fallibility: Thoughts on Code Integrity

Nicholas Eubank
Stanford University
nicholaseubank@stanford.edu

Two years ago, I wrote a piece about my experiences over two years testing the code for papers being published in the Quarterly Journal of Political Science, which found problems in the code of many papers. The piece was first published in The Political Methodologist (Eubank 2014), and later in PS: Politics and Political Science (Eubank 2016). This piece is an extension of that article based on conversations that article sparked and my own experiences over the past two years.

It's natural to think that the reason we find problems in the code behind published papers is carelessness or inattention on behalf of authors, and that the key to minimizing problems in our code is to be more careful. The truth, I have come to believe, is more subtle: humans are effectively

incapable of writing error-free code, and that if we wish to improve the quality of the code we write, we must start learning *and teaching* coding skills that help maximize the probability our mistakes will be found and corrected.

I myself once firmly believed the fallacy that the key to preventing errors was "to be more careful." Indeed, I fear this belief may have colored the tone of of my past work on this subject in unproductive ways. Over the last few years, however, my research has brought me into close contact with computer scientists, and I discovered that computer scientists' mentality about programming is fundamentally different from the mental model I had been carrying around. Computer scientists *assume* programmers will make mistakes, and instead of chiding people to "just be careful," they have developed a battery of practices to address the problem. These practices – often referred to as "defensive programming" – are designed to (a) minimize the probability mistakes occur and (b) maximize the probability that mistakes that do occur are caught.

If we as social scientists wish to continue adopting more and more computational techniques in our research, I feel this is a mentality we must also adopt. This will not always be easy. Defensive programming is a skill, and if it is to become a part of the discipline it will require effort on behalf of researchers to learn, implement, and most important teach these skills to the next generation. But I think this is necessary to ensure the integrity of our work.

With that in mind, I would like to advocate for two changes to our approach to the computational component of social science.

First, I think we must adopt a number of practices from defensive programming in our own code. This piece lays out a few simple practices that I think are most applicable and practical for social scientists, both for individuals and co-authors working collaboratively. They aren't meant as complete tutorials, but rather as illustrations of the type of practices I think should be promoted.

Second, I think we need to begin teaching these practices to students. Too often, students are either expected to figure out how to program on their own during their econometrics classes, or they are offered short, graduate-student-led workshops to introduce basic skills. Coding is now too central to our discipline to be given this second-tier status in our curriculum. If we are going to expect our students to engage in computational research, it is our obligation to equip them with the tools they need to stay out of danger.

Together, I think these two changes will improve the integrity of our research as coding becomes ever more central to our discipline. Will they preclude errors completely? Unlikely – even when perfectly employed, "defensive programming" is not fool-proof, and there will always be problems that these tools will not catch. But at least with these tools we can start to minimize the likelihood of errors, especially large ones.

This piece is organized into five sections. Section 1 presents an overview of specific defensive programming practices we can all implement in our own code. Section 2 then lays out some examples of how "defensive programming" principles can guide workflow in collaborative projects. Finally, after introducing these concrete skills, I offer a few reflections on the implications of the "defensive programming" paradigm for third-party review of code by academic journals in Section 3, and for how the discipline responds to errors in Section 4. Section 5 concludes with a short list of other resources, to which additional suggestions are welcome!

1. Defensive Programming Practices

Defensive Programming Practice 1: Adding Tests

If we could only adopt one practice to improve the quality of our code, my vote would be for the addition of tests.

Tests are simple true-false statements users place in their code. A test checks for a certain condition (like whether the sample size in a regression is what you expect), and if the condition is not met, stops your code and alerts you to the problem.

Right now, many users may say "Yeah, I always check that kind of stuff by hand when I'm writing my code. Why do I need to add tests?"

The answer is four-fold:

- **Tests are executed *every time your code is run.*** Most of us check things the first time we write a piece of code. But days, weeks, or months later, we may come back, modify code the occurs earlier in our code stream, and then just re-run the code. If those changes lead to problems in later files, we don't know about them. If you have tests in place, then those early changes will result in an error in the later files, and you can track down the problem.
- **It gets you in the habit of always checking.** Most of us only stop to check aspects of our data when we suspect problems. But if you become accustomed to writing a handful of tests at the bottom of every file – or after ever execution of a certain operation (I'm trying to always including them after a merge), we get into the habit of *always* stopping to think about what our data should look like.
- **Catch your problems faster.** This is less about code integrity than sanity, but a great upside to tests is that they ensure that if a mistake slips into your code, you become aware of it quickly, making it easier to identify and fix the changes that caused the problem.
- **Tests catch more than anticipated problems.** When problems emerge in code, they often manifest

in lots of different ways. Duplicate observations, for example, will not only lead to inaccurate observation counts, but may also give rise to bizarre summary statistics, bad subsequent merges, etc. Thus adding tests not only guards against errors we've thought of, but may also guard against errors we don't anticipate during the test writing process.

Writing Tests

Tests are easy to write in any language. In Stata, for example, tests can be performed using the `assert` statement. For example, to test whether your data set has 100 observations or that a variable meant to hold percentages has reasonable values, you could write:

```
* Test if data has 100 observations
count
assert `r(N)'==100

*␣Test␣variable␣percent_employed␣has␣reasonable␣values
assert␣percent_employed␣>␣0␣&␣percent_employed␣<␣100
```

Similarly in R, one could do the same tests on a data.frame `df` using:

```
# Test if data has 100 observations
stopifnot(nrow(df)==100)

# Test variable has reasonable values
stopifnot(df$percent_employed > 0 & df$percent_employed < 100)
```

Defensive Programming Practice 2: Never Transcribe

We've already covered tricks to maximize the probability we catch our mistakes, but how do we minimize the probability they will occur?

If there is anything we learned at the *QJPS*, it is that authors should never transcribe numbers from their statistical software into their papers by hand. This was *easily* the largest source of replication issues we encountered, as doing so introduced two types of errors:

- **Mis-Transcriptions:** Humans just aren't built to transcribe dozens of numbers by hand reliably. If the error is in the last decimal place, it doesn't mean much, but when a decimal point drifts or a negative sign is dropped, the results are often quite substantively important.
- **Failures to Update:** We are constantly updating our code, and authors who hand transcribe their results often update their code and forget to update all of their results, leaving old results in their paper.

How do you avoid this problem? For LaTeXusers, I strongly suggest tools that export `.tex` files that can be pulled directly into LaTeXdocuments. I also suggest users not only do this for tables – which is increasingly common – but also statistics that appear in text. In your code, generate the number you want to cite, convert it to a string, and save it as a `.tex` file (e.g. `exported_statistic.tex`). Then in your paper, simply add a `\input{exported_statistic.tex}` call, and LaTeX will insert the contents of that `.tex` file verbatim into your paper.

Directly integrating output is somewhat harder to do if you work in Word, but is still feasible. For example, most packages that generate `.tex` files also have options to export to `.txt` or `.rtf` files that you can easily use in Word. `write.table()` in R or `esttab` in Stata, for example, will both create output of this type you can put in a Word document. These tools can be used to generate tables can either be (a) copied whole-cloth into Word by hand (minimizing the risk of mis-transcriptions that may occur when typing individual values), or (b) using Word's Link to Existing File feature to connect your Word document to the output of your code in a way that ensures the Word doc loads the most recent version of the table every time Word is opened. Some great tips for combining R with Word can be found here (Ejdemyr 2015).

Defensive Programming Practice 3: Style Matters

Formatting isn't just about aesthetics, it also makes it easier to read your code and thus recognize potential problems. Here are a few tips:

- **Use informative variable names.** Don't call something `var212` if you can call it `unemployment_percentage`. Informative names require more typing, but they make your code so much easier to read. Moreover, including units in your variables names (percentage, km, etc.) can also help avoid confusion.
- **Comment!** Comments help in two ways.
  - First, and most obviously, they make it easy to figure out what's going on when you come back to code days, weeks, or months after it was originally written.
  - Second, it forces you to think about what you're doing in substantive terms ("This section calculates the share of people within each occupation who have college degrees") rather than just in programming logic, which can help you catch substantive problems with code that may run without problems but will not actually generate the quantity of interest.
- **Use indentation.** Indentation is a way of visually representing the logical structure of code – use it to your advantage!
- **Let your code breathe.** In general, you should put a space between every operator in your code, and feel free to use empty lines. Space makes your code more readable, as illustrated in the following examples:

```
# Good
average <- mean(feet / 12 + inches, na.rm = TRUE)

# Bad
average<-mean(feet/12+inches,na.rm=TRUE)
```

A full style guide for R can be found here (Wickham N.d.), and a Stata style guide can be found here (Hill 2015).

Defensive Programming Practice 4: Don't Duplicate Information

Tricks to minimize the probability of errors often require a little more sophisticated programming, so they won't be for everyone (tests, I feel, are more accessible to everyone). Nevertheless, here's another valuable practice: **Never replicate information**.

Information should only be expressed once in a file. For example, say you want to drop observations if the value of a set of variables is greater than a common cutoff (just assume this is something you want to do – the specific operation is not important). In Stata, for example, you could do this by:

```
drop if var1 < 110
drop if var2 < 110
drop if var3 < 110
```

And indeed, this would work. But suppose you decided to change that cutoff from 110 to 100. The way this is written, you've opened yourself up to the possibility that in trying to change these cutoffs, you may change two of these but forget the third (something especially likely if the uses of the cutoff aren't all in exactly the same place in your code).

A better way of expressing this that avoids this possibility is:

```
local cutoff = 110
drop if var1 < `cutoff'
drop␣if␣var2␣<␣`cutoff'
drop if var3 < `cutoff'
```

Written like this, if you ever decide to go back and change the common cutoff, you only have to make one change, and there's no way to make the change in some cases but forget others.

2. Collaboration

Until now, the focus of this piece has been on *individual* coding practices that minimize the risk of errors. But as social science becomes increasingly collaborative, we also need to think about how to avoid errors in collaborative projects.

In my experience, the way most social scientists collaborate on code (myself included, historically) is to place their code in a shared folder (like Dropbox or Box) and have co-authors work on the same files.

There are a number of problems with this strategy, however:

1. Participants can ever be certain about the changes the other author has made. Changes may be obvious when an author adds a new file or large block of code, but if one participant makes a small change in an existing file, the other authors are unlikely to notice. If the other authors then write their code assuming the prior coding was still in place, problems can easily emerge.

2. There is no clear mechanism for review built into the workflow. Edits occur silently, and immediately become part of the files used in a project.

I am aware of three strategies for avoiding these types of problems.

The first and most conservative solution to this is **full replication,** where each author conducts the full analysis independently and authors then compare results. If results match, authors can feel confident there are no problems in their code. But this strategy requires a massive duplication of effort – offsetting many of the benefits of co-authorship – and requires both authors be *able* to conduct the entire analysis, which is not always the case.

The second strategy is **compartmentalization**, in which each author is assigned responsibility for coding specific parts of the analysis. Author A, for example, may be responsible for importing, cleaning, and formatting data from an outside source while Author B is responsible for subsequent analysis. In this system, if Author B finds she need an additional variable for the analysis, she ask Author A to modify Author A's code rather than making modifications herself. This ensures responsibility for each block of code is clearly delimited, and changes are unlikely to sneak into an Author's code without their knowledge. In addition, authors can also then review one another's code prior to project finalization.[1][2]

The final strategy is to **use version control**, which is by far the most robust solution and the one most used by computer scientists, but also the one that requires the most upfront investment in learning a new skill.

"Version control" is the name for a piece of software specifically designed to manage collaboration on code (several exist, but `git` is by far the most well known and the only one I would recommend). Version control does several things. First, as the name implies, it keeps track of *every*

[1] Note that the separation of responsibility does not need to be as crude as "cleaning" and "analysis" – this strategy simply requires that a single person has clear and sole responsibility for every line of code in the project.

[2] Another intermediate strategy – which can be combined with compartmentalization – is to **maintain a change log** where authors record the date, files, and line-numbers of any changes they make. This eliminates the problem of edits going unnoticed. However, it is worth noting that this strategy only works if both authors are sufficiently diligent. If either (a) the author making changes fails to log all changes or does not describe them well, or (b) the reviewing author fails to go back into the code to check all the changes reported in the change log, the system may still fail.

version of your code that has ever existed and makes it easy to go back to old versions. This service is often provided by services like Dropbox, it is much easier to review old versions and identifying differences between old and new versions in `git` than through a service like Dropbox, whose interface is sufficiently cumbersome and most of us never use it unless we accidentally delete an important file.

Figure 1: `git` Pull Request on GitHub

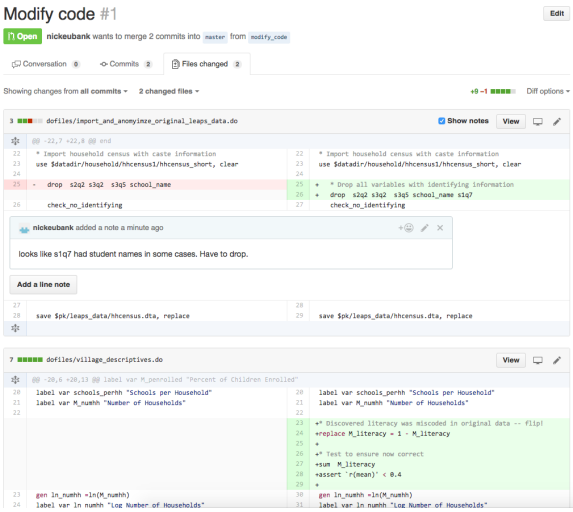


Figure 1 shows an example of a small proposed change to the code for a project on *GitHub*. Several aspects of the interface are worth noting. First, the interface displays *all* changes and the lines just above and below the changes across all documents in the project. This ensures no changes are overlooked. (Authors can click to "unfold" the code around a change if they need more context.) Second, the interface shows the prior contents of the project (on the left) and new content (on the right). In the upper pane, content has been changed, so old content is shown in red and new content in green. In the lower pane, new content has just been added, so simple grey space is shown on the left. Third, authors can easily comment (and discuss) individual lines of code, as shown here.

What really makes version control exceptional is that it makes it easy to (a) keep track of what differs between any two versions, and to (b) "propose" changes to code in a way that other authors can easily review before those changes are fully integrated. If Author A wants to modify code in version control, she first creates a "branch" – a kind of working version of the project. She then makes her changes on that branch and propose the branch be re-integrated into the main code. Version control is then able to present this proposed change in a very clear way, highlighting every change that the new branch would make to the code base to ensure no changes – no matter how small – go unnoticed. The author that made the proposed changes can then ask their co-author to review the changes before they are integrated into the code base. To illustrate, Figure 1 shows an example of what a simple proposed change to code looks like on *GitHub*, a popular site for managing `git` projects online.

Version control is an incredible tool, but it must be noted that it is not very user friendly. For those interested in making the jump, the tool to learn is `git`, and you can find a terrific set of tutorials from Atlassian (N.d.) , a nice (free, online) book on git (Chacon and Straub 2014), and a very nice, longer discussion of `git` for political scientists on *The Political Methodologist* (Jones 2013).

In addition, there are also two projects that attempt to smooth out the rough edges of the `git` user-interface. Github Desktop (Github 2016), for example, offers a Graphical User Interface and streamlines how `git` works. Similarly, git-legit (Reitz 2013) mimics the changes Github Desktop has made to how `git` works, but in the form of a command-line interface. These services are fully compatible with normal `git`, but learning one of these versions has the downside of not learning the industry-standard `git` interface. For researchers who don't plan to engage in contributing to open-source software or get a job in industry, however, that's probably not a huge loss.

3. Third-Party Code Review by Journals

As the discipline increasingly embraces in-house review of code prior to publication, one might wonder whether this is necessary. I am a strong advocate of Third-Party Review, but I think it is important to understand its limitations.

First, journals that conduct systematic review of replication code like *QJPS* and more recently the *AJPS* can only conduct the most basic of reviews. At the *QJPS*, in-house review only consists of ensuring the code is well-documented, that it runs without errors, and that the output it generates matches the results in the paper being published. Journals simply do not have the resources to check code line-by-line for correctness.

Second, even if Third-Party Review protects the integrity of the discipline, it does nothing to protect individual researchers. Appropriately or not, we expect researcher's code to be error free, and when errors are found, the career implications can be tremendous. Indeed, it is for this reason that I think we have an obligation to teach defensive programming skills to our students.

Finally, even detailed Third-Party Review is not foolproof. Indeed, the reason writing tests has become popular in computer science is a recognition of the fact that people aren't built to stare at code and think about all possible issues that might arise. Even in computer science, Third-Party Review of code focuses on whether code passes comprehensive suites of tests.

4. Responding to Errors

For all the reasons detailed here, I think it makes sense for the discipline to think more carefully about how we respond to errors discovered in the code underlying published papers. The status quo in the discipline is, I think most people would agree, to assume that most code both is and *should be* error-free. When errors are discovered, therefore, the result is often severe professional sanction.

But is this appropriate? The people who work most with code (computer scientists) have long ago moved away from the expectation that code can be error-free. At the same time, however, we cannot simply say "errors are ok." The middle road, I believe, lies in recognizing that not all errors are the same, and that we must tailor our responses to the nature of the coding error. Errors caused by gross negligence are obviously unacceptable, but I feel we should be more understanding of authors who write careful code but nevertheless make mistakes.

To be more specific, I think that as a discipline we should try to coordinate on a set of coding practices we deem appropriate. Then if an error is uncovered in the work of someone who has followed these practices – adding tests, commenting their code, using good naming conventions, no duplicating information, etc. – then we should recognize that they took their programming seriously and that to err is human, even in programming. Errors should not be ignored, but in these settings I feel it is more appropriate to respond to them in the same way we respond to an error in logical argumentation, rather than as an indicator of sloppiness or carelessness.

To be clear, this is not to say we should persecute authors who do not follow these practices. Someday – when these practices are being consistently taught to students – I think it will be reasonable to respond to errors differentially depending on whether the author was employing error-minimizing precautions. But the onus is on us – the instructors and advisors of rising to scholars – to ensure students are appropriately armed with these tools if we wish to later hold them responsible for programming mistakes. To do otherwise is simply unfair.

But what we can do today is agree to be especially understanding of scholars who work hard to ensure the integrity of their code who nevertheless make mistakes, now and in the future. This, I feel, is not only normatively appropriate, but also creates a positive incentive for the adoption of good programming practices.

5. Other Resources

This document includes just a handful of practices that I think can be of use to social scientists. I'm sure there are many more I am unaware of, and I encourage readers who are aware of useful practices to send them my way.[3] Here are some starting references:

- Code and Data for Social Scientists handbook written by Matthew Gentzkow and Jesse M. Shapiro (2014)
- *The Political Methodologist* issue on Work-Flow Management (Bowers 2011)

Acknowledgments

The author is grateful to Adriane Fresh, Simon Ejdemyr, Darin Christensen, Dorothy Kronick, Julia Payson, David Hausman, and Justin Esarey for their comments and contributions to this piece.

References

Atlassian. N.d. "What is Version Control." `https://www.atlassian.com/git/tutorials/what-is-version-control` (August 1, 2016).

Bowers, Jake. 2011. "Six Steps to a Better Relationship with Your Future Self." *The Political Methodologist* 18(2): 2-8.

Chacon, Scott, and Ben Straub. 2014. *Pro Git: Everything You Need to Know about Git.* 2nd ed. New York, NY: Apress. `https://git-scm.com/book/en/v2` (August 1, 2016.)

Eubank, Nicholas. 2014. "A Decade of Replications: Lessons from the Quarterly Journal of Political Science." *The Political Methodologist* 22(1): 18-21.

Eubank, Nicholas. 2016. "Lessons from a Decade of Replications at the Quarterly Journal of Political Science." *PS: Political Science & Politics* 49(2): 273-6.

Ejdemyr, Simon. 2015. "Tables in R (And How to Export Them to Word)." October. `http://stanford.edu/~ejdemyr/r-tutorials/tables-in-r/` (August 1, 2016).

Gentzkow, Matthew, and Jesse M. Shapiro. 2014. "Code and Data for the Social Sciences: A Practitioner's Guide." March 10. `http://web.stanford.edu/~gentzkow/research/CodeAndData.pdf` (August 1, 2016).

Github, Inc. 2016. "GitHub Desktop." `https://desktop.github.com/` (August 1, 2016).

[3] Users who google "defensive programming" will find many resources, but be aware many may not seem immediately applicable. Most defensive programming resources are written for computer scientists who are interested in writing applications to be distributed to users. Thus much of what is written is about how coders should "never trust the user to do what you expect." There's a clear analogy to "never assume your data looks like what you expect," but nevertheless mapping the lessons in those documents to data analysis applications can be tricky.

Hill, Michael S. 2015. "In Stata Coding, Style is the Essential: A Brief Commentary on Do-file Style." July 31. https://michaelshill.net/2015/07/31/in-stata-coding-style-is-the-essential/ (August 1, 2016.)

Jones, Zachary M. "Git/GitHub, Transparency, and Legitimacy in Quantitative Research." *The Political Methodologist* 21(1): 6-7.

Reitz, Kenneth. 2013. "Legit." http://www.git-legit.org/ (August 1, 2016).

Wickham, Hadley. N.d. "Style Guide." *Advanced R by Hadley Wickham.* http://adv-r.had.co.nz/Style.html (August 1, 2016).

Shiny App: Course Workload Estimator

Justin Esarey
Rice University
justin@justinesarey.com

I recently had the opportunity to dip my toe in developing web applications using Shiny through R (Winston et al. 2016). I was first introduced to the idea by one of my former graduate students, Jane Sumner (now starting as an Assistant Professor in political methodology at the University of Minnesota). Jane developed a tool to analyze the gender balance of assigned readings (Sumner 2016) using Shiny that later went viral (at least among the nerds I run with).

Elizabeth Barre (of Rice University's Center for Teaching Excellence) had talked to me about creating an application that allows both teachers and students to figure out exactly how much out-of-class work is being expected of them. A web application would enable visitors to input information from a syllabus and obtain a workload estimate based on pedagogical research about reading and writing speeds. She did the research and much of the graphic design of the app; I wrote the back-end code, set up a hosting server, and chipped in a little on the user interface design.

The result of our efforts is available at http://cte.rice.edu/workload (Barre and Esarey 2016); clicking the link will open a new window containing the app and extensive details about how we calculate our workload estimates. Unfortunately, wordpress.com won't let me directly embed the app on this page (although generally Shiny apps can be embedded via iframes). However, you can see what it looks like in Figure 1.

The tool is interesting on its own because it illustrates just how easy it is to expect too much of our students, even our graduate students, in a three or four credit hour course (see an extended discussion of these issues on the CTE's blog (Barre 2016)). I found that a normal graduate student syllabus can easily contain 20+ hours of out-of-class work, a substantial burden for a student taking three courses and working on his/her own research. But it was also interesting as an experiment in using R skills for web developing.

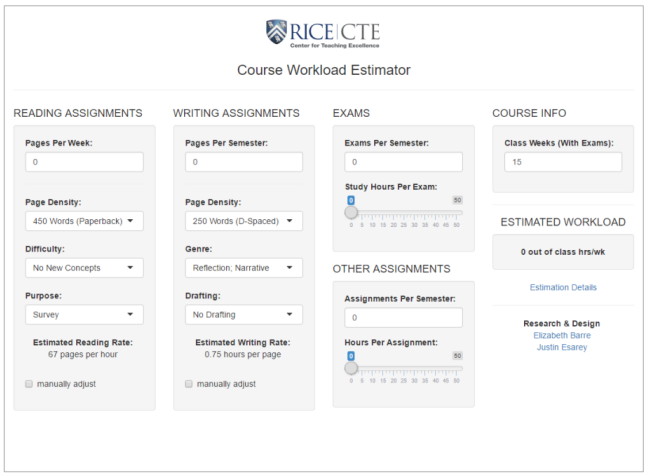


Figure 1: The Course Workload Estimator Tool, available at http://cte.rice.edu/workload.

The Shiny website provides a great tutorial that can help you get started, and going through that tutorial will probably be necessary even for an experienced R programmer just because Shiny's architecture is a little different. But I was still surprised how low the start-up costs were for me. It's especially easy if you're using RStudio as an IDE and hosting your application through Shiny's service, shinyapps.io. In just a few hours, I was able to create and publish a rough-draft but working web application for testing purposes.

Things got a little rougher when I decided that I wanted to save on bandwidth costs by building my own Shiny server in the cloud and hosting my apps there. Two tutorials posted by veteran users (Benton (2015) and Attali (2015)) helped me get started, but experience using Linux (and particularly Ubuntu) was helpful for me in understanding what was going on. Past experience became particularly relevant when I realized that the tutorials' recipes for setting up /etc/nginx/sites-available/default were causing my server to break and I had to go through line by line to figure out what was wrong. (Eventually, I was able to sequentially modify the default file until I had the features I wanted). Still, within 3 or 4 hours, I had my own server set up and hosted on DigitalOcean, with a domain at shiny.justinesarey.com pointing to the server and the course workload app running

smoothly.

In summary, I highly recommend checking out Shiny as a tool for teaching statistics and publicizing your research. It's incredibly easy to do if you're already experienced with programming in R and are hosting your apps through shinyapps.io.

References

Attali, Dean. 2015. "How to get your very own RStudio Server and Shiny Server with DigitalOcean." May 9. http://deanattali.com/2015/05/09/setup-rstudio-shiny-server-digital-ocean/ (August 1, 2016).

Barre, Elizabeth. 2016. "How Much Should We Assign? Estimating Out of Class Workload." July 11. http://cte.rice.edu/blogarchive/2016/07/11/workload (August 1, 2016).

Barre, Elizabeth, and Justin Esarey. 2016. "Course Workload Estimator." Center for Teaching Excellence, Rice University. http://cte.rice.edu/workload (Source code available at http://justinesarey.com/riceworkloadapp.zip) (August 1, 2016).

Benton, Morgan. 2015. "Deploying Your Very Own Shiny Server." December 9. https://qualityandinnovation.com/2015/12/09/deploying-your-very-own-shiny-server/ (August 1, 2016).

Sumner, Jane. 2016. "How Diverse is Your Syllabus?" https://jlsumner.shinyapps.io/syllabustool/ (August 1, 2016).

Winston, Chang, Joe Cheng, JJ Allaire, Yihui Xie, and Jonathan McPherson. 2016. *shiny: Web Application Framework for R.* R package version 0.13.2. https://CRAN.R-project.org/package=shiny (August 1, 2016).

Review of Political Analysis using R

Thomas J. Leeper
The London School of Economics and Political Science
thosjleeper@gmail.com

James E. Monogan, III. 2015. *Political Analysis Using R.* Springer.

There are a lot of books about R. A partial list on The R Project's website lists 157 as of May 2016 and that list has some glaring omissions such as Thomas Lumley's *Complex Surveys* (2010) and Hadley Wickham's (in press) *R for Data Science* (Grolemund and Wickham N.d.). Jamie Monogan gives us a new addition to this long list in the form of *Political Analysis Using R*. Even in a crowded field Monogan's text - hereafter *PAUR* - is a welcome addition and one that will fit nicely into a political science course on quantitative methods.

Contents

PAUR offers 11 chapters beginning with a novice, illustrated introduction to R and ending with a relatively high level discussion of R programming. Each chapter contains clearly highlighted example code, reproductions of R console output, numerous tables and figures, and a set of practice problems to test knowledge of content covered in the chapter.

Chapter 1 offers the very basics: how to download and install the software, and how to install add-on packages and use the R graphical user interfaces across Windows, Mac OS, and Linux. Chapter 2 covers data import (and export), along with basic manipulations, merging, and recoding.

Chapter 3 introduces R's base graphics functionality, covering histograms, bar charts, boxplots, scatterplots, and line graphs. It then offers a quick overview of lattice graphics (Sarkar 2008) to implement these and other visualizations. A notable absence from the chapter (and one noted by the author) is the increasingly popular ggplot2 package (Wickham and Chang 2015). The choice to rely exclusively on base graphics walks a difficult line, favoring the underlying strengths and limitations of the core library over those of add-on packages. Instructors using *PAUR* but wishing to teach ggplot2 will have to look elsewhere for relevant coverage of this material, perhaps to Wickham's ggplot2 book (Wickham 2013), now in its second edition. Chapter 4 covers familiar territory of descriptive statistics, including central tendency and dispersion. I appreciated the way *PAUR* covered these topics, presenting formulae, code, and graphical depictions of distributions close to one another.

As is a consistent theme throughout the text, *PAUR* presents practical R implementations of statistical problems as part of larger substantive discussion of real political science examples. Indeed, one of *PAUR*'s key strengths for a political science audience is its reliance on a familiar set of datasets from real political science applications. Leveraging and role modelling good open science practices, *PAUR* provides a Dataverse with complete data and code for all examples, which are in turn drawn from publicly available data and code used in published research articles. This should make it extremely easy for instructors to use *PAUR* in a quantitative methods sequence, by closely linking formal coverage of techniques, the substantive application of those techniques in political science articles, and implementation of those techniques in R. *PAUR* means there is little

excuse to continue to use *iris* to teach scatterplots or *mtcars* to teach linear regression.

Chapter 5 offers basic statistical hypothesis testing, as well as other techniques of bivariate association (e.g., cross-tabulation). This chapter uses the gmodels package (Warnes et al. 2015) to provide cross-tabulations, which is a somewhat unfortunate reminder of R's weaknesses in basic cross-tabulation, but a good decision from the perspective of teaching tabulation to those new to statistics or coming to the language from Stata or SPSS. This chapter probably could have taken a different route and used R to teach the logic of statistical significance (e.g., through simulations), but instead focuses mainly on how to implement specific procedures (t-test, correlation coefficient, etc.).

Chapter 6 marks a rapid acceleration in the breadth and density of content offered by *PAUR*. While the first 5 chapters provide a first course in statistical analysis in R, the second half of the book quickly addresses a large number of approaches that may or may not fit in that setting. Chapter 6 covers OLS and Chapter 7 covers logit and probit models, ordinal outcome models, and count models. (By comparison, John Verzani's (2014) *Using R for Introductory Statistics* ends with half of a chapter on logistic regression; John Fox and Sanford Weisberg's (2011) *An R Companion to Applied Regression* covers GLMs over two chapters as a final advanced topic.)

This transition from an elementary textbook on statistics to a sophisticated introduction to the most commonly used methods in political science is a strength and challenge for *PAUR*. On the one hand it greatly expands the usefulness of the text beyond an undergraduate companion text to something that could reasonably fit in a masters-level or even PhD methods sequence. On the other, it means that some instructors may find it difficult to cover all of the topics in the text during a 15-week semester (and certainly not in a 10-week quarter). That said, the text covers many of the topics that were addressed in the "grab bag" 1st year methods course I took in graduate school and would have been an immensely helpful companion as I first trudged through linear algebra, maximum likelihood estimation, and time-series in R.

To highlight some of the content covered here, Chapter 6 addresses linear regression and does a good job of leveraging add-on packages to introduce model output (with xtable (Dahl 2016)), model diagnostics (with lmtest (Zeileis and Hothorn 2002) and car (Fox and Weisberg 2011), and heteroskedasticity (with sandwich (Zeileis 2004)). Chapter 7 turns to generalized linear models using examples from the Comparative Study of Electoral Systems.

Chapter 8 is a real gem. Here Monogan made the right choice to enlist an army of excellent packages to teach advanced topics not commonly covered in competing textbooks: lme4 (Bates et al. 2015) to teach mixed effects or multi-level models, along with some of political scientists' contributions to R in the form of MCMCpack (Martin, Quinn, and Park 2011) to teach bayesian regression, cem (Iacus, King, and Porro 2015) to showcase matching methods, and wnominate (Poole et al. 2011) to teach roll call analysis. These are topics and packages that would be unusual to see in other introductions to R or other statistical texts, which clearly shows Monogan's intention in *PAUR* to provide a textbook for up-to-date *political* analysis.

Chapter 9 covers time series methods. I am always a bit ambivalent about teaching these in a general course, but the chapter presents the methods clearly so the key aspects are there for those who want to include them. Chapter 10 and 11 serve as a high-level capstone with coverage of matrix operations, basic programming (functions, loops, conditional expressions, etc.), optimization, and simulation. Again, as with everything in the latter third of the book, these elements make *PAUR* stand out among competitors as a text that is particularly appropriate for teaching methods of quantitative political science as it is currently practiced.

Overall Evaluation

PAUR is not a reference manual nor a book about R as a programming language. It is, as its title clearly states, a guidebook to practicing quantitative political science. It is the kind of text that will make it easier to teach postgraduate students how to use R, as well as provide a relevant companion text to an intermediate or advanced course in quantitative methods to be taught at other levels.

I suspect political scientists coming to R from Stata would also find the text particularly attractive given its coverage of nearly all statistical techniques in wide use in the discipline today and its reliance on familiar disciplinary examples. It rightly does not attempt to rival say Cameron and Trivedi's (2010) *Microeconometrics Using Stata* in scope, but adopts a focus on more cutting edge techniques at the expense of minutiae about older methods.

I applaud Monogan for *Political Analysis Using R*, for the ambition to provide a broadly relevant and useful new text on R, and for showcasing the value added of data sharing and reproducible research as a model of learning and teaching quantitative research. And I only dock him a few points for leaving out ggplot2. Well done.

References

Bates, Douglas, Martin Maechler, Ben Bolker, and Steve Walker. 2015. "Fitting Linear Mixed-Effects Models Using lme4." *Journal of Statistical Software* 67(1): 1-48. `http://dx.doi.org/10.18637/jss.v067.i01` (August 1, 2016).

Cameron, A. Colin, and Pravin K. Trivedi. 2010. *Microeconomics Using Stata*. Revised edition. College Station, TX: A Stata Press Publication.

Dahl, David B. 2016. *xtable: Export Tables to LaTeX or HTML.* R package version 1.8-2. `https://CRAN.R-project.org/package=xtable` (August 1, 2016).

Fox, John, and Sanford Weisberg. 2011. *An R Companion to Applied Regression.* 2nd ed. Thousand Oaks, CA: SAGE Publications, Inc. `http://socserv.socsci.mcmaster.ca/jfox/Books/Companion/` (August 1, 2016).

Grolemund, Garrett, and Hadley Wickham. N.d. *R for Data Science: Visualize, Model, Transform, Tidy, and Import Data.* Sebastopol, CA: O'Reilly Media, Inc. Forthcoming.

Lumley, Thomas. 2010. *Complex Surveys: A Guide to Analysis Using R.* Hoboken, NJ: John Wiley & Sons, Inc.

Iacus, Stefano M., Gary King, and Giuseppe Porro. 2015. *cem: Coarsened Exact Matching.* R package version 1.1.17. `https://CRAN.R-project.org/package=cem` (August 1, 2016).

Martin, Andrew D., Kevin M. Quinn, and Jong Hee Park. 2011. "MCMCpack: Markov Chain Monte Carlo in R." *Journal of Statistical Software.* 42(9): 1-21. `http://www.jstatsoft.org/v42/i09/` (August 1, 2016).

Monogan III, James E. 2015. *Political Analysis Using R.* Cham, Switzerland: Springer International Publishing Switzerland.

Monogan III, James E. 2015. "Political Analysis Using R: Example Code and Data, Plus Data for Practice Problems." `http://dx.doi.org/10.7910/DVN/ARKOTI` Harvard Dataverse, V1 [UNF:6:+itU9hcUJ8I9E0Kqv8HWHg==] (August 1, 2016).

Poole, Keith, Jeffrey Lewis, James Lo, and Royce Carroll. 2011. "Scaling Roll Call Votes with wnominate in R." *Journal of Statistical Software* 42(14): 1-21. `http://www.jstatsoft.org/v42/i14/` (August 1, 2016).

Sarkar, Deepayan. 2008. *Lattice: Multivariate Data Visualization with R.* New York: Springer.

Verzani, John. 2014. *Using R for Introductory Statistics.* 2nd ed. Boca Raton, FL: CRC Press.

Warnes, Gregory R., Ben Bolker, Thomas Lumley, Randall C Johnson. Contributions from Randall C. Johnson are Copyright SAIC-Frederick, Inc. Funded by the Intramural Research Program, of the NIH, National Cancer Institute and Center for Cancer Research under NCI Contract NO1-CO-12400. 2015. *gmodels: Various R Programming Tools for Model Fitting.* R package version 2.16.2. `https://CRAN.R-project.org/package=gmodels` (August 1, 2016).

Wickham, Hadley. 2013. "ggplot2." `http://had.co.nz/ggplot2` (August 1, 2016).

Wickham, Hadley, and Winston Chang. 2015. *ggplot2: An implementation of the Grammar of Graphics.* R package version 1.0.1. `https://cran.r-project.org/package=ggplot2` (August 1, 2016).

Zeileis, Achim. 2004. "Econometric Computing with HC and HAC Covariance Matrix Estimators." *Journal of Statistical Software* 11(10): 1-17. `http://www.jstatsoft.org/v11/i10/` (August 1, 2016).

Zeileis, Achim, and Torsten Hothorn. 2002. *Diagnostic Checking in Regression Relationships.* R News 2(3), 7-10. `http://CRAN.R-project.org/doc/Rnews/` (August 1, 2016).

Rice University
Department of Political Science
Herzstein Hall
6100 Main Street (P.O. Box 1892)
Houston, Texas 77251-1892

The Political Methodologist is the newsletter of the Political Methodology Section of the American Political Science Association. Copyright 2016, American Political Science Association. All rights reserved. The support of the Department of Political Science at Rice University in helping to defray the editorial and production costs of the newsletter is gratefully acknowledged.

Subscriptions to *TPM* are free for members of the APSA's Methodology Section. Please contact APSA (202-483-2512) if you are interested in joining the section. Dues are $29.00 per year for students and $44.00 per year for other members. They include a free subscription to *Political Analysis*, the quarterly journal of the section.

Submissions to *TPM* are always welcome. Articles may be sent to any of the editors, by e-mail if possible. Alternatively, submissions can be made on diskette as plain ASCII files sent to Justin Esarey, 108 Herzstein Hall, 6100 Main Street (P.O. Box 1892), Houston, TX 77251-1892. LaTeX format files are especially encouraged.

TPM was produced using LaTeX.

THE SOCIETY FOR
POLITICAL METHODOLOGY
ad indicia spectate

Chair: Jeffrey Lewis
University of California, Los Angeles
jblewis@polisci.ucla.edu

Chair Elect and Vice Chair: Kosuke Imai
Princeton University
kimai@princeton.edu

Treasurer: Luke Keele
Pennsylvania State University
ljk20@psu.edu

Member at-Large : Lonna Atkeson
University of New Mexico
atkeson@unm.edu

Political Analysis Editors:
R. Michael Alvarez and Jonathan N. Katz
California Institute of Technology
rma@hss.caltech.edu and *jkatz@caltech.edu*