The Political Methodologist

Newsletter of the Political Methodology Section
American Political Science Association
Volume 14, Number 1, Spring 2006

Editors:
Adam J. Berinsky, Massachusetts Institute of Technology
berinsky@mit.edu
Michael C. Herron, Dartmouth College
Michael.C.Herron@dartmouth.edu
Jeffrey B. Lewis, University of California Los Angeles
jblewis@ucla.edu

Editorial Assistant:
Seth J. Hill, University of California Los Angeles
sjhill@ucla.edu

Contents

Notes from the Editors **1**

Articles **2**

Philip A. Schrodt: Twenty Years of the Kansas Event Data System Project 2

Computing and Software **6**

William G. Jacoby: The Dot Plot: A Graphical Display for Labeled Quantitative Values . . . 6

Jasjeet S. Sekhon: The Art of Benchmarking: Evaluating the Performance of R on Linux and OS X 15

The LaTeXCorner **19**

Ian Yohai: PowerPoint™ for LaTeX: The beamer Class 19

Book Reviews **21**

Joshua D. Clinton: Review of Keith T. Poole' *Spatial Models of Parliamentary Voting* 21

Section Activities **24**

A note from our Section President 24

Announcements **25**

Association of Religion Data Archives 25

Notes From the Editors

Welcome again to *The Political Methodologist*. Our issue leads off with an article by Phil Schrodt describing the 20-year evolution of the Kansas Event Data System Project (KEDS). We then move to two articles in the computing section. William Jacoby makes the case that political scientists should present their data with dot plots and provides R code to produce these graphics. Jasjeet Sekhon then describes a general methodology for benchmarking software and presents an application, evaluating the relative performance of R on Linux and OS X. Next, in the LaTeXcorner, Ian Yohai describes BEAMER, a program that facilitates the preparation of PowerPoint-like presentations using LaTeX. We also have in this issue a review of Keith Poole's recent book on Spatial Models of Parliamentary Voting, another title in the Analytical Methods for Social Research series, published by Cambridge University Press. We plan to review additional titles from this series in future issues. We close with a note from our section President, Janet Box-Steffensmeier.

The next issue of TPM is beginning to take shape, but we are always on the lookout for more material. Your submissions and ideas for topics to address are most welcome.

The Editors

Articles

Twenty Years of the Kansas Event Data System Project

Philip A. Schrodt
University of Kansas
schrodt@ku.edu

In the beginning there was event data

Event data—sequences of nominal codes recording the interactions among political actors—have been a major focus of quantitative international relations (IR) research since the 1960s and 1970s.[1] Until recently most event data analysis used either Edward Azar's (1982) Conflict and Peace Data Bank (COPDAB) or Charles McClelland's (1976) World Event Interaction Survey (WEIS), but over the past decade the combination of machine-readable news reports and automated coding have dramatically reduced the costs of generating new data sets.

This article will discuss the development of the automated coding systems of the Kansas Event Data System (KEDS) project, which developed the first automated coding system—the eponymous KEDS computer program—that could produce data acceptable in refereed articles. This is a personal history rather than an attempt to provide a fully balanced account, and is told from my perspective with an emphasis on the developments that seemed most important to me; other recollections may differ.

Roots

The initial enthusiasm for event data among researchers was tempered by the fact that it was very expensive to produce. Historically, event data have been coded by legions of bored students flipping through copies of *The New York Times*. Event data collection slowed in the late 1970s as funding—which had largely come from the U.S. Department of Defense Advanced Research Projects Agency (DARPA)—was discontinued. The existing data sets continued to be widely used—one study found them to be the third most frequently used form of data in quantitative IR research—but they were not updated.

In the mid-1980s, I became involved in a fascinating set of work influenced by the then very trendy research on "artificial intelligence" (AI). A number of political scientists, dissatisfied with the limitations of conventional statistical analysis, attempted to apply AI to the formal study of international relations and foreign policy. This produced a variety of quite novel approaches using computational methods before being tragically destroyed after exposure to a particularly virulent strain of post-modern deconstructionism, for which at the time there was very little acquired immunity.[2] My own efforts at computational modeling largely revolved around applications of event data, in particular trying to use these to simulate the problem of reasoning by analogy (see `http://www.ku.edu/~keds/papers.dir/Schrodt.PRL.2.0.pdf`).

I used an assortment of event data sets in this work, mostly versions of COPDAB and WEIS that I had acquired from various sources. It was clear, however, that these were not very dense, particularly on the Arab-Israeli conflict, the area where I was also doing field research. More critically, they were not being updated, so I could not use them to study contemporary issues or do true forecasting.

In the process of doing some contract programming for a West Coast consulting firm developing a political decision-support system "for a major U.S. ally", I became acquainted with (and obtained a copy of) a WEIS data set that had been collected by various defense consulting firms for no less a client than the National Security Council in the Reagan White House, where event data was championed by a McClelland student named Richard Beale. This effort continued until Beale's untimely death in 1985.

One of the novel features of this version of WEIS was the inclusion of brief English-language summaries of the news story that generated the event. This provided an opportunity to check whether it was possible, in principle, to go from a natural language text to event codes. Working with a Northwestern University undergraduate, David Leibsohn, I developed a simple computer program that mapped keywords to event codes, and presented this at the 1985 International Studies Association meetings (Schrodt and Leibsohn 1985). This produced credible results, and in particular provided early evidence that while some WEIS categories were subtle and difficult to code consistently (probably for humans as well as machines), many of the most common

[1] The editors note with sadness that Deborah Gerner, Phil Schrodt's wife and collaborator on the KEDS project, died on June 19, 2006 after a long illness. We express out condolences to Phil.

[2] Sylvan and Chan (1984) provides early examples of the AI approach; Hudson (1991) has later, more sophisticated examples, and Trappl (2006) shows the approach is not dead yet.

events—notably meetings and uses of force—were straightforward because they were described using a very distinct and specific vocabulary.

In the late 1980's, the National Science Foundation undertook a major initiative titled "Data Development in International Relations" to update the most widely used international relations data sets (and, one suspects, reduce the grousing from the quantitative IR community over the resources going to the American National Election Survey). The second phase of DDIR was headed by my dissertation adviser Dina Zinnes and her colleague, the late Richard Merritt (Merritt, Muncaster and Zinnes 1994). A group of about twenty researchers was convened, and eventually NSF invested about $350,000 in a number of different event data projects.

Automated coding was seen as a possible, but by no means proven, approach to reducing the costs of producing event data. My initial contribution to DDIR was a machine learning program that was an elaboration of the Schrodt-Leibsohn work. It worked reasonably, though not spectacularly well, and its ultimate contribution was to simply provide some of the basic code for what would develop into KEDS. The machine learning aspect was consistent with most of my work in AI, but turned out to be a dead-end: KEDS and TABARI both eventually required extensive, and highly expert, dictionary development by humans. In retrospect, this simply reflected a general lesson from automated natural language processing in the 1980s—humans are so good at language, and language is such an idiosyncratic human construct, that it is better to let humans tell a machine what to do (and then have the machine routinely do it) than to try to develop machine learning algorithms.[3]

By the late 1980s, we had a reasonable set of techniques that provides credible results, but we had only shown that we could map natural language news *summaries* into event categories. This was a long way from the "holy grail" of producing data directly from news wire accounts. The closest thing available seemed to be various machine-readable indices, but these typically had insufficient information to resolve events beyond the level of the dozen or so major "cue categories" in the event coding schemes.

At this point I had a chance encounter with a student involved with the University of Kansas debate program who, on hearing about my research, asked "Why don't you just use Reuters?—it's available on NEXIS." "Where do I get NEXIS?" "At the Law School—all the debaters use it there."[4]

I arranged a meeting with one of the Law School librarians, who demonstrated that one could, in fact, download Reuters stories via a dial-up connection. He assured us that there was no marginal cost to the Law School for using the service, and then noted that the Law library was closed overnight. He then suggested we wait while he showed us a couple more things about the system, but he had to check with NEXIS technical support first. He placed a couple of calls, asking questions at the level of "Which one is the *any* key??" but in the process of getting authorization, repeated, loudly and slowly "Our NEXIS password is...".

Dial-up connection, library is closed, here's the password: we can take a hint. We set up a simple script to automate a log-in to NEXIS from about 2 a.m. to 5 a.m., and over the next several months downloaded tens of thousands of stories.

KEDS

Based on the WINR demonstration, DDIR provided a small $40,000 grant in 1991-1993 for the development of what became the KEDS program. While I continued to do most of the computer programming, the bulk of the value added from the KEDS project has been provided by the twenty or so "dictionary developers" who have been involved with the project and devoted thousands of hours to refining the dictionaries that are essential to producing data.[5] As I am endowed with the inter-personal skills typical of a computer programmer, this aspect of the project has been directed by my collaborator Deborah J. Gerner.

While the KEDS work for DDIR included some experimentation with German-language sources and foreign policy chronologies (Gerner et al 1994), most of our development focused on WEIS coding of interactions in the Middle East reported by the Reuters news service in English. We focused on this area both because it is very thoroughly covered in the international press, but also because we were doing field work in the area and could therefore cross-check the validity of event data based on our experience in the region.

KEDS had its professional debut at the 1992 International Studies Association meetings in Atlanta. The conference paper was being written, typically, at almost the last minute, and focused on a 12-year time series for the Arab-Israeli conflict. A number of different pieces had to come together to generate this—downloading and formatting the Reuters stories, on-going dictionary development, and aggregation of the resulting events into an interval-level time

[3]The contemporaneous DARPA sponsored "Message Understanding Conference" project (DARPA 1993) had somewhat similar objectives—extracting details of terrorist incidents from news wire stories (and involving real computer scientists and real linguists!)—and came to similar conclusions: systems using extensive phrase dictionaries developed by humans far out-performed machine-learning systems.

[4]Yes, kiddies, in those days NEXIS was a highly restricted resource, not something available on a browser at most research universities. We also walked to and from school every day in the snow. Barefoot. In June. Uphill. Both directions.

[5]An eloquent description of the challenges of dictionary development can be found in Joseph Pull's "Ode on Coding" `http://www.ku.edu/~keds/home.dir.ode.html` which Pull wrote prior to leaving the project for Yale Law School.

series using the Goldstein (1992) scale—so only when the paper was nearly finished that could I actually look at the results. I still vividly remember finally getting the Israel-Palestinian series, and plugging it into MS-EXCEL to get a basic plot. My great fear was that the Palestinian intifada would not show up in the data. To my tremendous relief, there it was as a lovely (if noisy) spike followed by an exponential decay, the most conspicuous feature of the series. On early-1990s hardware, the system coded about 70 events per second, a huge improvement over human coding projects, which typically have a sustained output of five to ten events per coder per hour.

One of the people who heard that ISA presentation was Doug Bond from the Program on Nonviolent Sanctions in Conflict and Defense at the Center for International Affairs at Harvard who was beginning the development of a new event coding scheme, the Protocol for the Assessment of Nonviolent Direct Action (PANDA). The PANDA project worked in close collaboration with us for the next two years during the most intense development of KEDS in dictionary development, identification of bugs, and validation.

The PANDA work eventually spun off a commercial event coding operation—VRA, Inc. (`http://vra.com`)—which developed a coding program that used quite different principles than KEDS. The PANDA coding system, with the added collaboration of Craig Jenkins (Ohio State) and Charles Taylor (VPI) morphed into the Integrated Data for Events Analysis (IDEA) coding scheme (Bond et al 1997). Reuters reports dealing with the entire world have been coded for by VRA for 1985-2004; the resulting data set contains about 10-million events and can be downloaded from `http://gking.harvard.edu/data.shtml`.

Tabari

KEDS was written in the PASCAL programming language and worked only on Apple Macintosh computers. The choice of PASCAL made sense at the time—it was the core language for the Macintosh operating system and my visceral loathing of Microsoft made the Macintosh the only option if I were to be doing the programming.[6] However, by the late 1990s PASCAL had been largely superceded by the C/C++ as the most common general-purpose programming language and compiler support for the language was dwindling. Furthermore, while KEDS was generally stable from 1995 to 2000, it contained some deep-seated idiosyncrasies that could only be eliminated by completely rewriting the program.

In response to this, TABARI —Textual Analysis By Augmented Replacement Instructions—was created in the spring of 2000. It is based on the same sparse-parsing principles as KEDS but is written as "open-source" code in ANSI C++ and was immediately ported to the LINUX and WINDOWS operating systems. The conversion to C++ resulted in a program that was substantially faster than the PASCAL code—the program codes about 8,500 records per second even on an inexpensive machine, a $500 1.2 Ghz G4 Mac Mini. This is about 300-times faster than KEDS, and about 33-million times faster than typical human coding. A simple keyboard-driven interface is implemented using the UNIX "ncurses" terminal library, and consequently we now have 100% compatibility between the Macintosh and UNIX/LINUX versions, as well as allowing the program to run remotely from a server.[7]

The most recent development in our project has been the CAMEO—Conflict and Management Event Observations—coding scheme. This is a new coding system specifically designed for automated coding, and has also evolved to accommodate the post-Cold War emphasis on political events involving sub-state actors. This work has been primarily done by Deborah Gerner and her graduate student Ömür Yilmaz.

In addition to TABARI and CAMEO, the KEDS project has produced an increasingly diverse set of utility programs to support the production of event data. The most important of these have been our automated downloading programs, which have evolved from a script running a dial-in connection followed by processing in PASCAL to an integrated PERL program that does downloading and reformatting from HTML files taken off the web. ACTOR_FILTER is another one of our stalwarts: this identifies the actors in a set of text based on capitalization patterns, and produces a keyword-in-context index of these, sorted by frequency.

Funding

The KEDS and TABARI systems—both the programming and the more labor intensive dictionary development—have been funded by a combination of NSF grants and government contracts, with occasional bridge funding from KU, and an interesting contract doing conflict monitoring for a Swiss-based NGO. We've been lucky (okay, we've also made our luck. . .): money has always been available for the tasks we needed to do, and in fact at times we've turned down work because of our limited number of trained coders [take the hint: if you can master the relevant tools, there is more funding available for this than we can handle].

We've always kept our work unclassified, for both principled and pragmatic reasons. We've no desire to become the Kansas equivalent of Los Alamos scientist Wen Ho

[6] Linux was still a gleam in Linus Torvald's eye when work began on KEDS, and machines running various flavors of UNIX were quite expensive. This is Kansas: we don't do expensive.

[7] We've had less success finding someone to keep the WINDOWS version current. Whatever. . .

[8] Trust us, any FBI agent based in Topeka will want to get out.

Lee and advance the career of a zealous FBI agent anxious to get out of Topeka.[8] Meanwhile the value-added of classified material in the real world doesn't quite live up to its portrayal in the movies: consider for example the timely warnings provided by classified analysis on the collapse of the Soviet Union, India nuclear weapons tests, Iraqi WMDs and the recent Hamas electoral victory. As the "open source" concept was popularized in the late 1990's, we shifted all of our work to that mode.

The KEDS project has generally been a relatively small affair: typically Gerner and me, one or two graduate assistants, a data manager, and a half-dozen coders. We've been larger—last summer PRI's accountant came to me and said "Do you realize you've got twenty people on your payroll??" (uh, no, I hadn't—kinda creeps up on you. . .)—but smaller is the norm. I occasionally get emails from people wanting to come and visit "our shop"—presumably envisioning the vast KEDS Building with its own cafeteria, weight room and day-care center. I explain that there really isn't a shop, just a web page: nothing to see here, move along, move along.

Mama don't your babies grow up to be event data analysts

At this point we have spent five years in initial experimentation with automated coding methods, devoted about fifteen years to operational program and dictionary development, produced regional data sets for about thirty countries, and now have the capability of maintaining data sets with a resolution of about a day at close to zero marginal cost (or at least a lower marginal cost than any other known method of creating data in the social sciences, including curb-stoning). Event data analysis has therefore taken the quantitative international relations world by storm, right?

Well, no. While articles utilizing event data have appeared on a relatively regular basis in all of the refereed "sacred journals" that carry quantitative work, it remains very much a niche approach in international relations and comparative politics. Individuals focusing on event data analysis have, with a couple of exceptions, not fared particularly well in the academic job market: in fact the individual who I feel was doing some of the very best work outside of KU was, at last report, running a coffee shop.

Event data has fared substantially better in the policy community, and several people who have been unable to secure academic employment have gone on to positions as quantitative policy analysts in the defense and intelligence communities. There they pull down salaries twice those of academics, don't have to attend faculty meetings, don't grade bluebooks, and can't take their work home because it is classified.

This latter point, however, means that we have had very little feedback from the policy community. Six months after one of my best-trained students took a defense-related job where he was hired explicitly for his event data training, we met at the APSA. "Job going well?" says I. "Yep." says he. "Bet you can't tell me a single thing about it." says I. "Yep." says he.[9]

Two things stand in the way of this. The first is paradigmatic: quantitative research in international relations is dominated by the "Correlates of War" approach that has almost nothing in common with event data analysis. COW studies typically involve the analysis of interval-level variables measured at the nation-state dyad-year level across two centuries and the entire international system. In contrast, contemporary event data analysis focuses nominal measurements in protracted conflicts across a couple of decades or less, but with daily resolution and an increasing focus on sub-state actors. The COW community has generally focused on retrospective inference guided by theoretical issues; the event data community on policy-relevant forecasting.

The second problem involves the shortage of nominal-level time series methods. These exist—for example hidden Markov models—but they are generally closer to pattern recognition methods than to classical frequentist statistics. Interval-level time series are used extensively in econometrics, a field already familiar to most political methodologist. In contrast, the two major sources of nominal-level methods are the very unfamiliar fields of linguistics and bioinfomatics.

Central to the Japanese "manufacturing miracle" of the second half of the twentieth century was the concept of *kaizen*—incremental improvement. A worker's suggestion that increases the quality of a product by only 0.1% will, when combined with similar suggestions by thousands of workers over a period of decades, provide the technological leverage to reduce a device for playing music from the size of a suitcase to the size of a pocket knife, while hugely increasing capacity and quality.

KEDS was an improvement over human coding. TABARI and the VRA CODER are improvements over KEDS, and TABARI can be incrementally augmented through the open-source development process. The CAMEO and IDEA coding schemes are improvements over WEIS and COPDAB. Each time a coder finds another verb phrase to add to the dictionary, or adds another name to the list of actors being coded, the probability of sentences being coded correctly increases, however slightly.

At this point we probably have a good idea of how to produce event data—all event data articles published in major journals over the past ten years have used machine-

[9]This "I could tell you but then I'd have to kill you" problem has also affected forecasting models using rational choice methods. These may, or may not, be extensively used in the intelligence community, depending on who you want to believe.

coded data, and the last major human coding project was shut down in 2004 following a comparison between its data and a comparable data set produced using TABARI. We still need to take the next step in figuring out some really good things to do with it. But at least we've started.

For further information:

The KEDS project maintains a very extensive web site at `http://www.ku.edu/~keds`. At this site you will find the most recent versions of the software and documentation, assorted coding dictionaries, data sets and utility programs, a FAQ (frequently-asked-questions) section, and copies of papers from our project and related efforts.

References

Azar, Edward E. 1980. "The Conflict and Peace Data Bank (COPDAB) Project." *Journal of Conflict Resolution* 24:143-152.

Bond, Doug, J. Craig Jenkins, Charles L. Taylor and Kurt Schock. 1997. Mapping Mass Political Conflict and Civil Society: The Automated Development of Event Data. *Journal of Conflict Resolution* 41, 4: 553-579.

Defense Advanced Research Projects Agency. 1993. *Proceedings of the Fifth Message Understanding Conference (MUC-5).* Los Altos,CA: Morgan Kaufmann.

Gerner, Deborah J., Philip A. Schrodt, Ronald A. Francisco, and Judith L. Weddle. 1994. "The Machine Coding of Events from Regional and International Sources," *International Studies Quarterly* 38:91-119.

Goldstein, Joshua S. 1992. "A Conflict-Cooperation Scale for WEIS Events Data." *Journal of Conflict Resolution* 36: 369-385.

Hudson, Valerie, ed. 1991. *Artificial Intelligence and International Politics.* Boulder: Westview

McClelland, Charles A. 1976. *World Event/Interaction Survey Codebook.* (ICPSR 5211). Ann Arbor: Inter-University Consortium for Political and Social Research.

Merritt, Richard L., Robert G. Muncaster and Dina A. Zinnes, eds. 1993. *International Event Data Developments: DDIR Phase II.* Ann Arbor: University of Michigan Press.

Schrodt, Philip A. and David Leibsohn. 1985. "An Algorithm for the Classification of WEIS Events from WEIS Textual Data." Paper presented at the International Studies Association, Washington, March 1985.

Schrodt, Philip A. and Deborah J. Gerner. 1994. "Validity assessment of a machine-coded event data set for the Middle East, 1982-1992." *American Journal of Political Science* 38: 825-854.

Trappl, Robert, ed. 2006. *Programming for Peace : Computer-Aided Methods for International Conflict Resolution and Prevention.* Berlin: Springer.

Computing and Software

The Dot Plot: A Graphical Display for Labeled Quantitative Values

William G. Jacoby
Michigan StateUniversity
jacoby@msu.edu

The dot plot is an extremely useful tool for obtaining pictorial representations of quantitative information.[1] This display method is very flexible and potentially applicable to any situation where numeric values are associated with descriptive labels. For example, dot plots can be used to depict raw data, frequency counts, descriptive statistics, and parameter estimates from statistical models. A carefully constructed dot plot contains an enormous amount of in-

[1]I would like to thank David Armstrong, Michael Colaresi, and Saundra Schneider for their excellent comments and suggestions on earlier drafts of this paper.

formation. More important, a dot plot can convey that information in a way that overcomes some of the problems frequently encountered with other graphical displays.

Definition and Examples

A dot plot is a two-dimensional graphical display of objects, showing some quantitative characteristic of those objects. One axis of the dot plot (usually the horizontal) is a scale covering the range of quantitative values to be plotted. The other axis (usually the vertical) shows descriptive labels that are associated with each of the numeric values. The data objects usually are sorted according to the quantitative values. Plotting symbols are placed within the display area of the dot plot, locating each data object at the intersection position for its label on the vertical axis and associated numeric value on the horizontal axis. While this simple definition covers the basic features of the dot plot, it is important to emphasize that the utility and flexibility of such a display come from the details that are included in its construction. Let us consider several examples that will illustrate the dot plot's various features and advantages.

The simplest application of the dot plot is to display the empirical distribution of values on a single variable. Figure 1 shows a dot plot of policy priority scores for the American states in 1992. This is an interval-level variable obtained from an unfolding analysis of the states' proportionate spending levels across fifteen program areas (Jacoby and Schneider 2001). Larger values on this variable indicate that a state spent more on a set of policies that Jacoby and Schneider labeled "collective goods" such as highways, parks, and law enforcement. Smaller values correspond to more spending on "particularized benefits," including welfare, health care, and employment security.

Figure 1 is very easy to interpret. The labels in the margin of the vertical axis are the state names. The spending priority score for each state can be determined from the horizontal position of the plotted point within that row: The farther to the right, the larger the data value (i.e., higher spending on collective goods); the farther to the left, the lower the data value (i.e., more spending on particularized benefits).

Note the efficiency with which the information is presented in Figure 1. The horizontal dotted lines facilitate table look-up without being too intrusive on the data points. And, because the observations are sorted by the data values, the graph is, effectively, a transposed quantile plot. Therefore, the display provides information about the shape of the distribution. It is also very easy to obtain visual estimates of the "important" quantiles, such as the median (the value that occurs at the midpoint along the vertical axis), the quartiles (the values that occur one-fourth of the way in from the top and bottom of the graph), and the extremes (the first and last values along the vertical axis). Thus, a dot plot enables the analyst to see both "the forest" (i.e., the distribution) and "the trees" (i.e., the individual observations).

Figure 2 illustrates a variation of the basic dot plot, showing state education spending for fiscal year 2000, in dollars per capita. This display provides exactly the same kind of information as Figure 1. But here, the varying-length horizontal lines emphasize the shape of the point array (and, hence, of the distribution, itself) more clearly. Note also that the dashed lines in Figure 2 all emanate from the zero point on the horizontal axis. Therefore, the line lengths for different states can be compared to facilitate magnitude judgments about the data values.

The basic dot plot data display can be adapted very easily to allow comparisons across subgroups of observations. For example, Figure 3 is a *divided dot plot*, showing individual state policy priority scores within separate regions. Again, a great deal of information can be extracted from this display. The data values are sorted within the regions. The order of the regions, themselves, within the plot is determined by their respective medians. Intra-region variability can be assessed through the slope of the point array for a particular region, or through the spread of its points along the horizontal axis. And, the graphical display makes it easy to see potentially misleading elements of the data, such as within-region outliers, that might affect the calculated values of region-specific summary statistics.

Dot plots are certainly not limited to displays of raw data. They can also be used very effectively to show quantitative summaries of data values within partitions of an overall dataset either across subsets of the observations or across separate variables (or both). As an example, Figure 4 uses data from the 2004 CPS National Election Study to show the mean importance ratings that survey respondents assigned to ten issues (scored on a one-to-five scale, with larger values indicating the issue is more important). The symmetric horizontal "wings" around each plotted point correspond to the 95% confidence interval for each mean. Horizontal reference lines are omitted from this dot plot, not only because of the relatively small number of points, but also because they would interfere with visual perception of the error bars.

The dot plots presented here should hint at the variety of ways this kind of display can be used. Of course, there are many other possibilities. For example: The plotted points could be arranged according to some substantive scheme, rather than sorted by the data values; repeated measures of a variable could be handled by including more than one plotting symbol on each horizontal line; parameter estimates from statistical models (e.g., cell frequencies from a contingency table or coefficients from a regression equation) could be displayed; and so on. Regardless of specific context, the "trick" in constructing an effective dot plot lies

in adjusting the details to facilitate the kinds of judgments that the analyst wants readers to make when interpreting the graphical information in the display.

Advantages of Dot Plots

Dot plots are covered extensively in Cleveland's work (e.g., 1993; 1994) and my own monograph (Jacoby 1997) on statistical graphics. However, it is definitely accurate to say that the dot plot is a relatively uncommon graphical display in the political science research literature. This is unfortunate, because dot plots have some clear advantages over their "competitors" for displaying labeled data: pie charts and bar charts.

First, there is a simple, practical advantage: Dot plots can show a larger number of data points than either pie charts or bar charts. A dot plot can include a surprisingly large number of points– it is really only limited by the space available in the display medium. Pie charts are limited to a fairly small number of distinct "wedges;" otherwise, visual perception of the quantitative information becomes nearly impossible. With bar charts, the width of the bars necessarily becomes narrower as the number of distinct data values increases. In fact, with a large number of plotted values, a bar chart becomes virtually indistinguishable from a dot plot.

A second, theory-based, advantage is that dot plots facilitate relatively accurate graphical perception. Visual processing of a pie chart requires the observer to make comparative judgments about the angles, arcs, and/or sizes of the various wedges within the circular diagram. In contrast, a dot plot only involves comparisons of point locations along a common scale. Cleveland and McGill (1984) show that the latter task is usually carried out much more accurately than the former.

With bar charts, a different problem emerges. A bar chart *should* be interpreted using the relative heights (or widths) of the bars along a common scale. But, Cleveland (1984) argues that bar charts actually encourage observers to make judgments based upon the relative *sizes* of the bars within the display. And, if the scale in the bar chart begins at some arbitrary value, then it is inappropriate to regard the lengths and/or areas of the bars as any sort of meaningful information about the relative magnitudes of the quantitative values being displayed in the chart.

A properly-constructed dot plot avoids these problems by either extending the horizontal reference lines across the entire width of the plotting region (as in Figures 1 and 3) or omitting them entirely (as in Figure 4). In either case, the display provides no visual cues that would encourage inappropriate judgments about the relative sizes of the data values. If magnitude judgments are appropriate for the data, then the reference lines in a dot plot can be extended from the origin of the scale out to the plotted data values (as in Figure 2).

A third potential advantage emerges when a dot plot is used to display the distribution of values on a single variable. As explained earlier, such a display can be viewed as a transposed quantile plot. It shows all of the data and, therefore, provides a particularly accurate depiction of distributional shape. Just as with a quantile plot, a dot plot avoids potential distortions that may be introduced by the binning process required to construct a more traditional histogram.

Creating Dot Plots in R

Most statistical software can be used to generate dot plots. STATA, SPSS, and SYSTAT all have routines that are either explicitly designed, or easily adapted, for this purpose. Friendly (1991) provides macros that create dot plots in SAS. But, as is often the case, R provides the most powerful facility and flexibility for constructing this kind of graphical display (R Development Core Team 2006). In the discussion below, I will focus on the `dotplot` function in the `lattice` package (Sarkar 2006). Note, however, that most of the displays could also be produced with the `dotchart` function of the traditional R graphics system.

General Principles

In order to produce a dot plot of values stored in variable x, with labels in variable y, and both x and y contained in an R data frame called *dataset*, one could use the following R commands:

```
> library(lattice)
> dotplot(y ~ x, data = dataset, other optional arguments)
```

The first statement loads the `lattice` package. The second statement calls the `dotplot` function. Since the results of the function call are not assigned to an object, they are listed on the current output device (probably a window in the screen display). The only required argument in the `dotplot` function is `y ~ x`. This is actually a simple formula in the R modeling language; it will produce a dot plot with the labels from y listed along the vertical axis, and corresponding points located at the proper horizontal location according to their respective x values. The `data` argument is optional, but, it is very convenient specifying the data frame containing x and y.

Before proceeding to specific examples, it is useful to consider several general principles about the lattice package and the `dotplot` function. First, trellis graphs (i.e., the type produced by the `lattice` package) are created by issuing a *general display function* which, in turn, calls a *panel function*. The general display function sets up the exterior components of the graph (i.e., axes, scales, titles, and so on) while the panel function deals with everything within the plotting region, itself (i.e., points, reference lines, etc.).

In the present context, general display function `dotplot` calls panel function `panel.dotplot`. Distinguishing between these two components is important for specifying non-default parameter values in a graph (e.g., axis labels, plotting symbols, line characteristics, and so on).

Second, the `lattice` package effectively regards a dot plot as a scatterplot between a categorical variable (or a *factor* in R nomenclature) and a quantitative variable (a *vector* in R-speak). It is very useful to keep this in mind when trying to produce non-standard displays and also when trying to understand why the `dotplot` function sometimes produces strange and unexpected results!

Third, R defaults to an alphabetical ordering of the levels (i.e., the unique values) within a factor. This is usually problematic for dot plots because data values are typically random with respect to such an arrangement. Therefore, the plotted points in a dot plot created with an alphabetized factor would look like a shapeless "cloud." In order to prevent this from occurring, it is usually necessary to change the order of the factor's levels before calling the `dotplot` function. Given the factor, y and the quantitative variable, x (both contained in the data frame called *dataset*), the `reorder` function can be used to sort the factor's levels so that they are ordered according to the values of the variable:

```
> dataset$y <- reorder(dataset$y, dataset$x)
```

Note the use of the fully-qualified names (i.e., for each variable, the data frame is given first, followed by the dollar sign and the variable name) in the preceding R command. This is necessary in order to direct R to the proper data frame and also to guarantee that the newly-sorted factor is added to that data frame (thereby replacing the original version of factor y), rather than left as a separate object.

R Commands for Specific Examples

Let us begin by reproducing the dot plot shown back in Figure 1. The data are contained in an R data frame called *policy*, which contains two variables: *state* and *priority* for the state names and policy priority scores, respectively. Figure 1 is created with the following statements:

```
> policy$state <- reorder(policy$state, policy$priority)
> dotplot(state ~ priority, data = policy,
+    aspect = 1.5,
+    xlab = "State policy priority scores, 1992",
+    scales = list(cex = .6),
+    panel = function (x, y) {
+    panel.dotplot(x, y, col = "black", lty = 2)})
```

The first statement sorts the levels of the factor, *state*, by the values of the vector, *priority*. The second statement (which extends across six lines) calls the `dotplot` function. The first line of the function should be self-explanatory. The next three lines contain arguments that are part of the general display function. First, `aspect` sets the aspect ratio, so that the height of the graph is one and one-half times the size of the width (the default is an aspect ratio of 1.0). Next, `xlab` provides a label for the horizontal axis (the default is the variable name, which is often uninformative to readers). The `scales` argument provides a list; in this case, the list only contains a single element which uses the `cex` argument to set the size of the tick labels on the axes to 60% of their default size (otherwise, the state names would collide along the vertical axis).

The `panel` argument explicitly creates the panel function for this graph, by modifying the default elements of `panel.dotplot`. In this panel function, "`x`" and "`y`" are the horizontal and vertical axis variables in the dotplot (i.e., *priority* and *state*, respectively)— they are passed to the panel function from the general display function. The `col` argument sets the color of the plotting symbols (the default color is blue) and the `lty` argument (for "line type") specifies dashed horizontal lines, rather than the default lines (which are solid).

In fact, `panel.dotplot` actually calls two further panel functions: `panel.xyplot` to control the plotting symbols and `panel.abline` to control the reference lines. So, we could bypass `panel.dotplot` entirely and use the following function call to produce Figure 1:

```
> dotplot(state ~ priority, data = policy,
+   aspect = 1.5,
+   xlab = "State policy priority scores, 1992",
+   scales = list(cex = .65),
+   panel = function (x, y) {
+   panel.abline(h = as.numeric(y),
+   col = "gray", lty = 2)
+   panel.xyplot(x, as.numeric(y),
+   col = "black", pch =16)})
```

Here, "`as numeric(y)`" appears as an argument to both `panel.xyplot` and `panel.abline`. This specification coerces the vertical-axis variable into a numeric vector; the result is the ordering of the factor's levels. This enables `panel.xyplot` to treat the graph as a simple scatterplot. And, in `panel.abline`, the "`h = as.numeric(y)`" instruction places a horizontal line at each distinct location along the vertical axis. The remaining arguments in the panel functions should be fairly obvious.

In recreating Figure 1, the explicit use of the panel functions was not really necessary. Instead, the `col` and `lty` arguments simply could have been included as part of the `dotplot` general display function; in that case, `dotplot` would pass them along to the panel function when it is implicitly (but invisibly) called to construct the plotting region of the display. However, there are many situations where explicit use of the panel function is necessary in order to modify the default specifications of the `dotplot` function.

For example, assume that we want to construct a

dot plot of a ratio-level variable, with horizontal reference lines that only extend from the origin to the plotted points. This is problematic, because `panel.dotplot` creates lines that extend across the entire width of the plotting region. The easiest way to change this is to modify the panel functions. To show how this is done, we will recreate the display from Figure 2. The data are contained in an R data frame called *spending* which contains the factor *state* and the vector *educ.per.cap*. The former are the state names, and the latter are the 2000 education expenditures from each state. The following R statements produce the dot plot in Figure 2:

```
> spending$state <-
+   reorder(spending$state, spending$educ.per.cap)
> dotplot(state ~ educ.per.cap, data = spending,
+    aspect = 1.5,
+    scales = list(cex = .65),
+    xlim = c(-100, 2300),
+    xlab = "State education spending, fiscal year 2000",
+    panel = function (x, y) {
+    panel.segments(rep(0, length(x)), as.numeric(y),
+      x, as.numeric(y), lty = 2, col = "gray")
+    panel.xyplot(x, as.numeric(y), pch = 16, col = "black")} )
```

Once again, the first statement sorts the levels of the factor (*state* in this case) by the values of the quantitative vector (here, *educ.per.cap*). In the call to `dotplot`, the `xlim` argument specifies the range of values for the horizontal axis. Here, the minimum is set to -100 in order to provide a small margin to the left of the reference lines within the plotting region.

Turning to the panel function, the first two arguments to `panel.segments` are vectors containing the horizontal and vertical coordinates of the starting positions for the line segments. The `rep` function creates a vector of zeroes. The size of this vector is equal to the number of values being plotted (i.e., "`length(x)`"). So, this vector contains the horizontal coordinate for the starting point of each observation's line segment (i.e., they all begin at zero). The third and fourth arguments to `panel.segments` are vectors containing the coordinates of the terminal points for the line segments. The horizontal coordinate of the terminal point for each observation is simply the value of the variable being plotted (i.e., `x`). The vertical coordinates for the initial and terminal points of the line segments are both supplied by "`as.numeric(y)`".

Next, `panel.xyplot` plots the data points, themselves. Once again, the function replaces "`x`" and "`y`" with the actual variable names, which are passed from the general display function. The "`pch = 16`" specification sets the plotting symbol to a solid circle. The remaining arguments in both the general display function and the panel function should be self-explanatory.

Next, we will recreate the divided dot plot shown back in Figure 3. Producing this kind of display with the `dotplot` function is a bit tricky and it involves as much work in preparing the dataset as it does in specifying the function. The data are contained in a data frame called *regions*; the contents of the text file used to create this data frame are shown in Table 1. The first line is a header record, giving the variable names. The observations are grouped by region and, within each region, they are sorted by values of the *priority* variable. Following the last observation for each region, there is a "dummy" observation. In each one, the value of *state* is the region name and the value of *priority* is -9. The latter is an arbitrary, meaningless, value which falls outside the actual range of the variable.

The first step in creating the divided dot plot is to fix the order of the factor, *state*, to that shown in Table 1. This can be accomplished by using the following R commands:

```
> regions$sequence <- seq(1, length(regions$priority))
> regions$state <- reorder(regions$state, regions$sequence)
```

This newly-ordered factor is employed as the vertical axis variable in the dot plot, as follows:

```
> dotplot(state ~ priority, data = regions,
+    aspect = 1.5,
+    xlab = "State policy priority scores, 1992",
+    xlim = c(.49, .55),
+    scales = list(cex = .65),
+    panel =function (x, y) {
+    panel.dotplot(x[x > 0], y[x > 0],
+    pch= 16, col = "black", lty = 2)} )
```

Most elements of the preceding function should be familiar to the reader. The only new technique is the use of logical conditions within the square brackets following the `x` and `y` variables specified in `panel.dotplot`. The panel function will only carry out the plotting tasks for those observations where the condition is true. Since the region labels were given values of -9 on the *priority* variable (i.e., `x` in `panel.dotplot`), the condition is false for those observations. Therefore, no horizontal lines or data points are plotted in those cases.

The panel function only affects the interior of the plotting region. Therefore, all 52 levels of the factor, *state* (i.e., the 48 state names plus the four region names), still appear along the vertical axis (which is, of course, outside the plotting region). In this case, it is important to specify the `xlim` argument so that it just contains the legitimate values of the *priority* variable. Otherwise, the general display function for `dotplot` would regard the nonsense value, -9, as the minimum value of *priority* for purposes of constructing the horizontal axis.

As our final example, we will examine the R commands used to create Figure 4, the dot plot of ten sample means with error bars representing confidence intervals. R makes it very easy to pass information from a statistical analysis over to a graphing function, with a minimum of manual copying or cutting-and-pasting. Assume that the

Figure 1: 1992 State Policy Priority Scores

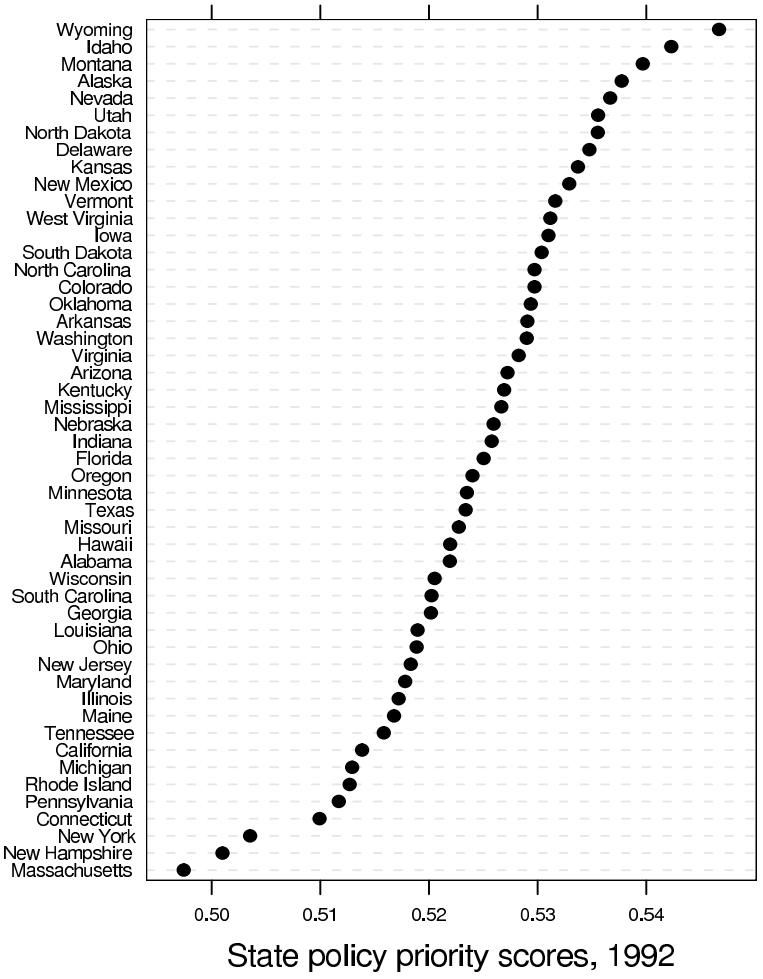
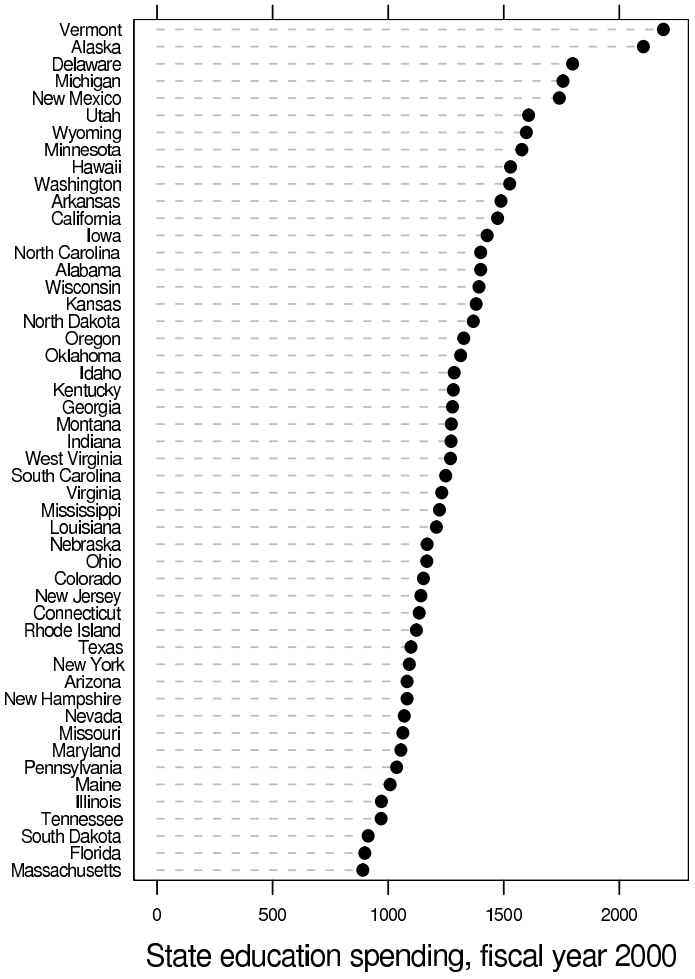
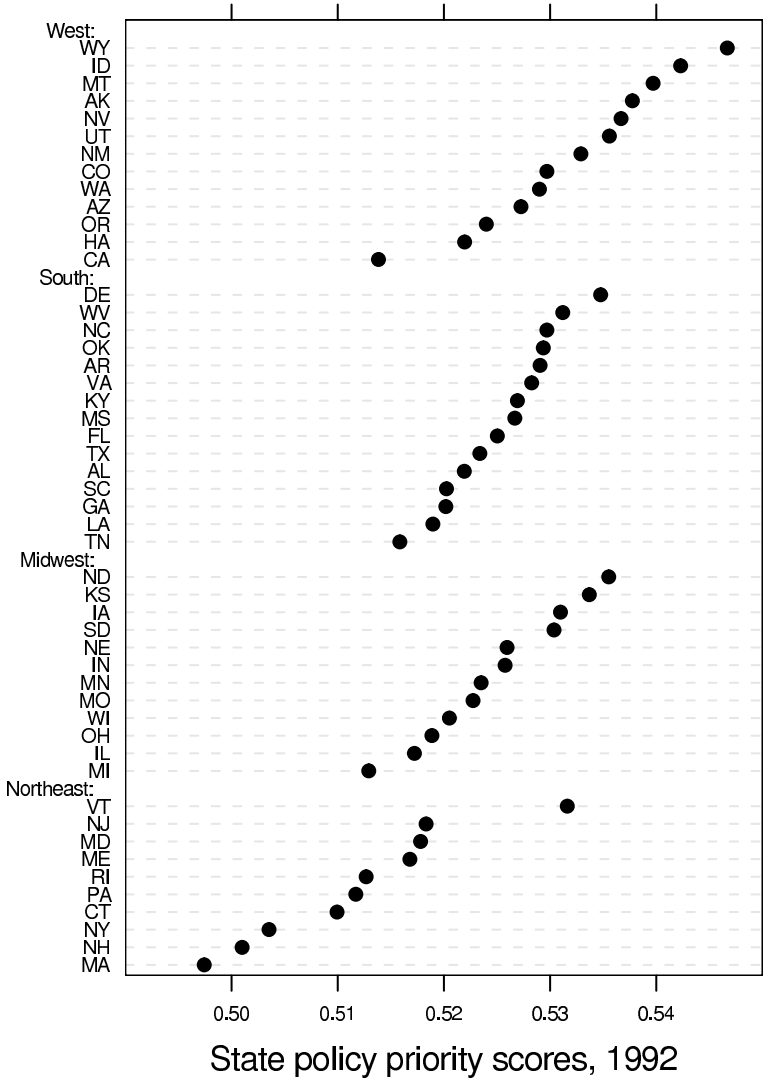


Figure 2: State Education Spending, 2000 (in dollars per capita)



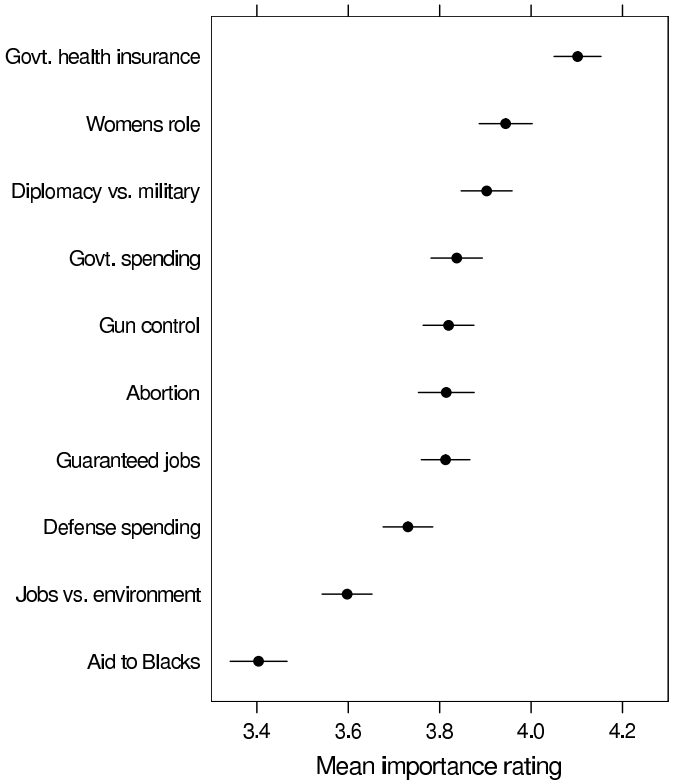
Data Source: *State Government Finances.*

Figure 3: 1992 State Policy Priority Scores, by Region



Data Source: Jacoby and Schneider (2001).

Figure 4: Mean Issue Importance Ratings, 2004



Note: Error Bars Represent 95% Confidence Intervals Around the Respective Mean Values. Data from the CPS National Election Study.

raw data used to calculate the statistics plotted in Figure 4 are contained in a data frame, *import.2004*, with 1212 rows (i.e., the sample size for the 2004 NES) and ten columns (one for each of the importance ratings). The task is complicated a bit by the presence of missing values (coded as NA's) within the data. Descriptive labels for the ten variables in *import.2004* are contained in the separate vector, *var.labels*. The means and confidence intervals for the dot plot are created with the following statements:

```
> sample.means <- mean(import.2004, na.rm = T)
> std.devs <- sd(import.2004, na.rm = T)
> sample.ns <- apply(!is.na(import.2004), 2, sum)
> std.errs <- std.devs / (sample.ns ^ .5)
> lower <- sample.means +
+   (std.errs * qt(.025, (sample.ns - 1)))
> upper <- sample.means +
+   (std.errs * qt(.975, (sample.ns - 1)))
> new.data <- data.frame(var.labels,
+   sample.means, lower, upper)
> new.data$var.labels <- reorder(new.data$var.labels,
+   new.data$sample.means)
```

The first two statements use R's statistical functions to calculate the column means and standard deviations from data frame *import.2004*; in each case, the result is a ten-element vector. The next statement determines the number of nonmissing observations within each column. The ten-element *sample.ns* vector is created by using the `apply` function to sum within the columns of a logical matrix created by the argument "`!is.na(import.2004)`". The latter matrix is the same size as *import.2004*. It has value TRUE in the cells that correspond to nonmissing data in *import.2004*, and FALSE in the cells that correspond to missing data. Now, TRUE evaluates to one and FALSE to zero in the `sum` function). So, summing within the columns of this logical matrix produces the number of nonmissing values on each of the ten variables.

The *std.errs* vector contains the standard errors of the sample means, created by dividing the elements of the *std.devs* vector by the square roots of the elements in the *sample.ns* vector. The lower and upper bounds of the confidence intervals for the respective means are obtained by adding the product of the standard errors and the appropriate t values (obtained using the `qt` function) to the means. Next, the `data.frame` function concatenates the vectors of variable labels, sample means, and the limits of the confidence intervals into a new data frame, called *new.data*. Finally, the levels of the factor *var.labels* are ordered according to the sample means, using the `reorder` function. The display in Figure 4 is produced by the following call to `dotplot`:

```
> dotplot(var.labels ~ sample.means, data = new.data,
+   aspect = 1.5,
+   xlim = c(3.3, 4.3),
+   xlab = "Mean importance rating",
+   panel = function (x, y) {
+   panel.xyplot(x,y, pch = 16, col = "black")
+   panel.segments(new.data$lower,as.numeric(y),
+      new.data$upper, as.numeric(y),
+   lty = 1, col ="black")} )
```

Once again, we use `panel.xyplot` rather than `panel.dotplot` in order to eliminate the horizontal reference lines. The `panel.segments` function draws the error bars using the variables *lower* and *upper* as the horizontal coordinates for the ends of the line segments. Note that the fully-qualified variable names must be specified, since the general display function does not pass the name of the data frame from the `data` argument to the panel function.

Further Resources and Conclusions

Hopefully, the examples presented above will help readers use the R statistical computing environment in order to generate not only basic dot plots, but also more complex versions of this graphical display. As with any set of specific examples, those provided here only scratch the surface of a potentially vast subject. For that reason, further documentation would be very helpful. As a starting point, there are a number of relevant materials on my own web site, including a longer version of this article, a number of datasets, and R scripts to produce not only the dot plots discussed here but also many others as well. The URL is: `http://polisci.msu.edu/jacoby/`.

More generally, the most convenient source of information is the R online help system. However, many users find the help files for `lattice` functions to be a bit terse. As a more user-friendly alternative, the *S-Plus Trellis Graphics User's Manual* is an excellent guide to the entire trellis system and its general usage. Another extremely helpful source of information is "A Tour of Trellis Graphics," by Richard A. Becker, William S. Cleveland, Ming-Jen Shyu, and Stephen P. Kaluzny. This paper expands upon the basic information provided in the *User's Manual* and provides detailed examples illustrating how to create and modify trellis graphs. While these documents were written for the trellis graphics system in the commercially-available S-Plus computing environment, virtually all of their content applies directly to the `lattice` package in R, as well. Both are available on the Trellis Display web site: `http://cm.bell-labs.com/cm/ms/departments/sia/project/trellis/`.

I have also included copies of these two documents on my own web site. Still another source of information is the book, *R Graphics*, by Paul Murrell. This is a general reference work which provides comprehensive and readily-accessible treatment of both the traditional and the grid (which contains the `lattice` package) graphics systems in R. Finally, John Fox's book, *An R and S-Plus Companion to Applied Regression*, provides an enormous amount of information and advice about working with the statistical and

graphics functions in R.

In conclusion, dot plots are excellent graphical displays for labeled quantitative data values. They contain a great deal of information, are easy to interpret, and overcome a number of the problems associated with other kinds of displays. Dot plots are also extremely flexible; they can be modified in various ways to handle many different data analysis situations. They are useful both for analytic purposes (to paraphrase Tukey, they show the researcher features that he/she never expected to see) and for presentational displays (i.e., they guide the observer to perceive the researcher's conceptions of the most important features in the data). For all of these reasons, dot plots (along with the requisite programming knowledge to create them) constitute a very useful addition to the methodologist's "toolbox."

Listing of text file used to create the R data frame, called regions, used to construct Figure 3

```
state priority
MA      0.49743
NH      0.50099
NY      0.50354
CT      0.50994
PA      0.51171
RI      0.51269
ME      0.51679
MD      0.51781
NJ      0.51831
VT      0.53162
"Northeast:   "  -9
MI      0.51292
IL      0.51722
OH      0.51886
WI      0.52052
MO      0.52275
MN      0.52350
IN      0.52577
NE      0.52595
SD      0.53037
IA      0.53099
KS      0.53370
ND      0.53553
"Midwest:   "    -9
TN      0.51585
LA      0.51896
GA      0.52019
SC      0.52024
AL      0.52193
TX      0.52337
FL      0.52503
MS      0.52667
KY      0.52691
VA      0.52825
AR      0.52905
OK      0.52937
NC      0.52970
WV      0.53118
DE      0.53475
"South:   "      -9
CA      0.51385
HA      0.52196
OR      0.52400
AZ      0.52724
WA      0.52900
CO      0.52970
NM      0.53291
UT      0.53557
NV      0.53668
AK      0.53774
MT      0.53969
ID      0.54230
WY      0.54669
"West:   "       -9
```

References

Becker, Richard A.; William S. Cleveland; David A. James. (1996) S-Plus Trellis Graphics User's Manual (Trellis Versions 2.0 & 2.1). Seattle, WA: MathSoft, Inc.

Becker, Richard A.; William S. Cleveland; Ming-Jen Shyu; Stephen P. Kaluzny. (1996) "A Tour of Trellis Graphics." Unpublished manuscript.

Cleveland, William S. (1984) "Graphical Methods for Data Presentation: Full Scale Breaks, Dot Charts, and Multibased Logging." *American Statistician* 38: 270-280.

Cleveland, William S. (1993) *Visualizing Data*. Summit, NJ: Hobart Press.

Cleveland, William S. (1994) *The Elements of Graphing Data (Revised Edition)*. Summit, NJ: Hobart Press.

Cleveland, William S. and Robert McGill. (1984) "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods." *Journal of the American Statistical Association* 79: 531-553.

Fox, John. (2002) *An R and S-Plus Companion to Applied Regression*. Thousand Oaks, CA: Sage.

Friendly, Michael. (1991) *SAS System for Statistical Graphics*. Cary, NC: SAS Institute.

Jacoby, William G. (1997) *Statistical Graphics for Univariate and Bivariate Data*. Thousand Oaks, CA: Sage.

Jacoby, William G. and Saundra K. Schneider. (2001) "Variability in State Policy Priorities: An Empirical Analysis." *Journal of Politics* 63: 544-568.

Murrell, Paul. (2006) *R Graphics*. Boca Rotan, FL: Chapman & Hall/CRC.

R Development Core Team (2006). "R: A Language and Environment for Statistical Computing." Vienna, Austria: R Foundation for Statistical Computing.

Sarkar, Deepayan. (2006). "lattice: Lattice Graphics." R Package, Version 0.13-8.

The Art of Benchmarking: Evaluating the Performance of R on Linux and OS X

Jasjeet S. Sekhon
UC Berkeley
sekhon@berkeley.edu

With the growing use of computational statistical methods which tax even today's powerful computer chips, it is of interest how various applications perform on modern operating systems.[1] Of course, there is more to picking an operating system than speed (e.g., ease of administration, viruses, and most importantly applications). But speed is key especially when purchasing servers and clusters which many of us are doing. How various statistical packages perform is also an important consideration. For example, is Matlab generally faster than R ("yes") and are both faster than Stata ("yes")? But since much of the statistics and political methodology community has coordinated on R, I focus on it and examine how efficiently it runs on various operating systems.

The short summary is that for some key operations Linux is faster than Windows XP and both are faster than OS X unless the default OS X memory allocator is replaced. In fairness to OS X, the Windows XP version of R already uses a modified memory allocator instead of the system default. Although some Windows benchmarks are presented here, the focus is on Linux and OS X benchmarks because few if any methods people (I don't know of any!) would run computational servers with Windows.

Modern computers and operating systems are so complex and the tasks they are ask to perform so variable that there is no single measure or test of overall performance. Such a measure of performance is even more elusive than g—the general intelligence factor. Therefore, it is of utmost importance when conducting benchmarks that they be as closely related as possible to the computations one will perform with production code. If R is the application whose performance one cares about most, relying on benchmarks from seemingly closely related tasks such as video encoding can often lead to erroneous inferences even though both tasks are floating-point intensive.[2] Even within R, on should benchmark the code one is actually going to be using. For example, if one going to be running Matching estimators using a given algorithm, then it is best to run benchmarks as similar to this usage scenario as possible. The relative performance of various operating systems or computer chips when inverting matrices may be different than the relative performance when sorting the contents of a matrix even though both tasks are done in R and use the same data. The different tasks may, for example, have different memory requirements.

The difficulty of understanding and separating out what is happening on a computer is a cautionary tale regarding the difficulty of making inferences in general. Modern computers, which are Turing machines, are deterministic systems. There is nothing stochastic about them, and there are no unobservable factors at play. For example, even the "random" number generators most statistical software rely on are generated by deterministic algorithms such as the Tausworthe-Lewis-Payne generator (Bratley, Fox and Schrage 1983) which is one of the better ones currently available—it is, for example, used by `rgenoud`.[3] Notwithstanding these deterministic properties, it is difficult to determine why a certain algorithm performs better on one operating system than another. To make our inferential task easier, I try to limit the number of factors in play by using exactly the same hardware for each operating system. Thanks to Apple's switch to Intel chips, this matching exercise is now straightforward to do. Back when Apple was still using PowerPC chips, comparisons between OS X and Windows were more difficult to conduct, and for any given benchmark Apple or Microsoft could blame the chip instead of the operating system or claim that a given application was tuned to one particular chip or another.[4]

All benchmarks are conducted on what is at the time of this writing, Apple's fastest Intel hardware: a MacBookPro with Intel Duo (2.16GHz) and 2GB of RAM and 120GB hard disk. This machine has two CPU cores, but all of the presented benchmarks only use one of the cores.

Given that one should benchmark the algorithm one will actually be using, benchmarks are based on my Matching package for R (Sekhon 2006).[5] The package provides functions for multivariate and propensity score matching and for finding optimal balance based on a genetic search

[1] I thank Nate Begeman of Apple for software optimizations and Michael Herron for helpful suggestions.

[2] Floating-point numbers are the way that a subset of real numbers are represented on a modern computer. Such numbers consist of an integer exponent and a signifcand which consists of the significant digits of the number. On a computer, floating-point operations are handled differently than integer operations and different chips may be better at one than the other. For example, the latest generation AMD Opteron chips generally have better floating-point performance than Intel x86 chips, but the latter have better integer performance.

[3] `rgenoud` is an **R** package for Genetic Optimization Using Derivatives and it is used below. See `http://sekhon.berkeley.edu/rgenoud`.

[4] On a more mundane level, the same deterministic random number generator will produce different "random" numbers on different chips because of architectural differences. There are obvious ways around this issue, but this issue and many others like it complicate benchmarking across chips.

[5] See `http://sekhon.berkeley.edu/matching`

algorithm. A variety of univariate and multivariate tests to determine if balance has been obtained are also provided. The most computationally intensive part of the package is the `GenMatch` function which finds optimal balance using multivariate matching where a genetic search algorithm determines the weight each covariate is given (Sekhon 2006, Diamond and Sekhon 2005). Because the genetic algorithm calls the matching function many, many times, a great deal of time has been spent optimizing the matching procedure itself. Indeed, after I had posted earlier benchmarks on my website which looked particularly poor for OS X, programmers at Apple, include members of Apple's OS X Performance Group, helped optimize my code. After these and other optimizations, my matching algorithm—implemented in the `Match` function—is the fastest I know of in any language. The computationally intensive portion of the code is written in C++, and key matrix operations are handled by the BLAS libraries.[6] So any benchmark using this algorithm becomes a benchmark of floating-point performance, the system BLAS implementation, compiler performance, and operating system memory management.[7] In order to eliminate the possibility of these factors confounding our results, we match on all of the non-operating system factors. The same BLAS implementation is used for both Linux and OS X as well as the same compiler (gcc) and optimization flags. And since we are using the same computer, the CPU's floating-point unit, cache and other hardware components are identical in all simulations. We are then left with differing operating systems.

Linux versus Mac OS X on Intel Dual Core

In early May I posted on my website benchmarks comparing Linux and OS X (I later added Windows XP benchmarks).[8] In one of the original benchmarks, both Linux and Windows XP were more than twice as fast as OS X. And in a second (more representative) benchmark, Linux was about 20% faster than OS X. The benchmarks were posted on Digg (`http://digg.com`) and a variety of other high traffic Internet websites such as OSnews (`http://OSnews.com`). This attention generated a lot of comments and suggestions.

With the help of a variety of developers working at Apple and elsewhere, the large OS X performance gap previously reported was significantly reduced. The most important improvement is the use of a more efficient algorithm which relies on optimized BLAS to perform key matrix operations. This change increased the performance of the code on all platforms. The performance gap was further closed by compiling and linking R on OS X against Doug Lea's malloc (called dmalloc for short). Malloc is a function which allocates memory for an object (such as a variable or matrix) requested by a program, in this case R.

However, a Linux speed advantage remains which varies with dataset size. For example, the gap ranges from 0% for a small dataset to 10% for what is a medium size dataset for the algorithm in question. The gap shrinks again to 0% for a larger dataset. The performance gap is much greater if the default OS X malloc is used notwithstanding the new algorithm: the gap goes from essentially zero for a small dataset, to 40% for a medium one, and up to 50% for a large one. Therefore, I recommend that the OS X version of R be compiled so that memory is handled by dmalloc instead of the default OS X malloc. R for OS X should be linked against dmalloc just as it is for Windows.

The default malloc on OS X, like the default malloc on Windows XP, causes a large performance degradation relative to the default malloc on Linux. R developers use the default system malloc on every operating system but Windows. It turns out that this decision is a bad one in the case of OS X because OS X makes more frequent system calls when allocating memory. A system call when allocating memory is done so that the kernel can allocate and clean up memory. This helps with memory defragmentation and insures that free memory is recovered by the operating system so it can be used by other processes. However, all of this comes at a cost. Not only do these system calls take time, they also cause page faults. These occurs when a program requests data that is not currently in real memory. The operating system then fetches the data from virtual memory and loads it into RAM. Therefore, it is much faster for the application to not call the kernel and simply manage memory itself. The downside of this is that the operating system will generally run out of memory more quickly. The threshold at which different memory allocators turn memory allocation over to the kernel varies greatly. A key performance issue arises because OS X makes system calls for allocations of 15 kilobytes (KB) and larger while, for example, the current version of dmalloc makes system calls for allocations of 256KB and larger.[9]

To make matters worse, unlike on Linux, this thresh-

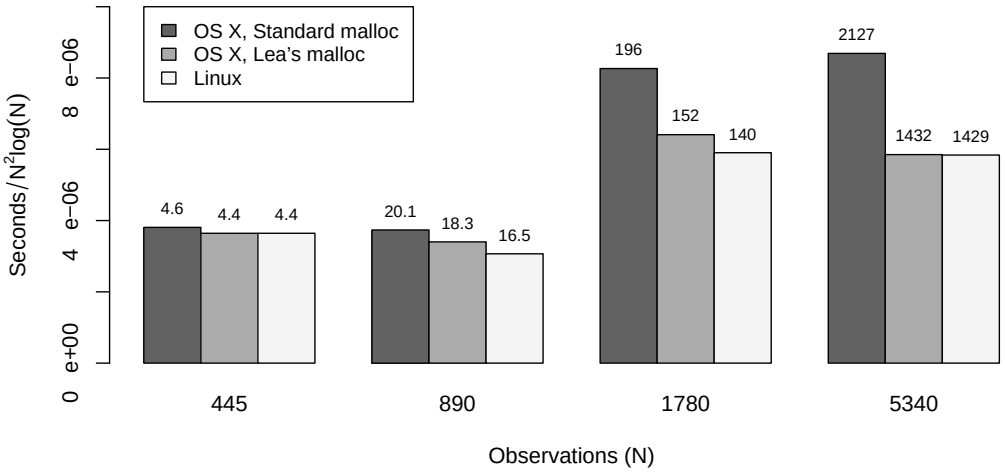
[6] The BLAS (Basic Linear Algebra Subprograms) are routines for performing basic vector and matrix operations. They provide a consistent programming interface across hardware specific implementations.

[7] The same BLAS implementation was used for all of the benchmarks, ATLAS (non-threaded). The OS X vecLib Framework is based on the ATLAS BLAS. Goto's BLAS are currently the fastest, but at the time these benchmarks were run they were not available for x86 OS X. I have since helped Goto produce a beta patch for x86 OS X.

[8] `http://sekhon.berkeley/macosx`

[9] One KB is equal to $2^{10} = 1024$ byes. Dmalloc's threshold used to be 128KB but it was increased to 256KB as computers have changed—compare version 2.6.6 with 2.8.3.

Figure 1: Benchmark Comparison



Entries above bars are gross processing seconds. Scaling factor for the y-axis is the average asymptotic order of the algorithm which is $O(N^2log(N))$. Asymptotically, the dominating factor is quicksort which is applied N times, and the average order of quicksort is $O(Nlog(N))$.

old is not changeable by the user. On Linux, it is common practice in high performance computing to completely avoid calls to the kernel when allocating memory for the computationally intensive process.[10] On Linux, which uses the GNU malloc, calls to the kernel (via the mmap function) can be avoided completely by setting two runtime environmental variables: MALLOC_TRIM_THRESHOLD to -1 and MALLOC_MMAP_MAX to 0. It is unfortunate that it is not possible to do something similar with OS X's default malloc because it would help alleviate the performance issue. Therefore, one has to use an alternative memory allocator on OS X.

Picking what the threshold should be for a memory allocator to allow the kernel to allocate memory is an art. There is no generally optimal solution. An issue which arose with OS X was that professional video and graphics users were running out of allocatable memory, thus the current OS X memory allocator minimizes the amount of memory used—i.e., it needs to aggressively recover freed memory and optimally allocate memory pages. This was an issue because OS X only assigns 2GB of memory space to processes regardless of the amount of physical memory in a computer. Linux does not have this limitation. Therefore the current OS X memory allocator was written to minimize memory usage even if that means that much smaller memory allocations are handled by the OS X kernel than by the Linux kernel.

The Benchmarks

As noted above, the benchmarks are based on my Matching package for R (Sekhon 2006). The benchmark scripts only vary by the sample size of the dataset being examined. The data sizes are: 445, 890, 1780 and 5340 observations.[11] Each script runs the benchmark three times and the best runtime of the three is recorded. Each script is executed 1000 times and the average times are reported below. The setup is outlined in Table 1.

Figure 1 presents the results for the benchmarks. For the first benchmark, there is only a small difference between Linux and OS X with the default malloc and no difference when dmalloc is used. This is the benchmark which was used to optimize the software on OS X by Apple's OS X Performance group. The difference between Linux and OS X default malloc is small but statistically significant—the p-value based on the empirical distribution over 1000 simu-

[10] For example, see the following documentation from Lawrence Livermore National Laboratory `http://www.llnl.gov/LCdocs/linux/index.jsp?show=s7`.

[11] Each script runs the benchmark three times and the best runtime of the three is recorded. Each script is executed 1000 times and the average times are reported below. The scripts are available on my website. 445: `http://sekhon.berkeley.edu/macosx/GenMatch.R`, 890: `http://sekhon.berkeley.edu/macosx/GenMatch2.R`, 1780: `http://sekhon.berkeley.edu/macosx/GenMatch3.R`, and 5340: `http://sekhon.berkeley.edu/macosx/GenMatch5.R`.

lations is 0.09.

The results for this benchmark with the new code are in sharp contrast to the original results (which were obtained using exactly this script), but which used an algorithm which was not as optimized—e.g., it did not make use of the BLAS libraries. In the original benchmark, Linux took 8.84 seconds, Windows XP 9.38 seconds and OS X (with the default malloc) 22.78 seconds! Clearly the code improvements have speed up the code on all operating systems (on Linux from 8.84 seconds to 4.4), but the improvement for OS X has been tremendous: from 22.78 seconds to 4.6!

However, even with the new algorithm, as we increase the sample, differences begin to become more pronounced. With 890 observations, Linux is now 20% faster than default OS X (p-value=0.00) and 10% faster than OS X with dmalloc (p-value=0.00).

For 1780 observations, there is some evidence that the difference between Linux and OS X with dmalloc is either asymptoting at about 10% or possibly even shrinking—from 1.11 times as slow as Linux in the previous benchmark to 1.08 times as slow now. The next simulation will help to nail this down. In any case, the gap between Linux and the default OS X malloc version has doubled and OS X is now about 1.4 times slower than Linux.

In an attempt to answer the asymptoting vs shrinking gap question, a benchmark was run with 5340 observations (12 times the original dataset). The Linux advantage over OS X using dmalloc was only present for a given range of dataset size and with 5340 observations it has disappeared once again. But the gap between default OS X and Linux continues to grow.

Why exactly there remains a gap for some dataset sizes between the OS X dmalloc and the Linux results is not clear. One way to try to answer the question is to use Shark which allows one to see what functions an application is spending its time in.[12] With the default OS X malloc, Shark is able to quickly show that the process, in this case the 5240 observations benchmarks, is spending a lot of time calling a system library.[13] However, with dmalloc, it is not clear why OS X is slower than Linux for some sample sizes. No system call jumps out, so a mystery remains which further analysis would no doubt clear up.[14]

Conclusion

This brief report gives a flavor of how to conduct software benchmarks. It is striking how even though everything a computer does is deterministic and observable, the experimental method is still essential for making inferences. Methods like those proposed by qualitative researchers for making inferences with deterministic systems are not used in the literature. When benchmarking, it is common to match on (and hence eliminate) as many confounders as possible and to report measures of uncertainty. Since computers are deterministic, the remaining uncertainty must come from confounders. For example, when conducting these benchmarks, I tried to stop as many daemons as possible.[15] But background tasks, such as those related to the graphics system whether it be X11 or Quartz, will still take up CPU resources at times the analyst does not predict.

References

Bratley, P., B.L. Fox and L.E. Scrage. 1983. *A Guide to Simulation*. New York: Springer-Verlag.

Diamond, Alexis and Jasjeet S. Sekhon. 2005. "Genetic Matching for Estimating Causal Effects: A General Multivariate Matching Method for Achieving Balance in Observational Studies." See `http://sekhon.berkeley.edu/papers/GenMatch.pdf`.

Sekhon, Jasjeet S. 2006. "Matching: Algorithms and Software for Multivariate and Propensity Score Matching with Balance Optimization via Genetic Search." See `http://sekhon.berkeley.edu/matching`.

[12] For information about Shark see `http://developer.apple.com/tools/sharkoptimize.html`.

[13] About 12% of the runtime is spent in 'match_msg_trap' which is a symbol in the libSystem.B.dylib library. The only other libSystem calls taking more than 1% are '_isnand' (1.4%) and 'dyld_stub_isnand' (1.3%). 'malloc' itself is reported to take up (directly) 0.0% of the runtime—so calls to it are being accounted elsewhere.

[14] With dmalloc the only libSystem calls which take up greater than 1% of the time are '_inand' (2.5%), 'dyld_stub__isnand' (2.5%) and 'syscall' (1.2%). '_mmap' takes up 0.3% of the runtime and 'malloc' 0.4%. Does the large amount of time spent in 'mach_msg_trap' indicate that XNU kernel is spending a lot of time message passing as it is often accused of or is this simply how Shark is reporting the kernel's default virtual memory manager?

[15] A daemon is a background process which handles services as automatic disk mounting and printing.

Table 1: Operating System and Computer

Label	OS and Chip
OS X Standard malloc	Tiger on MacBookpro, Intel 2.16GHz Dual Core 2GB RAM
OS X Lea's malloc	Same as above but with dmalloc
Linux	Ubuntu Linux (Drapper Drake) with i686-SMP kernel on MacBookpro, Intel 2.16GHz Dual Core 2GB RAM Note: Xorg server running with GNOME

The LaTeXCorner

PowerPoint™ for LaTeX: The beamer Class

Ian Yohai
Harvard University
yohai@fas.harvard.edu

Many of the readers of *The Political Methodologist* are undoubtedly already sold on the benefits of LaTeX as a word processing program. The ease with which LaTeX handles complicated mathematical expressions, figures and tables, references, and any number of other tasks is a strong incentive for those presenting quantitative material to break free of the point and click structure of, shall we say, other popular programs. But in an age where lectures and conference presentations seem dominated by a constant stream of PowerPoint™slides, there finally exists a LaTeX alternative that can compete. While other presentation programs for LaTeX such as Prosper and PPower4 have existed for some time, the BEAMER[1] class offers a dizzying array of options designed not only to handle the π, Σ, and $\int$ in your presentations seamlessly, but also to satisfy the needs of those who want to spice up the aesthetics of their slides.

What Can BEAMER Do?

Perhaps most importantly, BEAMER can do just about anything that LaTeX can, in approximately the same way. For example, the syntax to create formulas and tables, incorporate figures, and bold or italicize particular portions of text works in BEAMER just as it does in the standard `article` class. The advantage of this for those who know LaTeX is clear: it is quite simple just to lift a table or formula in a working paper and throw it into a BEAMER presentation. The downside, of course, is that familiarity with LaTeX is an absolute prerequisite for using BEAMER. But the good news is that if you are already using LaTeX for the bulk of your word-processing, you have enough experience to adapt to BEAMER pretty quickly.

BEAMER, however, offers quite a good deal more than simple compatibility with LaTeX. First, BEAMER comes stocked with a number of canned "presentation themes," which control just about every feature of the slides (except the substance, of course). The themes, which are named for various cities like Madrid, Frankfurt, Ann Arbor, and so on, set the color layout for the background, choose the font size for headers and footers, and even pick the type of symbol used for bullet points in itemized lists — be it a circle, triangle, star or some other option. Given that BEAMER is structured so that there are many choices for each one of these "elements" of a presentation, it is quite handy to have one theme specified at the beginning that controls most aspects of the layout.

Brave readers should have a look at the BEAMER manual to get a feel for all the possibilities.[2] It will soon be obvious that the aforementioned presentation themes grow quite rapidly in complexity. The relatively simple themes just require a title for each slide. More advanced themes allow the user to create something like a table of contents

[1] According to the BEAMER manual, the package was created by Till Tantau in 2003 for his Ph.D. defense. I have heard that the name BEAMER derives from the German word for video projector, though I do not know all of the details.

[2] Available at `http://www.ctan.org/tex-archive/macros/latex/contrib/beamer/doc/beameruserguide.pdf`.

(TOC) that can appear either as a header or as a sidebar. As the presentation progresses, the first section of the moving TOC dims out as the next section is highlighted. This is useful to keep the audience (and the presenter!) informed about what is to come next in the presentation. Some of the most advanced themes jam so much information into headers, footers, and sidebars that the main point of a slide can often be lost. While I personally think that such themes detract from the substance of talks, others may find them useful to rigorously structure their presentations into sections and subsections. Here again is an advantage of BEAMER. There is a canned theme ready for just about everyone — from those who only want a large title at the beginning of each slide, to those who want a full blown navigation tree of their presentation. And for the artistically inclined or for those who are not satisfied with the canned themes, it is possible to create your own custom theme.[3]

Beyond mere aesthetics, BEAMER also has a powerful option for creating "overlays," or pauses scattered throughout a given slide. For example, one can use this feature to show the audience one bullet point at a time (while concealing the rest) or show some text first and then move to a figure at the bottom of a slide. It is even possible to move through a table row by row. This command is appropriately called `uncover`. While `uncover` functions somewhat similarly to the pause commands in PPower4, `uncover` can actually do quite a bit more. It can be used to show material at the top and bottom of the slide first, and then reveal the middle section. While this might seem like a bit much, it is quite useful in the class setting, where the instructor wants to setup a problem and then reveal the answer after the students have had a chance to offer their own solution. Finally, `uncover` can be used to reveal sections of a flow diagram or schematic one at a time, no matter where on the slide the particular elements of the figure happen to reside.

Although users will probably want to incorporate overlays in their presentations, it is possible to produce a version without the pauses, simply by using the `[handout]` option in the preamble, while keeping everything else the same. For long presentations like class lecture notes, this option is useful if the instructor wants to post the entire set on a website, so that students can print out the material without having to scroll through all the pauses. You can even create those PowerPoint™-like handouts with multiple slides on a page.

Startup Costs

Learning all of the advanced features of BEAMER will no doubt take several hours of frustration, but for someone with a good deal of familiarity with LaTeX it should be possible to create an elegant presentation after about 10 or 15 hours of experimentation. While not trivial, the time will be well spent, particularly for those who want to incorporate existing LaTeX material into their presentations.

Below is some example syntax that could form the basis of a simple BEAMER presentation. Of course, those seeking all the details should read the BEAMER manual, which is quite comprehensive.

```
\documentclass{beamer}
\usetheme{Madrid}

\title{Our First \textsc{beamer} Presentation}
\author{Ian Yohai}

\begin{document}

\begin{frame}
  \titlepage
\end{frame}

\begin{frame}
  \frametitle{The First Slide}
  \begin{itemize}[<+->]
    \item The first bullet point.
    \item  Now let's try something more fancy.
    \uncover<+->{This line
      will only be revealed after the previous one.}
    \item  We can include equations, too.
      \begin{align*}
        \beta=(X'X)^{-1}X'y
      \end{align*}
    \item The last bullet point on this slide.
  \end{itemize}
\end{frame}

\end{document}
```

Much of the syntax looks quite similar to the `article` class in LaTeX, with the first exception being that the BEAMER class is specified instead. In this case, I have specified the `Madrid` theme, which is a relatively simple one as themes go. `Madrid` creates a wide header at the top of each slide containing the title of the slide, and also contains a footer that consists of the author's name, the title of the presentation, and a helpful counter that indicates the current slide number out of the total number of slides. The theme has a rather soothing blue on white color scheme. As I noted earlier, there are many other options if this theme proves to be either too busy or too simple.

You may have also noticed the `begin{frame}` and `end{frame}` syntax. Each slide is roughly equivalent to one frame,[4] and hence all the substantive material should be placed between the two commands. Not surprisingly, the `frametitle` command specifies the title of the slide. The `itemize` environment works the same as in a standard LaTeX document, except the `[<+->]` syntax specifies that I want

[3] Again, details are available in the BEAMER manual. Modifying or creating a theme basically requires setting options for headers, footers, sidebars, colors, and font types.

[4] Technically a frame can span two or more slides, but most purposes it is convenient just to think of a frame as one slide.

each item to be shown one at a time, rather than all at once. This little trick saves the user from having to use the `uncover<+->` command for each item. To reveal a portion of an item after the rest, I also use `uncover<+->` within the `itemize` environment.

Obviously any introduction presented here will merely scratch the surface, but I hope I have demonstrated that elementary features in BEAMER are not difficult to use if you have prior knowledge of LaTeX. There are a few other minor details that will have to be learned through experimentation. For example, you may be familiar with the `verbatim` commands that tells LaTeX to print text exactly as it is entered. This is useful when the text contains symbols such as $ or % that would normally be read by LaTeX as commands. To use the `verbatim` option in BEAMER, you have to specify the `[fragile]` option when you begin the frame.[5] Needless to say, that particular little oddity was not clear to me until reading the manual, but overall the transition was quite smooth.

Finally, you may be wondering how a BEAMER presentation is actually displayed. When compiled with `pdflatex`, the output is simply a pdf file. Since pdf viewers are available for nearly all platforms, including Windows, OSX, and Linux, viewing the presentation is quite easy. This is an especially nice feature when traveling to conferences where the available operating system may be different than on your own personal machine.

How to Get BEAMER

If I have not scared you away already, the latest version of BEAMER can be downloaded from `http://sourceforge.net/projects/latex-beamer`. You should also download the associated PGF and XCOLOR packages, which contain additional files that are needed to produce a presentation. Installation instructions will depend on the platform on which you run LaTeX, but some basic information is provided in the BEAMER manual.

Book Review

Review of Keith T. Poole Spatial Models of Parliamentary Voting

Joshua D. Clinton
Princeton University
clinton@princeton.edu

Spatial Models of Parliamentary Voting. Keith T. Poole. Cambridge University Press, New York, 2006, 248 pages. $24.99, ISBN 0521617472 (paperback); $65.00, ISBN 0521851947 (hardcover).

In recent years, the use of ideal point estimates has exploded. No longer used simply to measure the preferences of members of Congress, ideal points are now widely used in many studies of American politics to analyze state legislatures (e.g., Aldrich and Battista 2002; Wright and Schaffner 2002) and historical assemblies such as the Confederation Congress (Jenkins 2000) and the Continental Congress. Moreover, the application of ideal point models is no longer restricted to the legislative arena; applications use ideal points to measure preferences for voters (e.g., Lewis 2001), justices (e.g., Martin and Quinn 2002), presidents (e.g., McCarty and Poole 1995; Poole 1998) and executive agencies (e.g., Bertelli and Grose 2003). Nor is the usage restricted to the analysis of American politics, as the technology has also been used to analyze voting behavior in both comparative (e.g., Londregan 2000; Morgenstern 2004) and international (e.g., UN (e.g., Voeten 2000; 2004) and the European Parliament (e.g., Hix, Noury and Roland 2006; Hix 2006) institutions. Ideal points offer more than just the promise of characterizing the political environment, they are also prominently used to test theories of lawmaking (e.g., Cox and McCubbins 2005) and delegation (Epstein and O'Halloran 1999) as well as test accounts of party pressure (e.g., Groseclose and Snyder 2000; McCarty, Poole and Rosenthal 2001) and strategic voting (e.g., Clinton and Meirowitz 2004). That is, whereas roll call voting behavior was first used to characterize the amount of party influence in the legislature (e.g., Lowell 1902; Stuart 1928), roll call analysis now typically involves the estimation of ideal points for use in testing the predictions of formal theories.

Given their ever-increasing use, it is critical that political scientists understand the benefits and limitations of ideal points. To this end, Keith Poole offers *Spatial Models of Parliamentary Voting*. *Spatial Models* largely focuses

[5] Apparently this is necessary so that overlays work properly with `verbatim` text.

on: 1] outlining how behavioral voting models can be used to derive statistical voting models, and 2] discussing the computation and interpretation of the resulting ideal point estimates.

The fundamental point of the book – at least in my interpretation – is a point worth reiterating: "Simply pushing a matrix of roll call data through a computer program does not itself produce a meaningful picture" (4). *Spatial Models* consistently and coherently argues that the statistical models used to analyze roll calls are necessarily models of legislator behavior and that the appropriateness of resulting ideal point estimates depends critically on an underlying behavioral model. *Spatial Models* is an important and notable book precisely because it provides a detailed presentation of many of the most popular models used to estimate ideal points in political science.

Spatial Models differentiates itself from Poole's prior work on the analysis of roll calls (see, for example, Poole (2000; 2001) and Appendix A of *Congress: A Political-Economic History of Roll Call Voting* (1997) with Howard Rosenthal) in two important ways.

First, the book presents statistical models in much greater detail. Readers who may have found earlier expositions too quick should take comfort in the fact that the models receive a much more discursive treatment in *Spatial Models*. For example, Poole pays particular attention to discussing the geometric interpretation of ideal point estimators and the algorithms used to compute estimates. The detailed descriptions and justifications will be welcomed by scholars making use of the technology and the estimates.[1] That said, the book does assume a strong familiarity with linear algebra and maximum likelihood.

Second, *Spatial Models* describes the connections and differences between the various models used to estimate ideal points and the relationship between the statistical models and the underlying behavioral models of voting. For example, in Chapter 2, Poole starts with a discussion of the analysis of perfect spatial voting then moves to the analysis of voting behavior in the presence of voting errors in later chapters. In so doing, he establishes how maximizing the classification of votes and the probability of observing votes yields different estimators.[2] The book nicely illustrates how different assumptions lead to different estimators – for example, why the Optimal Classification estimator may sometimes be more appropriate for analyzing voting behavior than the probabilistic models depending on the reasonability of the parametric error assumptions (e.g., Rosenthal and Voeten 2004).As Poole notes in the introduction "the spatial theory of voting is a theory of behavior that states if a set of assumptions holds, then voters should behave in a certain way and we should observe certain types of outcomes. It is a theory that makes predictions that can be tested" (9).

Spatial Models is not a substantive book in terms of providing and testing different models of legislator behavior, nor does it discuss why roll calls may or may not be relevant for particular applications. Although there are hints in the introductory and concluding chapters of a larger argument regarding the constraints that cognitive capacity may place on the dimensionality of a decision space, the book is focused almost exclusively on methodological and computational issues raised in the analysis of voting behavior. It is very much a manual for ideal point estimation.

The book proceeds in a sensible and straightforward manner – it starts with a discussion of error-less voting and concludes with clever applications using ideal points to test the impact of changing parties, retirement and redistricting on legislators' voting behavior. To provide a more nuanced account of the book it is useful to briefly consider the content of the more important chapters.

Chapter 2 discusses roll call voting if voting is perfect (i.e., there is no error present and members always vote correctly). Chapter 3 builds on this framework and discusses what happens in the presence of voting errors and how to generate an estimator based on the correct classification of observed votes. The Optimal Classification (OC) algorithm Poole proposes is discussed in great detail and the presentation illustrates how the geometry and spatial models discussed in Chapter 2 can be used to generate an estimator.

Chapter 4 discusses the implications of maximizing the probability of observing member choices rather than the correct classification of votes. This results in the standard probabilistic spatial model underlying most uses of ideal point estimates. Poole discusses the models and estimation of: NOMINATE, Quadratic-Normal, Heckman-Snyder, and the Bayesian simulation approach. In so doing, he notes the similarities and differences between the estimators – a useful discussion for scholars interested in deciding which off-the-shelf model to use. This chapter should be required reading for any scholar making use of these methods or their estimates.

Chapter 5 focuses largely on "under-the-hood" (or, more accurately, inside the computer chassis) computing issues. Poole discusses issues such as assessing model fit, generating starting values, the complications raised by imposing constraints, and how to determine the dimensionality of the space. The chapter consists largely of rules of thumb Poole has acquired throughout his lengthy career. Although

[1] On a lighter note, this reviewer appreciates (finally!) knowing the basis for some of the more obscure naming conventions Poole sometimes uses in his code (see, for example, fn. 4 in Chapter 3.)

[2] In fact, *Spatial Models* is partially an intellectual history of the use and development of ideal point estimation techniques in political science. Discussions of how the models were refined in light of technological advances and alternative behavioral models are sprinkled throughout the book.

largely of interest to those seeking to program their own estimators, it is also useful for scholars interested in computing their own estimates and assessing model fit and the dimensionality of the space.

Chapter 6 ("Conducting Natural Experiments with Roll Calls") contains a nice discussion of how ideal point estimates can be used to directly test accounts of legislative behavior. As ideal point estimates are possible only as a consequence of assuming a particular behavioral model, Poole illustrates how the resulting estimates can be used to assess theories of legislative behavior. Specifically, Poole examines whether voting behavior changes for members who switch parties, for members whose district is redistricted, or for members who decide to retire. The discussion nicely notes the benefits and potential pitfalls of the "natural experiment" approach and, subject to the caveats Poole notes, the chapter should provide scholars with a tool useful for investigating behavior in the face of institutional or electoral change.

In sum, *Spatial Models* should be required reading for all users of ideal point estimates. At a minimum, every user of NOMINATE (and its variants) should read Chapter 4 and every user of OC should read Chapter 3. In fact, given the prevalence with which ideal point estimates are used in political science – in no small part due to Poole's many contributions – the discipline would benefit from a much larger audience. As instructors of the next generation of scholars, political methodologists should certainly use this book to teach the uses (and abuses) of ideal point estimates and the importance of moving from assumed behavioral models to estimates with testable predictions.[3] The book is probably most appropriate for use in an advanced class populated by graduate students with a familiarity with maximum likelihood and linear algebra.

The underlying message of *Spatial Models* that "anyone can construct a spatial map...but the maps are worthless unless the user understands both the spatial theory that the computer program embodies and the politics of the legislature that produced the roll calls" (pg. 209) is an important point with broad implications. By providing a detailed discussion of the former, the book makes an important contribution while charting a path for future research.

References

Aldrich, John H. and James S. Coleman Battista. 2002. "Conditional Party Government in the States," *American Journal of Political Science*, 46(1): 164-72.

Bertelli, Anthony M. and Christian R. Grose. 2003. "A Bayesian Analysis of Senate Votes on Administrative Appointments," Texas A & M working paper.

Clinton, Joshua D., and Adam Meirowitz. 2004. "Testing Accounts of Legislative Strategic Voting: The Compromise of 1790," *American Journal of Political Science*, 48(4):675-89.

Cox, Gary W. and Mathew D. McCubbins. 2005. *Setting the Agenda: Responsible Party Government in the US House of Representatives*. NY, NY:Cambridge University Press.

Epstein, David and Sharyn O'Halloran. 1999. *Delegating Powers: A Transaction Cost Politics Approach to Policy Making Under Separate Powers*, NY,NY: Cambridge University Press.

Groseclose, Tim and James M. Snyder Jr. 2000."Estimating Party Influence in Congressional Roll-Call Voting," *American Journal of Political Science*, 44(2) 193-211.

Hix, Simon, Abdul Noury, and Gerrard Roland. 2006 "Dimensions of Politics in the European Parliament," *American Journal of Political Science*, 50(2):494-511.

Hix, Simon. 2006 *Democratic Politics in the European Parliament*, NY,NY: Cambridge University Press.

Jenkins, Jeffrey A. "Examining the Robustness of Ideological Voting: Evidence from the Confederate House of Representatives," *American Journal of Political Science*, 44(4):811-22.

Lewis, Jeffrey B. 2001. "Estimating Voter Preference Distributions from Individual-Level Voting Data," *Political Analyis*, 9(3):275-97.

Londregan, John. 2000.*Legislative Institutions and Ideology in Chile*. NY,NY: Cambridge University Press.

Lowell, Lawrence A. 1902. "The Influence of Party Upon Legislation in England and America," *Annual Report of the American Historical Association for 1901* Washington DC: AHA pp. 321-542.

Martin, Andrew D. and Kevin M. Quinn. 2002. "Dynamic Ideal Point Estimation via Markov Chain

[3]As a side note, Poole's recounting of the development of the methods of roll call analysis in political science also serves as a nice illustration of scientific progress. *Spatial Models* recounts how political scientists have worked to develop and refine a set of tools that are currently used in almost every subfield of political science.

Monte Carlo for the U.S. Supreme Court, 1953-1999" *Political Analysis*, 10(2):134-53.

McCarty, Nolan M. And Keith T. Poole. 1995. "Veto Power and Legislation: An Empirical Analysis of Executive and Legislative Bargaining from 1961 to 1986," *Journal of Law, Economics & Organization*, 11(2):282-312.

McCarty, Nolan M., Keith T. Poole and Howard Rosenthal. 2001. "The Hunt for Party Discipline in Congress," *American Political Science Review* 95(3):673-687.

Morgenstern, Scott. 2004. *Patterns of Legislative Politics: Roll Call Voting in the United States and Latin America's Southern Cone*, Cambridge University Press:NY,NY.

Poole, Keith T. 1998 "Recovering a Basic Space for a set of Issue Scales," *American Journal of Poltical Science*, 42(3):964-93.

Poole, Keith T. 2000. "Non-parametric Analysis of Binary Choice Data," *Political Analysis* 8(3):211-37.

Poole, Keith T. 2001. "The Geometry of Multidimensional Quadratic Utility in Models of Parliamentary Roll Call Voting," *Political Analysis*, 9(3):211-26.

Poole, Keith T. and Howard Rosenthal. 1997. *Congress: A Political-Economic History of Roll Call Voting*, Oxford University Press: NY, NY.

Rice, Stuart A. 1928 *Quantitative Methods in Politics*, NY,NY: Knopf.

Rosenthal, Howard and Erik Voeten. 2004. "Analyzing Roll Calls with Perfect Spatial Voting: France 1946-1958," *American Journal of Political Science* 48(3):620-32.

Voeten, Erik. 2000. "Clashes in the Assembly," *International Organization*, 54(2): 185-217.

Voeten, Erik. 2004. "Resisting the Lonely Superpower: Responses of States in the United Nations to U.S. Dominance," *Journal of Politics*, 66(3):729-54.

Wright, Gerald C. and Brian F. Schaffner. 2002. "The Influence of Party: Evidence from the State Legislatures," *American Political Science Review*, 96(2):367-79.

Section Activities

A note from our Section President

Please join us for our annual business meeting, which is held at the APSA meetings. The date is Saturday, September 2, 2006. Location information is not yet available but will be posted on Polmeth or check your program. The Long Range Planning Committee will be presenting its first report at the meeting. Jim Granato (jgranato@mail.la.utexas.edu) is the chair of the committee. (A full list of all committees and their members is available on Polmeth at `http://polmeth.wustl.edu/society.php`). The purpose of the inaugural report is to explore opportunities for broadening participation in the activities and mission of the Political Methodology Society and to consider the format of the annual meeting. Our annual awards will also be presented at the business meeting.

I would like to thank Langche Zeng, University of California, San Diego, for putting together the Political Methodology Section's APSA panels. Our section is sponsoring three of the fifteen working groups at the APSA meeting (`http://www.apsanet.org/section_584.cfm`). Ken Cousins, University of Maryland, is the coordinator for the working group on Social Network Analysis, Rebecca Morton, New York University, is the coordinator for Experiments, Causality, and the Study of Politics, and Stephen Purpura, Harvard University, is the coordinator for Automated Content Analysis and Computer Annotation. We appreciate their ingenuity for organizing these intellectual adventures.

The search for the successor editor(s) for TPM is underway. Adam Berinsky, Michael Herron, and Jeffery Lewis will soon step down as editors. We thank Adam, Michael, and Jeff for their service to the section. TPM is widely read by members of the section and we appreciate their high standards. Larry Bartels (bartels@Princeton.EDU) is chairing the search committee. Nancy Burns, Jon Pevehouse, and Christopher Zorn are the other committee members. If you have a nomination, please email one of the committee members. Self nominations are welcomed.

Jan Box-Steffensmeier
The Ohio State University

Announcements

Announcement - Association of Religion Data Archives

The Association of Religion Data Archives (ARDA), located at `http://www.thearda.com`, provides free access to high quality quantitative data on religion. The ARDA allows you to interactively explore American and international data using online features for generating national profiles, maps, church membership overviews, denominational heritage trees, tables, charts, and other summary reports. Over 400 data files are available for online preview (including the International Social Survey Program and multiple years of the General Social Survey) and virtually all can be downloaded free of charge. The ARDA has also developed a series of tools for education. Learning modules provide structured class assignments and the many online tools allow students to explore religion across the globe or in their own backyard. Housed in the Social Science Research Institute at the Pennsylvania State University, the ARDA is funded by the Lilly Endowment and the John Templeton Foundation.

The Political Methodologist
Department of Political Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Nonprofit Org.
U.S. Postage
Paid
MIT

The Political Methodologist is the newsletter of the Political Methodology Section of the American Political Science Association. The support of the MIT Department of Political Science in helping to defray the editorial and production costs of the newsletter is gratefully acknowledged.

Subscriptions to *TPM* are free to members of the APSA's Methodology Section. Please contact APSA (202 483-2512, `http://www.apsanet.org/about/membership-form-1.cfm`)to join the section. Dues are $25.00 per year and include a free subscription to *Political Analysis*, the quarterly journal of the section.

Submissions to *TPM* are always welcome. Articles should be sent to the editor by e-mail (`berinsky@mit.edu`) if possible. Alternatively, submissions can be made on diskette as plain ascii files sent to Adam J. Berinsky, MIT Department of Political Science, 77 Massachusetts Avenue, Cambridge, MA 02139 E53-459. LaTeX format files are especially encouraged. See the *TPM* web-site, `http://polmeth.wustl.edu/tpm.html`, for the latest information and for downloadable versions of previous issues of *The Political Methodologist*.

TPM was produced using LaTeX on a PC running MikTex and WinEdt.