The Political Methodologist

Newsletter of the Political Methodology Section
American Political Science Association
Volume 14, Number 2, Fall 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

Welcome to the latest issue of *The Political Methodologist*, and the last under our editorial regime. Beginning with the next issue, *TPM* will be in the capable hands of Paul Kellstedt, Dave Peterson, and Guy Whitten at Texas A&M. We hope that you have enjoyed the issues we have produced over the last three years and we look forward to joining all of you as consumers, rather than producers of *TPM*.

This issue begins with an article by David Firth and Arthur Spirling on resources that make it possible to analyze voting data from the House of Commons at Westminster. We then move a pair of articles in our Computing and Software section. The first, by David Park, Andrew Gelman, and Noah Kaplan discusses how to run WINBUGS through R. The second, by Simon Jackman draws attention to R's data processing capabilities by outlining procedures for importing data from web pages into R. Ryan Moore then reviews another title in the Analytical Methods for Social Research series, published by Cambridge University Press, Essential Mathematics for Political and Social Research, by Jeff Gill. We close the issue with notes from our section president, Janet Box-Steffensmeier and a message from the new *TPM* editorial team.

The Editors

Articles

tapiR and The Public Whip: Resources for Westminster Voting

David Firth and Arthur Spirling
University of Warwick and University of Rochester
d.firth@warwick.ac.uk and spln@mail.rochester.edu

As no doubt most readers of TPM are aware, the spatial analysis of legislatures has become a mainstay of our discipline. With some important exceptions, almost all this work has been based on Congressional roll call voting in the United States. This is unfortunate but understandable: unfortunate because there are other parliaments in the world with potentially very interesting patterns to be uncovered, but understandable because (a) data has not been easily available in electronic or machine readable format for non-US parliaments and (b) the most technically minded scholars in political science have historically been Americanists. Here we intend to draw interested readers' attentions to a set of new resources, partly contributed by the current authors, that ameliorate the data availability problem for the United Kingdom (from 1992 onwards) and hence allow analysis of the House of Commons at Westminster. This is probably not the place to wax lyrical about the joys of studying Britain's House of Commons, suffice to note that it is literally is *the* Westminster system that readers may (hopefully!) recall from introductory Comparative politics classes, and contrasts with the US Congress insofar as it shows highly cohesive parties and a 'fused' legislature and executive (Cabinet).

tapiR and error corrected votes

Parliamentary voting in the United Kingdom is recorded in both hard and electronic copy in the volumes of the official *Hansard* record. Unfortunately, the pages of *Hansard* provide only individual division lists in textual form and even the online *Hansard* webpages—available for all House of Commons debates and votes since 1995—cannot be used directly for analysis, as they contain inconsistencies in the names used for Members of Parliament (MPs), extraneous markup codes etc.

The current authors provide two resources: data on House of Commons voting for the period 1992 to 2005; and software tools to enable future House of Commons divisions to be added easily by accessing the relevant pages of *Hansard* online and incorporating the votes of each MP into a single rectangular dataset. As we explain in the next section, the software has become somewhat redundant since, in recent times, others have taken up the mantle of collating and recording votes in essentially the same way, so we only briefly discuss our package here. It was designed for the `R` statistical computing environment and is named *tapiR* which stands for *tools for accessing parliamentary information in R*.[1]

Three separate datasets are provided for the 1992–1997 parliament, the 1997–2001 parliament and the 2001–2005 parliament. The data are in spreadsheet format, with rows representing MPs and the columns representing divisions. The first column after each name represents the MP's political party; in the few cases where the MP's affiliation changes during the period party membership at the end of the parliament is recorded.

Table 1 illustrates the format of the data. Shown in Table 1 are votes for the first ten MPs alphabetically, in the first three divisions of the session which began after the May 1997 General Election. Divisions are identified by the Division Number as reported in *Hansard*, and the date on which the division took place. Votes are either '`y`' or '`n`', signifying respectively Ayes and Noes as recorded in the *Hansard* division lists (including the Tellers for each side). A dash means that the MP did not vote, i.e., was in neither the Ayes nor the Noes. Note that it is not possible to distinguish between different types of non-voting, the three most important such being absence from the House, deliberate abstention when present, and 'pairing' in which MPs on opposite sides of an issue agree that neither need vote as their votes would cancel (although this disappeared as an arrangement after 1997). Occasionally, *Hansard* records an MP in both the Ayes and Noes lists for a division, and in such instances we have used the code '`b`' (for 'both'); this can happen either because the MP went through both the Aye and the No lobbies, for example because of a change of mind or the realization of a mistake on their part, or as the result of an error in the published division lists. The original hard copy records were rechecked manually for the latter cases, and this typically resolved the issue (i.e. we were able to ascribe `y` or `n` to the MP in question). In this sense the

[1] TPM readers should be aware of the conceptual distinction between the *tapiR* which is our software package and the *tapir*, which is a large (300–600lbs) browsing forest-dwelling mammal, roughly pig-like in shape but with a short, prehensile trunk.

Table 1: A small part of the voting data from May, 1997

	party	div001.970519	div002.970520	div003.97052020
Abbott, Diane	Lab	n	n	n
Adams, Irene (Pai)	Lab	-	n	n
Ainger, Nick	Lab	n	n	n
Ainsworth, Peter (E S)	Con	y	y	-
Ainsworth, Robert (Cov)	Lab	n	n	n
Alexander, Douglas	Lab	-	-	-
Allan, Richard (She)	LD	-	n	y
Allen, Graham (Not)	Lab	n	n	n
Amess, David	Con	y	y	-
Ancram, Michael	Con	y	y	-
Anderson, Donald (Swa)	Lab	n	n	n

votes are 'error corrected' and *ipso facto* a more accurate record than the online pages of *Hansard* themselves.

Each of these three datasets are made available as a standard 'comma-separated values' (`.csv`) file of the kind that can be read by many standard computer programs, including Stata, SPSS and Excel. In Stata the data for 1997 to 2001, for example, can be read in by using the command

```
insheet using Votes9701.csv
```

and in `R` the same data can be read into a data-frame object by

```
Votes9701 <- read.csv("Votes9701.csv",
        row.names = 1)
```

where `row.names=1` means that the first column of the file, the MP names, will be used as row names in the resultant data frame. The (zipped) data are held at Firth's personal website at the University of Warwick.[2]

The files are rather large: in 1992–97 there were 668 distinct MPs and 1285 divisions, and in 1997–2001 there were 671 MPs and 1279 divisions. Our experience with Stata and `R` has been that this is unproblematic; but with other software packages, especially those which impose limits on the number of data columns, some external preprocessing may be necessary, for example to select a subset of the divisions for analysis. According to reports we have received from users, the current version of Microsoft Excel appears to have difficulty with data of this width.

All divisions up to the last vote of the 2001–2005 parliament are already recorded in the files described above. The *tapiR* package is available on the Comprehensive `R` Archive Network (CRAN) for straightforward installation into the user's local implementation of `R`. The package contains R functions to perform data-handling tasks described below.

The package is internally documented in `R` in the usual way, and essentially converts one or more *Hansard* division lists into columns of a spreadsheet. There are three main tasks involved in this operation, and *tapiR* provides functions to complete them:

1. Obtain a complete 'master' list of unambiguous MP names for the parliamentary session. Some MPs share the same name, and our approach to eliminating ambiguity in such cases is to append the first three characters of an MP's constituency name; thus Peter Ainsworth's constituency, which *Hansard* normally records as '`(E Surrey)`', becomes '`(E S)`'.

2. Download the relevant page(s) from the online *Hansard*, extract the division-list information and strip out redundant information such as hypertext markup (`HTML`) codes.

3. For each MP name in each division list obtained, identify the MP in the 'master' list and record their vote in the spreadsheet, or '`-`' if no vote, as shown in Table 1.

Once a 'votesheet' object has been created for a set of divisions, it can be operated upon in the standard ways for data frames in `R` . In particular, the combination of votes for two or more sets of divisions is easily achieved using `R`'s `cbind` function. To make the voting data available for analysis in other systems, a standard 'comma-separated values' (`.csv`) file can be written by using the tapiR function `write.csv`; for example,

```
> write.csv(Votes4Feb03, file = "Votes4Feb03.csv")
```

Obviously, this function will work for all `R` data frames, regardless of whether they pertain to parliamentary voting or not. One further *tapiR* function worth noting here is

[2] `http://www.warwick.ac.uk/go/tapir/`

`write.votesheet`, which makes a compact representation of the voting data either for on-screen display or in a file. For example,

```
> write.votesheet(Votes4Feb03)
                          party votes
Abbott, Diane               Lab -nyyy
Adams, Irene (Pai)          Lab -ynnn
Ainger, Nick                Lab nynnn
Ainsworth, Bob (Cov)        Lab nynnn
Ainsworth, Peter (E S)      Con -----
Alexander, Douglas          Lab nnyyy
...
```

The codes '`y`', '`n`', etc., can be changed as desired. For example, certain analytic computer programs commonly used in the discipline—like `NOMINATE` (e.g. Poole & Rosenthal (1997))—take as their input `(1,6,0)` in place of `(y,n,-)`; to write such a file, with code '`b`' also replaced by '`-`' wherever it occurs, the usage of `write.votesheet` would be of the form

```
> write.votesheet(Votes4Feb03,
    file = "Votes4Feb03.txt", keep.b = FALSE,
    aye = "1", no = "6", novote = "0")
```

The Public Whip

The Public Whip, currently located at `http://www.publicwhip.org.uk/` is a website that performs almost identical tasks to *tapiR* and is continually updated with new Westminster voting data as it occurs. Rather than `R`, the authors of *The Public Whip* use `Perl` to capture division data from XML files of parliamentary debates. This information is then loaded into a `MySQL` database, where it can be accessed directly from the website.

This arrangement allows users to search by date, or by a particular bill, or by a particular MP for division voting information. The website also contains various links to MP's constituency addresses, and MP's parliamentary speeches via an associated website, `http://www.TheyWorkForYou.com`. There are various summary statistics available for each MP, such as their attendance at votes and their 'rebelliousness.' This latter information comes with the important caveat that it is literally calculated as the proportion of times that the MP in question votes against the majority position as voted for by members of the MP's party. Readers familiar with British politics will know that this tends to overstate how often MPs vote contrary to their party whip (which is the usual understanding of 'rebel'), but this is (a) openly acknowledged via *The Public Whip*'s frequently asked questions and (b) somewhat unavoidable because British political parties do not publicize which votes are whipped and which are 'free.'

The data matrices for the 1997–2001 parliaments and 2001–2005 parliaments are provided at `http://www.publicwhip.org.uk/project/data.php`, also in comma delimited format, but with slightly different numerical coding. The columns and rows are switched relative to *tapiR*'s data, so that the rows contain the name, number and date of the divisions, and the columns contain the MP identifying number.

Conclusion

Above, we presented two resources for those interested in roll call voting and British politics. Our own efforts, *tapiR*, and its associated data sets were generated via `R`, and contain voting records from 1992–2005 for Westminster. *The Public Whip*'s efforts are complete from 1997 through 2005 and beyond, in the sense that they are updating their records continuously. Both resources are free to researchers, and our work has some particular licence conditions pertaining to due citation and non-commercial use that users are requested and required to abide by. There is further information on this topic at Firth's website. Source code for both projects is available under the GPL license. As suggested above, we will not continue to update our data using *tapiR*, since *The Public Whip* is essentially fulfilling our objectives, though, of course, *tapiR* and our data remain available for anyone interested in using them.

References

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

Computing and Software

R2WinBUGS: Running WinBUGS from R

David K.Park, Andrew Gelman, and Noah Kaplan
George Washington University, Columbia University, and University of Houston
dkp@gwu.edu, gelman@stat.columbia.edu, and nkaplan@uh.edu

Introduction

Bayesian inference is the process of fitting a probability model to a set of data and summarizing the result by a probability distribution on the parameters of the model and on unobserved quantities such as predictions for new observations. There are several options for researchers to use Bayesian inference in R (2006).[1] One such option is to use the R package R2WinBUGS to call WinBUGS (1999) from R.[2] First, why use WinBUGS?

- Bugs is one of the most general programs currently available for Bayesian inference.[3]
- Its model specification closely resembles the theoretical specifications of models. Therefore,
  - Bugs is useful for teaching, since students can easily follow the move from theoretical specification of a model to computational specification for estimation purposes; and
  - Bugs facilitates the development and estimation of models for which routines have not been written: it's a wonderful environment to work if there is no canned routine for the problem of interest.

Why not just use WinBUGS directly? Why do we need R2WinBUGS? There are several advantages to calling WinBUGS from R:

- R2WinBUGS Automatically writes the data, generates initial values, lists the parameters to extract, calls the model to be read by WinBUGS, and returns an object with a convenient graphical and numerical summary.
- Saving the WinBUGS estimate as an R object makes it easier to manage the posterior simulations for further analysis and to combine with the data stored in R that were used to fit the model.

Example: Dialogue in American Political Campaigns

In this note, we use an example, understanding dialogue between competing political candidates, to demonstrate the usefulness of R2WinBUGS. Consider the goal of estimating the distribution of dialogue, y_i, across campaigns and issues, i. For example, the level of dialogue between two competing candidates on an issue like abortion in the 2002 Alabama Senate campaign. We have indicators for the different campaigns, j, for example, the 2002 Senate campaign in Alabama, and the different issues, k, for example, abortion. We have important campaign-level predictors — the competitiveness of the campaign, cq_j, and the level of negativity, negative_j; and important predictors as the issue level — if the issue is "owned" by a particular party, ownership_k, and the salience of the issue, salience_k.[4]

We also have several error terms that we need to consider. First, the errors ϵ_i represent "within campaign-issue variation" which in this case includes measurement error and variation between campaign-issues. Second, the errors η_j represent variation between campaigns, beyond what is explained by campaign-level predictors. Finally, the error ν_k represent the variation between issues, beyond what is explained by issue-level predictors.

Writing a WinBUGS model

We formally write the model as:

$$y_i = \alpha_0 + \lambda_{j[i]} + \beta_{k[i]} + \epsilon_i \text{ for campaign-issue } i = 1, \ldots, n$$

$$\lambda_j = \gamma_1 \cdot \text{cq}_j + \gamma_2 \cdot \text{negative}_j + \eta_j \text{ for campaigns } j = 1, \ldots, J$$

$$\beta_k = \delta_1 \cdot \text{ownership}_k + \delta_2 \cdot \text{salience}_k + \nu_k \text{ for issues } k = 1, \ldots, K$$

[1] For a comprehensive list of R packages that provide tools for Bayesian inference see the CRAN Task View: Bayesian Inference at http://cran.r-project.org/src/contrib/Views/Bayesian.html.

[2] R2WinBUGS works with OpenBUGS which is an open-source version of WinBUGS.

[3] Various R packages exist that fit particular Bayesian models, e.g., MCMCPack.

[4] This is a simplified version of the model presented in Kaplan, Park and Ridout (2006).

where $j[i]$ represents the campaign j containing campaign-issue i, and $k[i]$ represents the issue k containing campaign-issue i.

Now that we have formally written down the model, we need to translate that model to `WinBUGS` notation. We can simply write down the `WinBUGS` model in a text editor, such as `Emacs` or `WinEdt`.

Campaign-issue level model
Before we do that, however, rewrite () as:

$$y_i \sim N(\alpha_0 + \lambda_{j[i]} + \beta_{k[i]}, \sigma_y^2)$$

First, `WinBUGS` does not allow composite expressions in its distribution specifications, thus we cannot write `y[i] ~ dnorm(alpha.0 + lambda[campaign[i]] + beta[issue[i]], tau.y)` because the first argument to `dnorm` is too complex. So, we split () into two lines:

$$\begin{aligned} y_i &\sim N(\hat{y}_i, \sigma_y^2) \\ \hat{y}_i &= \alpha_0 + \lambda_{j[i]} + \beta_{k[i]} \end{aligned}$$

The campaign-issue level model can be written in `WinBUGS` as follows:

```
model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- alpha.0 +
                lambda[campaign[i]] +
                beta[issue[i]]
    }
```

Second, we use the variable names `campaign[i]` and `issue[i]` for the indexes which we have labeled as $j[i]$ and $k[i]$ in the mathematical formulation of the model. This makes it easier to interpret the indexes as constructed and stored in R, and frees up the variables j and k to use as loop counters in the Bugs model. Finally `WinBUGS` parameterizes normal distributions in terms of the inverse-variance, $\tau_y = 1/\sigma_y^2$, a point to which we shall return shortly.

Campaign-level model
The next step is to model the campaign-level parameters. For our example, these are the campaign-level intercepts λ_j. Again, before we translate the formal notation to `WinBUGS` notation we rewrite () as:

$$\begin{aligned} \lambda_j &\sim N(\mu_\lambda, \sigma_\lambda^2) \\ \mu_\lambda &= \gamma_1 \cdot \text{cq}_j + \gamma_2 \cdot \text{negative}_j \end{aligned}$$

This is expressed almost identically in `WinBUGS`:

```
for (j in 1:J){
  lambda[j] ~ dnorm(mu.lambda[j], tau.lambda)
  mu.lambda[j] <- gamma.1*cq[j] +
                  gamma.2*negative[j]
  }
```

Again, the only difference from the preceding statistical formula is the use of the inverse-variance parameter $\tau_\lambda = 1/\sigma_\lambda^2$.

Issue-level model
We also need to model the issue-level parameters. For our example, these are the issue-level intercepts β_k. Again, before we translate the formal notation to `WinBUGS` notation, we rewrite () as:

$$\begin{aligned} \beta_k &\sim N(\mu_\beta, \sigma_\beta^2) \\ \mu_\beta &= \delta_1 \cdot \text{ownership}_k + \delta_2 \cdot \text{salience}_k \end{aligned}$$

Again, this is expressed almost identically in `WinBUGS`:

```
for (k in 1:K){
  beta[k] ~ dnorm(mu.beta[k], tau.beta)
  mu.beta[k] <- delta.1*ownership[k] +
                delta.2*salience[k]
  }
```

Again, the only difference from the preceding statistical formula is the use of the inverse-variance parameter $\tau_\beta = 1/\sigma_\beta^2$.

Prior distributions
Every parameter in a `WinBUGS` model must be given either an assignment (as is done for the temporary parameter `y.hat[i]` defined within the data model) or a distribution. The parameters `lambda[j]` were given a distribution as part of the campaign-level model, but this still leaves `alpha.0` and `tau.y` from the campaign-issue model and `gamma.1`, `gamma.2` and `tau.lambda` from the campaign-level, and `delta.1`, `delta.2` and `tau.beta` from the issue-level model to be defined.

The specification for these parameters are called *prior distributions* because they must be specified before the model is fit to the data. In the dialogue example, we follow the common practice and use noninformative prior distributions:

```
alpha.0 ~ dnorm(0, .0001)
gamma.1 ~ dnorm(0, .0001)
gamma.2 ~ dnorm(0, .0001)
delta.1 ~ dnorm(0, .0001)
delta.2 ~ dnorm(0, .0001)
```

The regression coefficients γ's and δ's are each given normal prior distributions with mean 0 and standard deviation 100 (thus, they each have an inverse-variance $1/100^2 =$

10^{-4}). This states, roughly, that we expect these coefficients to be in the range $(-100, 100)$, and if the estimates are in this range, the prior distribution is providing very little information in the inference.[5]

We now define the inverse-variances, τ_y, τ_λ, and τ_β in terms of the standard deviation parameters, σ_y, σ_λ, and σ_β, which are each given uniform distributions on the range $(0, 100)$.

```
tau.y <- pow(sigma.y, -2)
sigma.y ~ dunif(0, 100)
tau.lambda <- pow(sigma.lambda, -2)
sigma.lambda ~ dunif(0, 100)
tau.beta <- pow(sigma.beta, -2)
sigma.beta ~ dunif(0, 100)
```

Therefore, the full `WinBUGS` model is written as:

```
model {
  for (i in 1:n){
    y[i] ~ dnorm (y.hat[i], tau.y)
    y.hat[i] <- alpha.0 +
                  lambda[campaign[i]] +
                  beta[issue[i]]
  }
  for (j in 1:J){
    lambda[j] ~ dnorm(mu.lambda[j], tau.lambda)
    mu.lambda[j] <- gamma.1*cq[j] +
                      gamma.2*negative[j]
  }
  for (k in 1:K){
    beta[k] ~ dnorm(mu.beta[k], tau.beta)
    mu.beta[k] <- delta.1*ownership[k] +
                      delta.2*salience[k]
  }
  alpha.0 ~ dnorm(0, .0001)
  gamma.1 ~ dnorm(0, .0001)
  gamma.2 ~ dnorm(0, .0001)
  delta.1 ~ dnorm(0, .0001)
  delta.2 ~ dnorm(0, .0001)
  tau.y <- pow(sigma.y, -2)
  sigma.y ~ dunif(0, 100)
  tau.lambda <- pow(sigma.lambda, -2)
  sigma.lambda ~ dunif(0, 100)
  tau.beta <- pow(sigma.beta, -2)
  sigma.beta ~ dunif(0, 100)
}
```

We can now save the model as *dialogue.bug*. For convenience, save the `.bug` file in the same directory as the data, for example, `"C:/Research/Dialogue"`.

Calling `WinBUGS` from `R`

We load the `R2WinBUGS` package in `R` by typing the following:[6]

```
library("R2WinBUGS")
```

(Actually, we include this and several other `library` calls in our *Rprofile.site* file in the directory `c:\Program Files\R\R-2.4.0\etc\` so that these are called automatically when R is started up.) Now that the `R` environment has been set up to call `WinBUGS`, we execute the following `R` code to set up the data, initial values, and parameters to save for the `WinBUGS` run:

```
dialogue.data <- list ("n", "J", "K", "y",
                       "campaign", "issue",
                       "cq","negative",
                       "ownership", "salience")
dialogue.inits <- function (){
  list (alpha.0=rnorm(1), lambda=rnorm(J),
          beta=rnorm(K), gamma.1=rnorm(1),
          gamma.2=rnorm(1), delta.1=rnorm(1),
          delta.2=rnorm(1), sigma.y=runif(1),
          sigma.lambda=runif(1),
          sigma.beta=runif(1))}
dialogue.parameters <- c ("alpha.0", "lambda",
          "beta", "gamma.1", "gamma.2",
          "delta.1", "delta.2", "sigma.y",
          "sigma.lambda", "sigma.beta")
dialogue <- bugs (dialogue.data,
                  dialogue.inits,
                  dialogue.parameters,
                  "dialogue.bug", n.chains=3,
                  n.iter=500, debug=TRUE)
```

The first argument to the `bugs()` function:

```
dialogue.data <- list ("n", "J", "K", "y",
                           "campaign", "issue",
                           "cq", "negative",
                           "ownership", "salience")
```

lists the data—including the outcome `y`, campaign and issue indicators, `campaign` and `issue`, respectively, and predictors at the campaign-level `cq` and `negative` and at the issue-level `ownership` and `salience`, and the indexing parameters `n`, `J`, and `K`—that are written to a file to be read by `WinBUGS`.[7]

The second argument to the `bugs()` function:

[5] See Gelman and Hill, 2006, pp. 355-356 for a more detailed discussion on noninformative prior distributions.
[6] We recommend saving the call functions to `WinBUGS` in a text editor and saving the file as *dialogue.R*.
[7] We are assuming that the data has been formatted and is currently loaded in `R`.

```
dialogue.inits <- function (){
  list (alpha.0=rnorm(1), lambda=rnorm(J),
          beta=rnorm(K), gamma.1=rnorm(1),
          gamma.2=rnorm(1), delta.1=rnorm(1),
          delta.2=rnorm(1), sigma.y=runif(1),
          sigma.lambda=runif(1),
          sigma.beta=runif(1))}
```

returns a list of the starting values for the algorithm. Within the list are random-number generators, for example, `rnorm(J)` is a vector of length J of random numbers from the $N(0,1)$ distribution, and these random numbers are assigned to λ to start the `WinBUGS` iterations. In this example, we follow our usual practice and assign random numbers from normal distributions for all the parameters - except those constrained to be positive (here, σ_y, σ_λ, and σ_β) to which we assign uniformly distributed random numbers (which, by default in `R`, fall in the range [0,1]).

The third argument to the `bugs()` function:

```
dialogue.parameters <- c ("alpha.0", "lambda",
  "beta", "gamma.1", "gamma.2", "delta.1",
  "delta.2", "sigma.y", "sigma.lambda",
  "sigma.beta")
```

is a vector of the names of the parameters that we want to save from the `WinBUGS` run. For example, the vector λ_j parameters is represented by `"lambda"`.

The last argument to the `bugs()` function:

```
dialogue <- bugs (dialogue.data,
                  dialogue.inits,
                  dialogue.parameters,
                  "dialogue.bug", n.chains=3,
                  n.iter=500, debug=TRUE)
```

takes the data (the variables listed in `"dialogue.data"`) and starting values (as constructed from the function `"dialogue.inits"`) and automatically writes a `WinBUGS` script, calls the model, `"dialogue.bug"` and saves the parameters named in the vector `"dialogue.parameters"` as an `R` object called `"dialogue"`. The object includes all the simulations for easy access in `R`.

Number of sequences and number of iterations
`WinBUGS` uses an iterative algorithm that runs several Markov chains in parallel, each starting with some list of initial values and endlessly wandering through a distribution of parameter estimates. We would like to run the algorithm until the simulations from separate initial values converge to a common distribution. Specifying initial values using random distributions ensures that different chains start at different points.

We assess convergence by checking whether the distributions of the different simulated chains mix; we thus need to simulate *at least 2 chains*. We also need to run the simulations "long enough," although it is generally difficult to know ahead of time how long is necessary. The `bugs()` function is set up to run for `n.iter` iterations and discard the first half of each chain (to lose the influence of the starting values). Thus in the example presented here, `WinBUGS` ran `n.chains=3` sequences, each for `n.iter=500` iterations, with the first 250 from each sequence discarded.[8]

Summary and convergence
From a `WinBUGS` run, you will see means, standard deviations, and quantiles for all the parameters that were saved. You also get, for each parameter, a convergence statistic, $\hat{R}$, and an effective number of independent simulation draws, n_{eff}. We typically monitor convergence using $\hat{R}$, which is call the *potential scale reduction factor*, for each parameter, the possible reduction in the width of its confidence interval, were the simulations to be run forever. Our usual practice is to run simulations until $\hat{R}$ is no greater than 1.1 for all parameters.

We can look at the output in the `WinBUGS` window because `debug` was set to `TRUE`. In other words, setting `debug=TRUE` keeps the `WinBUGS` program open so you can directly view the log file to see a list of possible error messages. We recommend keeping `debug=TRUE` until there are no more errors messages, and then setting `debug=FALSE`.[9] When we close the `WinBUGS` window, `R` resumes and we have the option of displaying the inferences in either graphical or text form. To view the results in graphical form, in the `R` window, we can type

```
plot(dialogue)
```

and inferences for the vectors `lambda` and `beta` and scalars `gamma.1`, `gamma.2`, `delta.1`, `delta.2`, `sigma.y`, `sigma.lambda`, and `sigma.beta` (the parameters included in the `dialogue.parameters` vector that was passed to `WinBUGS`) are displayed in a graphics window.

To view the results in text form, in the `R` window, we can type

```
print(dialogue)
```

Accessing the simulations
We can use the simulations for prediction and uncertainty intervals for any functions of parameters as with the propagation of error in classical regressions. To access the simulations, we must first `attach` them in `R`. In our example,

[8] See Gelman and Hill, 2006, pp. 357 for general advice on how long to run the simulations.
[9] When `R` calls `WinBUGS`, `debug=FALSE` will automatically close the `WinBUGS` window and resume `R`.

we saved the `bugs` output into the `R` object `dialogue`, and we can load in the relevant information with the command,

```
attach.bugs (dialogue)
```

Each variable that was saved in the `bugs` computation now lives as an `R` object, with its 750 simulations (3 chains x 500 chains x last half of the iterations are saved = 750). Each of the scalar parameters, $\alpha_0, \gamma_1, \gamma_2, \delta_1, \delta_2, \sigma_y, \sigma_\lambda$, and σ_β is represented by a vector of length 750, and the vector parameter λ is saved as a 750 x J matrix and β a 750 x K matrix.

We can access the parameters directly. For example, a 90% interval for γ_1 would be computed by,

```
quantile (gamma.1, c(0.05, 0.95))
```

We can also calculate fitted values, residuals and other calculations as well. If you would like to run the following model, you can download the `R` file, `WinBUGS` model, as well as the data from `http://home.gwu.edu/~dkp/tpm.htm`.

We have only highlighted a few of the options available in `R2WinBUGS`. For additional examples, as well as more detail on the various options in `R2WinBUGS`, see Sturtz, Ligges and Gelman (2005) "`R2WinBUGS`: A Package for Running `WinBUGS` from `R`" and Gelman and Hill (2006) *Data Analysis Using Regression and Multilevel/Hierarchical Models.*

Limitations of Bugs/WinBugs/OpenBugs

`Bugs`, as run from `R`, is a great first try for fitting a Bayesian model, and if it works, great. The great advantage of `Bugs` is that it has no problem with nonlinear models and nonstandard parameterizations. Generally, if you can write the model, `Bugs` can fit it. But `Bugs` can be slow, and potentially chokes, on large datasets or datasets with many parameters.

`Bugs` also can have problems with relatively simple linear or generalized linear models when the predictors are highly correlated. In these settings, we can have more success with `R` functions such as `lmer` and `MCMCpack` which are "hard-wired" for particular model specifications. A general difficulty with using `Bugs` is that its code is not easily accessible; thus there is no simple way, for example, to use `lmer` as a way to guide the `Bugs` computations. (One option is to use estimates from `lmer` as a starting point for `Bugs`, but this will not solve the problem of `Bugs` being slow with large datasets or with correlated predictors.)

Another awkwardness with `Bugs` is in writing the models: consider, for example, the two lines needed to reparameterize a variance parameter and set up its prior distribution. Ideally, it should be possible to write a macro or function (as in `R`) which would incorporate the transformation and prior distribution and remove the need for these lines in the program. Similarly, it should be possible to set default prior distributions (for example, $\mathrm{N}(0, 10^2)$ for regression coefficients and $\mathrm{U}(0, 100)$ for variance parameters). For small models, this is not such an issue, but for complicated models with varying slopes and intercepts, and redundant additive and multiplicative parameters (see Gelman and Hill, Sections 19.4–19.6), the "paperwork" required from all the reparameterizations and prior distributions can be so long as to result in Bugs model code that is dozens of lines long, obscuring the underlying statistical model. This can be seen even in the simple model we have presented here.

A related difficulty is the requirement of specifying details in the `R` code. For example, if data or a prior distribution is specified for a parameter that is not in the model, `Bugs` will crash. This might seem to be a reasonable precaution, but when building models it can be helpful to add and remove parameters, in which case it should be allowable to simply feed in all the data and parameters, and let the program figure out what is needed.

It is probably ungracious of us to complain about imperfections in free software. We anticipate, however, that in future years Bugs will be improved to allow more structured computation (for example, parallel vector updating of batches of parameters in a hierarchical model) and will ultimately achieve convergence with more specialized packages such as `MCMCpack`.

References

Gelman, Andrew and Jennifer Hill. 2006. *Data Analysis Using Regression and Multilevel/ Hierarchical Models.* Cambridge University Press.

Kaplan, Noah, David K. Park and Travis Ridout. 2006. "Dialogue in American Political Campaigns? An Examination of Issue Convergence in Candidate Television Advertising." *American Journal of Political Science* 50(3):724736.

R Development Core Team. 2006. *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. ISBN 3-900051-07-0. URL: http://www.R-project.org

Spiegelhalter, D.J., A. Thomas and N.G. Best. 1999. *WinBugs Version 1.4.* Cambridge, UK: MRC Biostatistics Unit.

Sturtz, Sibylle, Uwe Ligges and Andrew Gelman. 2005. "R2WinBUGS: A Package for Running WinBUGS." *Journal of Statistical Software* 12(3):117.

Figure 1: Summary output from example program

```
Inference for Bugs model at "dialogue.bug", fit using winbugs
 3 chains, each with 500 iterations (first 250 discarded)
 n.sims = 750 iterations saved
               mean   sd   2.5%    25%    50%    75%  97.5% Rhat n.eff
alpha.0         6.0  3.8   -2.8    3.7    6.3    8.5   12.5  1.1 37
lambda[1]       0.8  5.5   -9.9   -2.8    1.0    4.3   11.0  1.0 750
lambda[2]      27.0  5.7   16.7   23.0   26.9   30.7   38.8  1.0 300
.
.
.
lambda[64]     -3.2  5.5  -13.4   -6.8   -3.2    0.6    7.2  1.0 750
lambda[65]     25.4  6.3   13.8   21.0   25.4   29.5   39.0  1.0 750
beta[1]        13.8  5.2    4.3   10.1   13.7   17.3   24.1  1.0 140
beta[2]        42.8  5.2   32.9   39.1   42.7   46.2   53.4  1.0 76
.
.
.
beta[42]        9.8  7.8   -5.2    4.5    9.9   15.1   24.7  1.0 98
beta[43]       -1.5 13.7  -26.1  -10.5   -2.4    6.7   27.0  1.0 190
gamma.1         7.4  1.4    4.4    6.5    7.5    8.3   10.0  1.0 600
gamma.2         0.0  0.1   -0.2   -0.1    0.0    0.0    0.2  1.0 160
delta.1        -6.7  4.9  -15.7  -10.0   -6.8   -3.6    3.8  1.1 66
delta.2         0.3  0.5   -0.6    0.0    0.3    0.7    1.4  1.0 160
sigma.y        29.9  0.8   28.5   29.4   30.0   30.4   31.5  1.0 200
sigma.lambda    7.3  1.7    4.0    6.2    7.3    8.4   10.7  1.1 23
sigma.beta     14.5  2.1   10.9   13.0   14.3   15.6   19.2  1.0 210
deviance     9462.1 17.2 9431.0 9450.0 9461.0 9472.0 9500.3  1.1 47

For each parameter, n.eff is a crude measure of effective sample
size, and Rhat is the potential scale reduction factor (at
convergence, Rhat=1).

pD = 63.7 and DIC = 9525.8 (using the rule, pD = Dbar-Dhat) DIC is
an estimate of expected predictive error (lower deviance is better).
```

Data from the Web into R

Simon Jackman
Stanford University
jackman@stanford.edu

More data exists in machine-readable form than ever before, scattered across millions of web pages. As many of *TPM*'s readers will know, a frequently encountered problem is how to get that data from web pages into a form suitable for statistical analysis. I am sure that across the profession the standard tool for this research task remains RAs, keypunching the contents of web pages into Excel spreadsheets and the like: a slow, expensive, and error-prone process. But as many political methodologists know, there is a better way.

In this short note I draw attention to R's data processing capabilities, and in particular, R's "Perl-like" abilities for parsing text. Perl, of course, is a free, open source, programming language that is widely used to support web-based applications, largely because of its text manipulation abilities. The application I present below showcases R's abilities in solving what I think is a fairly common problem: getting quantitative data reported on a web site into one's statistical package and ready for analysis. In so doing I will make use of some very useful functions in R for manipulating text: these include grep, regexpr, sub, and some variants. The discussion below presumes a working familiarity with R: R is freely available via the Comprehensive R Archive Network, with mirror sites all over the world (e.g., http://lib.stat.cmu.edu/R/CRAN), and there are numerous book length introductions to R.[1]

At the outset, I want to stress that there is nothing particularly path breaking about "automating" the process of data acquisition. Many political methodologists are apparently using Perl and/or Python as a matter of course (witness the recent debate about text-processing in Stata on the political methodology e-mail list), and in ways far more sophisticated than what I will sketch below. For example, Jeff Lewis, Mike Herron, Phil Schrodt, Wendy Tam Cho, and Jonathan Wand are just some of the scripting gurus in our ranks who I know to have done some very nimble work acquiring data from web sites. Likewise, the massive Colaresi/Crespin/Monroe/Quinn/Radev et al. project on legislative speech surely has a tremendous amount of scripting to download and process data. And other approaches are possible: e.g., Mike Herron's piece on using XEmacs macros to process text in V13(2) of *TPM*. Moreover, a function like grep in R provide similar functionality as the grep command line utility that started life as part of Unix. Thus, my goal here is surely not to announce a "new" methodology to the profession. Rather, I simply want to underline that using computer programs to parse web pages and to generate data sets is not at all difficult (and indeed, can be done using the same computer program used for data analysis), dramatically increasing the pace of research.

The Application. After the 2006 midterm elections, I wanted to analyze the results across California's 53 congressional districts. The Secretary of State's website presents these data, with frequent updates in the days and weeks after the election, as absentee and provisional ballots are counted. Asking an RA to enter the data manually seems a waste (quite aside from the risk of keypunch error etc.), since the data must be considered "volatile" at least until the final certification of results. Another easily implemented approach is to write a Perl script to parse the HTML containing the election returns, writing the returns to a file that can then be read into a statistics package. Alternatively, one might skip the Perl "middle man", parse the HTML in R directly, and have the data saved in R itself. Since R's text processing functions are very Perl-like, my experience is that the R programs I write to strip data from web sites are not much longer than the equivalent Perl scripts.

Implementation in R. The first step is to pull in the data from the Secretary of State's web site. In this case, all the relevant data appears on one page, and the process of connecting to the site and reading the HTML into R is simple:

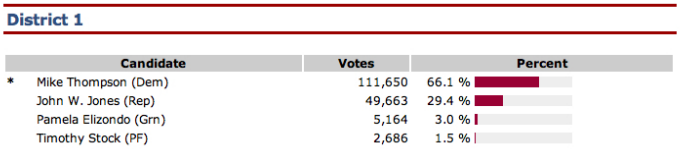
```
url <-(
    "http://vote.ss.ca.gov/Returns/usrep/all.htm")
ssData <- readLines(url)
```

The object ssData is simply a collection of character strings, one for each line in the .htm file. When reading multiple web pages — say, if each district's data was on a separate web page — I take care to sequentially open and close each a separate connection to each page; see help(connection) in R.

The web page is organized district by district, and we will exploit this feature of the layout. Figure 1 shows the displayed results for District 1; our job is to take the HTML that generated that web page and strip out the data into a form amenable for statistical analysis, which we will do using R's functions for manipulating character strings to extract the data for each district.

[1] e.g., P. Dalgaard (2002), *Introductory Statistics with R*, Springer: New York; J. Verzani (2005), *Using R for Introductory Statistics*, Chapman and Hall/CRC: Boca Raton.

Figure 1: Screenshot from California Secretary of State's website, showing results for District 1, as of Monday, November 20, 2006.



District 1

	Candidate	Votes	Percent
*	Mike Thompson (Dem)	111,650	66.1 %
	John W. Jones (Rep)	49,663	29.4 %
	Pamela Elizondo (Grn)	5,164	3.0 %
	Timothy Stock (PF)	2,686	1.5 %

Looking through the data recovered from the site, we see that the string "`District x `" marks the start of the information for district `x`, where `x` takes on the integer values $1, \ldots, 53$. We make a pass over the character data in `ssData`, finding the start of district's data:

```
nDistricts <- 53
districtNames <- paste("District ",
                       1:nDistricts,
                       " ",sep="")
districtLocs <- rep(NA,nDistricts)
for(i in 1:nDistricts){
  districtLocs[i] <- (
               grep(districtNames[i],ssData))
}
districtLocs <- c(districtLocs,length(ssData))
```

The `grep` function looks for its first argument in its second argument, returning the indices of the second argument in which the first argument appears. Thus, the `R` object `districtLocs` is a vector of length 54, with the first 53 entries indexing the start of each district's entry in the web page and the last entry simply indexing the last line of the web page. That is, lines `districtLocs[i]` through `districtLocs[i+1]` of `ssData` hold the data for district `i`. For instance, the HTML that generated the results for CA-1 (Figure 1) starts on line 85 and runs to line 95, with lines 86 (containing the first appearance of the string "`District 1 `") through 90 appearing as follows:

```
<tr class=DistrictBarRow>
<td class=topbar>District 1 </td></tr>
</table> <table class=content>
<tr class=spacerRow><td colspan=10
class=maplink>  </td></tr>
<tr class=contentHeader><td
colspan=2>Candidate</td> <td>Votes</td>
<tdcolspan=2>Percent</td></tr>
<tr class=candRow><td
class=candInc>* </td><td class=candName>Mike Thompson
(Dem)</td><td class=candVotes>111,650</td>
<td class=candPct>66.1 %</td><td class=voteBar>
<img src="/images/pix990033.gif" alt=""
hspace=0 height=11 width=66>
<img src="/images/pixEEEEEE.gif" alt=""
hspace=0 height=11 width=34>
</td></tr>
```

Note that the layout above does not exactly correspond to the layout of the raw HTML. For instance, line 90, containing the information for the Democratic incumbent (and victor) in CA-1, Mike Thompson, is actually one long combination of the seven lines used to display it above. But the good news is that the Secretary of State's office is using good web design standards, "tagging" each piece of information in the HTML file. For instance, the table entries (both entire rows and individual cells) are being tagged with `class` identifiers; e.g., `class=candRow`, `class=candName`, `class=candVotes`, and so on. These class identifiers are there to ensure standardization in the formatting and on-screen look of the results via a CSS (cascading style sheet)[2], but because they have been used throughout the web page they will help us parse the results. That is, if we can extract the data from the appropriate enclosing tag, we're pretty much done. And here is where `R`'s text manipulation functions will come into play. For the sake of argument, lets take line 90 (above) as a test case and refer to it as the `R` character string `text`. In order of appearance, the information we'll extract is

1. the fact that the candidate is an incumbent; note the asterisk appearing immediately after the `<td class=candInc>` tag
2. the candidate's name, appearing immediately after the `<td class=candName>` tag
3. the candidate's party, appearing in parentheses immediately the candidate's name
4. the candidate's vote total, appearing immediately after the `<td class=candVotes` tag

Regular Expressions. Most computing languages support *regular expressions*, which are character strings that describe, match, and parse other character strings, and are almost a programming language in themselves. The widespread popularity of `Perl` is in no small measure due to its efficient implementation of regular expressions. The core character manipulation functions in `R` also use regular expressions, as we will now.

Suppose we have a string like line 90, and we want to extract the contents of the `candName` tag. We can do this via a call to the `sub` function in `R`, which accepts regular expressions. The `R` command

```
sub(x=text,
    pattern=".*class=candName>(.*\\))<.*",
    replacement="\\1")
```

takes `text` as input, and processes it using the regular expression given in `pattern`. The regular expression shown above says to match everything in `text` up through and including the string `class=candName>`, but to designate everything that then appears up until and including a right

[2]See `http://www.w3.org/Style/CSS/`

parenthesis (followed by a < and then the rest of the string), as a variable called \1.

A detailed discussion of how regular expressions work is beyond my scope here,[3] but in this example we see that

- the meta-character "." means "match anything" and the asterisk means match any number of times (including zero times).
- the outer set of enclosing parentheses designate a subset of the regular expression that we can refer to later with the variable names \1, \2, etc (this variable naming convention is a feature of regular expressions, not R, and can't be controlled by the user). In this case we have only one such variable for "back-referencing."
- The right parenthesis is a meta-character, and so when we want to match against it we need to "escape it" by preceding it with the backslash. But when quoting backslashes in R (and many other programming environments) we need a double backslash, which can be a little confusing at first. Likewise, we need to quote the variable \1 back to the sub function in the replacement argument, and so we need a double backslash there as well.

What the sub function is doing here is to (a) match *all* of the string called text; but to (b) refer to a key subset of text as \1; and then (c) replace the matched text (i.e., all of text) with the contents of \1. In the case of line 90, applying the call to sub() as given above produces the output:

```
[1] "Mike Thompson (Dem)"
```

We'll wrap this command up in a function:

```
getName <- function(text){
  sub(x=text,
      pattern=".*class=candName>(.*\\))<.*",
      replacement="\\1")
}
```

We'll also define two more functions: one to extract the actual name from a string with both name and party information, and another to extract the party affiliation:

```
partyFromName <- function(name){
  sub(x=name,
      pattern=".*\\((.*)\\)$",
      replacement="\\1")
}

getOnlyName <- function(name){
  sub(x=name,
      pattern=" \\(.*\\)$",
      replacement="")
}
```

The latter function finds the space after the candidate's name and the party affiliation in parentheses and clobbers both (replacement=""), while the former function looks for something in parentheses where the right parenthesis terminates the enclosing string (the dollar sign $ is how we denotes the end of string in a regular expression).

We define a similar function for extracting vote totals:

```
getVotes <- function(text){
  voteString <- sub(x=text,
pattern=".*class=candVotes>([0-9]{0,3}[,]{0,1}[0-9]{1,3})<.*",
                    replacement="\\1")
  votes <- gsub(x=voteString,pattern=",",
           replacement="")
  as.numeric(votes)
}
```

The regular expression [0-9]{0,3}[,]{0,1}[0-9]{1,3} matches a string that starts with as few as zero or as many as three digits, followed by zero or one commas, followed by at least 1 but no more than 3 digits, and so covers the range of numbers between 1 and 999,999, a range which encompasses for the vote totals in California congressional races. Also note the use of gsub rather than sub, which clobbers all appearances of pattern, rather than just the first; we'll never have more than one comma in a voteString, but we could use this gsub to process a voteString of the form 1,234,567 with multiple commas, if we ever had to deal with vote totals that large.

Finally, we also define a very simple function to search for the asterisk designating a particular candidate as an incumbent:

```
getInc <- function(text){
  regexpr(pattern="class=candInc>\\*",text) != -1
}
```

This function exploits the fact that the function regexpr returns -1 if pattern can't be found in the target string. Thus, this function returns a logical TRUE for incumbents, and otherwise FALSE; n.b., in open seats, this function returns FALSE for all candidates.

With these functions defined, we loop over the California congressional districts, applying the functions to the lines of HTML found to contain the candidate-specific information we're looking to keep from each district. There is also a little bit of extra work to store the output and to aggregate the results for minor parties and independents into an "Other" category.

```
## initialize output to be all missing
## we write into this object below
cvote <- data.frame(matrix(NA,nDistricts,6))
names(cvote) <- c("D","R","Other",
```

[3]There are many excellent on-line references, or see Jeffrey E. F. Friedl's 1997 book, *Mastering Regular Expressions*, (O'Reilly, Cambridge).

```
                "IncParty","DName","RName")

## loop over districts
for(i in 1:nDistricts){
  cat("processing district",i,"\n")

  ## subset this district's data
  thisDistrict <- ssData[districtLocs[i]:districtLocs[i+1]]

  ## subset to lines with candidate-specific data
  candData <- thisDistrict[grep("class=candName",
                            thisDistrict)]

  ## get candName tag
  namesStrings <- sapply(candData,getName)
  ## get Name
  candName <- sapply(namesStrings,getOnlyName)
  ## get Party
  candParty <- sapply(namesStrings,partyFromName)
  ## get votes
  candVotes <- sapply(candData,getVotes)
  ## get incumbency
  candInc <- sapply(candData,getInc)

  ## match parties
  whereParties <- match(parties,candParty)
  if(!is.na(whereParties[1]))
        ## Dem votes, if available
    cvote[i,"D"] <- candVotes[whereParties[1]]
  if(!is.na(whereParties[2]))
        ## Rep votes, if available
    cvote[i,"R"] <- candVotes[whereParties[2]]

  ## are there minor parties (any unmatched parties)?
  matchParties <- match(candParty,parties)
  if(any(is.na(matchParties)))
    cvote[i,"Other"] <- sum(candVotes[is.na(matchParties)],
                            na.rm=TRUE)

  ## which party is the incumbent
  if(any(candInc))
    cvote[i,"IncParty"] <- candParty[candInc]

  if("Dem" %in% candParty)
    cvote[i,"DName"] <- candName[candParty=="Dem"]
            ## dem cand name
  if("Rep" %in% candParty)
    cvote[i,"RName"] <- candName[candParty=="Rep"]
            ## rep cand name
}
```

Note the use of the R function `sapply` to send multiple character strings to our utility functions. This allows us to process an entire district at once. This block of code produces a data frame named `cvote`, containing 53 observations on 6 variables. A little extra computation yields some other useful quantities:

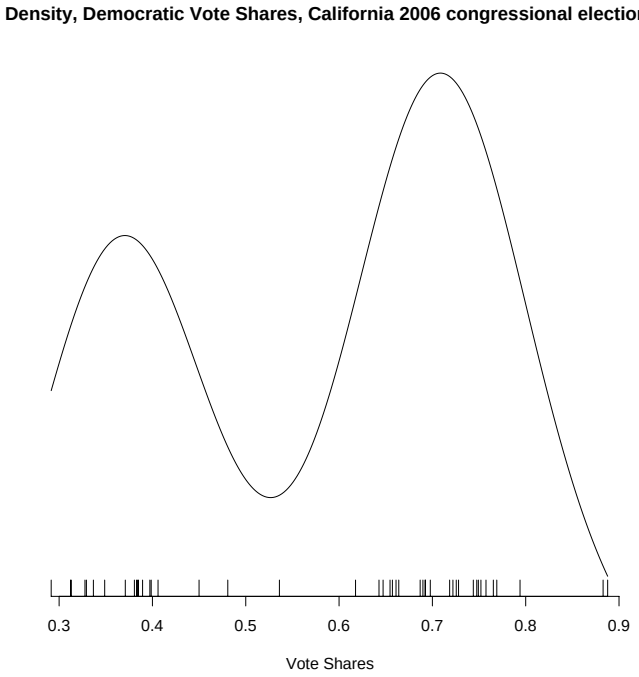
```
x <- cvote$D/(cvote$D+cvote$R)
cvote$contested <- !is.na(x)
cvote$open <- is.na(cvote$IncParty)
```

The first three records of this data frame appear in table 1.

The R program that generates this data frame is available for downloading at `http://jackman.stanford.edu/pscl/ca2006.R`.

Putting the Data to Work. Now that we've read the data from the Secretary of State's web site – quickly and reliably – we can move on to analysis. In 8 seats the incumbent faced no major party opposition: 7 of these seats are held by Democrats and of these, 5 are in Los Angeles. Of the 45 contested seats, Democrats picked up one seat from the Republicans (CA-11, in the Central Valley east of the San Francisco Bay Area), and lost none. Democrats won 27 of the 45 contested seats (60%), and so the Californian delegation in the 110th House consists of 34 Democrats and 19 Republicans. Across the 45 contested seats, the Democratic share of the two party vote ranged from 29.2% (in CA-22, taking in San Luis Obispo and Kern counties, and much of Bakersfield) to 88.8% (in CA-9, taking in Berkeley and Oakland), and averaged 57.7%.

Figure 2: Vote Share Densities



The most politically interesting fact about Californian congressional politics is the lack of competitive seats. Of the 45 contested seats, the closest result was CA-4, where the Republican incumbent won with 52% of two-party votes; the next closest seat was the seat that changed hands (CA-11), where the Democrat won with 53.2% of the vote. These are the only 2 of the 45 contested seats to be decided by margins of less than 5%; CA-50 was won by a Republican with 55% of the two-party votes, and the next closest result was Mary Bono's 60-40 win in CA-45. The closest result for a Democrat incumbent running for re-election was the 62.2% result recorded by Loretta Sanchez in CA-47 (centered on

Table 1: Example of data scraped from the California Secretary of State's web pages after processing

	D	R	Other	IncParty	DName	RName	contested	open
1	111650	49663	7850	Dem	Mike Thompson	John W. Jones	TRUE	FALSE
2	54829	108002	5613	Rep	Arjinderpal Sekhon	Wally Herger	TRUE	FALSE
3	82293	129505	5715	Rep	William E. Durston	Daniel E. Lungren	TRUE	FALSE

Orange County).

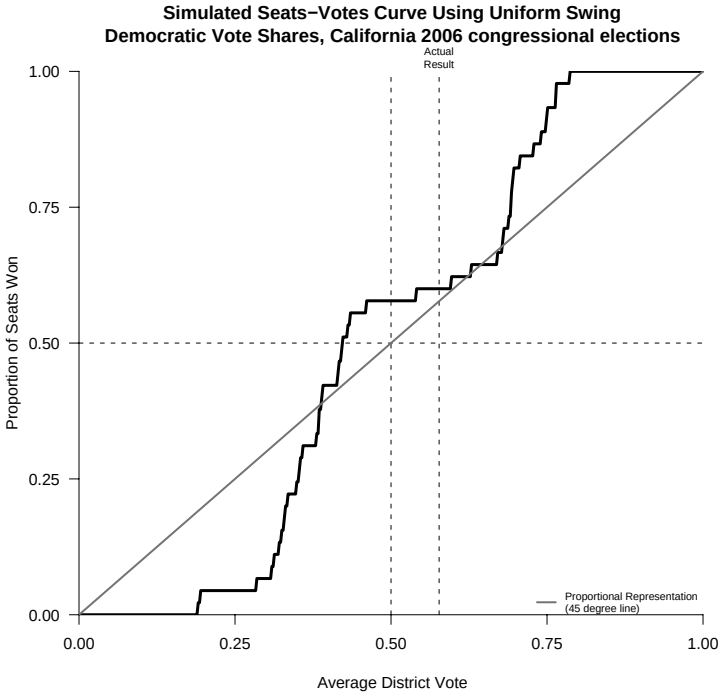
Figure 2 presents a density plot of Democratic two-party vote shares in the 45 contested seats. The bimodality of the density stems from the absence of competitive seats, as discussed above. Figure 3 displays a simulated seats-votes curve generated by applying various levels of uniform swing to the 2006 results, tracing out a series of average district vote shares and seat shares (while the uniform swing assumption is implausible, it remains a simple and convenient way of inducing a seats-votes curve from a set of vote shares). The relative "flatness" or "unresponsiveness" of the curve in the neighborhood of 50-50 again stems from the dearth of competitive seats. In addition to the lack of responsiveness, there is some evidence of partisan bias: Democrats are estimated to win 58% of the contested seats with just 50% of the average district vote. Incidentally, these graphs were generated using the `seatsVotes` classes and methods in the `pscl` package for `R`, developed in the Political Science Computational Laboratory at Stanford and available from any CRAN: in `R`, after loading the `pscl` package, enter `help(plot.seatsVotes)` to see how these graphs were created (the 2006 California data are the working example in the documentation for the `seatsVotes` class).

Of course, the lack of responsiveness in California's congressional districts has its roots in the redistricting following the 2000 Census. The account in the 2006 edition of the *Alamanac of American Politics* is worth reading; I reproduce a brief section of it here:

> The key player for Democrats was Michael Berman, brother of Congressman Howard Berman and a redistricting expert who had worked with Phil Burton on redistricting in the 1970s and 1980s. He came out of retirement and was hired as a redistricting consultant by House and state Senate Democrats, at $20,000 per incumbent. As Congresswoman Loretta Sanchez said, "Twenty thousand is nothing to keep your seat. I spend $2 million every election. If my colleagues are smart, they'll pay their $20,000 and Michael will draw the district they can win in." (p. 155)

As the data show, Sanchez's 62.2% result was the closest a Democratic incumbent got to losing in 2006; notwithstanding that 2006 was a very good election for Democrats, it would seem that Sanchez and her colleagues got their money's worth.

Figure 3: Seats-Votes Curve



Back to the Future. Parsing web sites as I have done here will most likely become a "legacy" skill in the years ahead. Slowly but surely, data is increasingly being presented on-line via XML (Extensible Markup Language), a data-description language designed to facilitate data exchange across computers. Parsing XML is thus much easier than stripping data from HTML, and already there are tools in `R` for traversing XML trees, and extracting data. Far less programming effort will be required to get data in a form suitable for statistical analysis as XML replaces HTML as a method for presenting data on the web. Until then, "hacks" like the one described in this note will likely remain part and parcel of how we (and our RAs) acquire data.

Book Review

Review of Essential Mathematics for Political and Social Research, by Jeff Gill

Ryan T. Moore
Harvard University
rtmoore@fas.harvard.edu

Introduction

Jeff Gill's new *Essential Mathematics for Political and Social Research* (EMPSR) is one of few texts developed for political scientists, by a political scientist, that motivates the range of topics taught during a typical first graduate course in quantitative methods. Different programs involve different first courses – semester courses may focus on probability and mathematical statistics, data analysis and regression, or research methods broadly construed; one- or two-week pre-semester "math camps" also abound. EMPSR speaks to all of these settings, but focuses on short prefresher courses. After some general comments about usage in various settings, I trace EMPSR's contents more systematically, and compare EMPSR to other popular introductory materials.

EMPSR could serve as the primary or secondary text in many introductory settings. The range of topics suffices to fill a semester-long course on prerequisite mathematics, especially in departments where new graduate students may arrive with little quantitative background. Departments that begin graduate training with data analysis and regression could view the text as a prerequisite to program entry, and encourage summer study by their incoming classes. The wealth of applied examples could inform discussion in courses on general social science research methods, either at the graduate or undergraduate level. Given their breadth, however, such courses may not invest the time required for mastery of most mathematical skills EMPSR addresses. Math camp courses should seriously consider using EMPSR as their primary text, and Gill offers several templates for doing so. My experience with EMPSR is largely as an instructor of one such course; thus, although EMPSR can contribute in several settings, a math camp perspective dominates this review.

The students' diverse set of backgrounds, aptitudes, and interests creates much of the difficulty in teaching a successful first methods course in political science. In many programs, future philosophers sit next to future statisticians. This diversity particularly complicates a math camp textbook's two-fold charge: to enable all students to feel reasonably well-prepared on the first day of a term-time course, and to remain relevant beyond the first week of everyone's graduate career. EMPSR achieves both, by starting with extremely elementary material, incorporating over 100 examples (most drawn from actual, published social research), and still touching on topics that may not resurface until the third or fourth graduate course in statistical methods.

The numerous examples provide instructors with a ready-made answer to a common question of new graduate students: "Why am I starting political science grad school with pure math?" Namely, "To learn useful tools for answering interesting substantive questions in politics." Although the title suggests that EMPSR might be purely a primer in basic mathematics, the text highlights social science applications. Prominent examples follow in the next section.

New graduate students and those anticipating graduate study should find comfort in EMPSR's approachable style, its warnings of common confusions, and in the connections it draws between the mathematics and the substantive questions of interest. EMPSR's chapters begin with explicitly-stated objectives, helping to socialize new political scientists into the academic discipline. Reference tables, intuitive explanations, chapter lists of new terminology, and the relevance and volume of the topics all imply that EMPSR will be helpful in the early days, but also a well-worn text by the time students finish their degrees.

Concepts and Examples

EMPSR begins with the most elementary mathematical topics required for quantitative research: arithmetic, notation, and functions. However, by page 5 students have already encountered something that political scientists will recognize as being of significant value: Riker and Ordeshook (1968)s model for voter utility, $R = PB - C$. Although simple, this model is still widely discussed; in 2006 alone it has appeared in the APSR, AJPS, BJPS, and JOP. The second chapter covers analytic geometry and includes the most relevant topics from a high school trigonometry course. Political-scientific examples include parabolic presidential approval and elliptical voting preference models.

Chapters 3 and 4 straightforwardly introduce linear

algebra. Chapter 3 defines vectors, matrices, operations, and related properties. Central topics in Chapter 4 include the geometry of matrices, the determinant, eigenvalues, quadratic forms, and inverses. Example 4.6, a two-page exercise in estimating OLS regression parameters, provides an introduction to a ubiquitous application of matrix algebra to political science data.

Scalar and vector calculus fill Chapters 5 and 6, which include traditional definitions and applications of limits, derivatives, and integrals. Example 5.9 applies scalar calculus to another mainstay of political science, the Median Voter Theorem of Black (1958). EMPSR highlights often-used skills such as extrema- and root-finding, multiple integration, and vector differentiation. Foundations such as the gradient and Hessian, Lagrange multipliers, and constrained optimization appear.

The pre-statistics section of EMPSR begins with probability theory in Chapter 7. True to his Bayesian roots, Gill opens the chapter with a discussion of subjective versus objective probability. This chapter includes set-theoretic definitions and properties, probability functions, conditional probability, Bayes' rule, Simpson's paradox, and independence. EMPSR also demonstrates odds, a topic that political scientists often encounter, but may have less prior exposure to than some other social researchers (like epidemiologists).

Chapter 8 covers random variables, and includes a 4-page discussion of levels of measurement. EMPSR introduces familiar distributional families as models for social data-generating processes: Bernoulli and binomial data, Poisson counts, and uniform, exponential, gamma, and normal Gaussian phenomena all appear. This chapter's wealth of applied modeling examples includes Supreme Court decisions, legislative bill passage, strategic alliance formation, incidence of war, income distributions, probit analysis of vote choice, and the presence of women in US state legislatures. The last of these features a quantile-quantile plot, thus giving a welcome introduction to model fit diagnostics.

Only after these examples does EMPSR cover measures of central tendency and spread. This ordering is consistent with texts like Rice (1995), but can lead to discussion of these measures that precedes their formal definition. EMPSR succeeds more than Rice in minimizing such discussion, but does not avoid it entirely as do treatments like Purves (1991). Other topics include summary statistics' breakdown points, correlation, expected value inequalities, and distributions' moments and central moments. A die-rolling example illustrates expected value, as does an extended sequence of calculations derived from craps bets.

The last chapter is somewhat unexpected. Here, Gill introduces Markov chains, a topic that most political scientists might not encounter until they take a course in Bayesian modeling or data analysis. The chapter elucidates major chain concepts (periodicity, homogeneity, irreducibility, reversibility) and state characteristics (recurrent, absorbing, transient, closed). Gill demonstrates the utility of Markov chains as descriptive of social processes in their own right, but the underlying motivation may be to lay foundations for future Bayesian work.

Comparisons and Conclusions

There are many candidate materials for political science math camps. Instructor's notes, Simon and Blume (1994), Morgan (1997), and Hagle (1996) appear particularly common. EMPSR contrasts with Hagle and Morgan in two primary ways. First, EMPSR's scope is broader. The last third of EMPSR covers probability and statistics material omitted from the other two texts, for example. Second, the exercises and examples in EMPSR are significantly more applied than those of Hagle or Morgan.

One's preference for adopting EMPSR may hinge on whether one prefers the clear lines of fundamental skills repetition or the more thought-provoking and interpretive fuzziness of examples of political science research. To illustrate the difference, consider the problem sets on differentiation. Hagle's includes six consecutive questions instructing simply, "Find the derivatives of the following functions." Meanwhile, EMPSR sandwiches its one such question between exercises using published political research on suburban demographics and the siting of US county seats. Exercises in Hagle, Morgan, and Simon and Blume are also split into relatively small, homogeneous sections, while those of EMPSR are collected at chapters' ends. The former design encourages rote repetition, while the latter can obscure basic skills, but more accurately reflects the problems and choices students of methodology will face.

In my view, occasional over-complexity is the weakness of EMPSR. While the variety of examples is generally a strength of EMPSR, sometimes there is too much of a good thing. For example, in demonstrating inner products and cross products, Examples 3.6 and 3.8 use the same definitions for 1×3 vectors u and v, but Example 3.7 uses different ones. Using consistent definitions would simplify the matter and allow readers to focus on understanding the algebra. For a math camp, EMPSR's problem sets are long and include some potentially intimidating problems. Selecting exercises to assign will require judicious consideration of one's audience. Also, at this time, an answer key is still in development. Until its release, instructors may have to supply their own solutions.

On the whole, EMPSR succeeds. Its range and depth of topics form appropriate standards for incoming and continuing political science graduate students. Its constant attention to published research introduces budding professionals to exactly how and why learning mathematics is an important first step.

References

Black, Duncan. 1958. *Theory of Committees and Elections*. Cambridge: Cambridge U Press.

Freedman, David, Robert Pisani and Roger Purves. 1991. *Statistics*. New York: Norton.

Hagle, Timothy M. 1996. *Basic Math for Social Scientists*. Thousand Oaks: Sage.

Morgan, Frank. 1997. *Calculus Lite*. Wellesley, MA: A.K. Peters.

Rice, John A. 1995. *Mathematical Statistics and Data Analysis*. Belmont, CA: Duxbury Press.

Riker, William H. and P.C. Ordeshook. 1968. "A Theory of the Calculus of Voting." *American Political Science Review* 62 (1):2542.

Simon, Carl P. and Lawrence Blume. 1994. *Mathematics for Economists*. New York: W.W. Norton.

Section Activities

A note from our Section President

I would like to offer a hardy thank you to Adam Berinsky, Michael Herron, and Jeff Lewis for their hard work and dedication in producing *The Political Methodologist* for the past three years! It is an invaluable communication tool for the section. Please join me in welcoming the new *TPM* editors, Paul Kellstedt, David Peterson, and Guy Whitten. We look forward to continued success of *TPM* under the guidance of the new Texas A&M editors.

Nominations for the second annual John T. Williams Dissertation Prize are being solicited. The prize is given in recognition of John T. Williams' contribution to graduate training and is for the best dissertation proposal in the area of political methodology. Proposals using quantitative or qualitative methods are welcomed and should follow the National Science Foundation length and format guidelines. Members of the committee are John Aldrich (chair), Tse-Min Lin, and Michael Colaresi. Materials should be sent to the John Aldrich at `aldrich@duke.edu`.

The 24th Annual Summer Meeting of the Society for Political Methodology will be held at Pennsylvania State University, July 19-21. The hosts, Suzanna DeBoef and Burt Monroe, have information about the conference available on the conference website at: `http://polmeth.psu.edu/`. The past success and popularity of the meetings have led the Society's membership to support the recommendation of the Long Range Planning Committee by implementing an alternative model for the meeting to accommodate increased demand. In an effort to extend participation, the meeting size is growing substantially. With increased size, however, come some inevitable changes. The basic program format and the venerated graduate student poster session will remain. The host institution will be providing breakfast and lunch for the participants throughout the conference and will host a dinner and a reception in 2007. All other expenses (notably, hotel accommodations and remaining dinners) will be covered by attendees. Registration applications are available at: `http://polmeth.wustl.edu/methods2007/register/`. We thank the program committee, Rebecca Morton (chair), Suzanna DeBoef, Burt Monroe, Kevin Quinn, and Jake Bowers for their hard work and dedication in bringing together the meeting. The National Science Foundation, in conjunction with Penn State University, will continue to support 35 graduate students through a competitive process. We thank the Graduate Student Selection Committee for their work as well. The committee includes Dan Wood (chair), Michele Claiborne, David Darmofal, and Kevin Clarke.

We now have over sixteen active committees. A full listing of all the committee members and a list of their charges is available on the Political Methodology website: `http://polmeth.wustl.edu/society.php`. We thank for Andrew Martin and Stephen Haptonstahl at Washington University for their work on the website. They provide ths excellent service for the section gratis. I want to highlight one new committee, the Undergraduate and Graduate Methodology Committee, which is chaired by Lonna Atkeson. Other committee members include Garrett Glasgow, Paul Gronke, Dean Lacy, and Alan Zuckerman. Agenda items include: 1) developing best practices for departments and students in order to be prepared for graduate school in political science; 2) increase the availability of methods syllabi; 3) sponsor panels at the APSA Teaching and Learning Conference on undergraduate and graduate methods; 4) brainstorm about what the section should be doing for it's members who are at schools where there is significant emphasis on undergraduates and teaching; 5) explore best practices for interdisciplinary methods training for graduate students. Please contact them if you want to get involved or have an idea to share with them.

Best wishes,

Jan Box-Steffensmeier
The Ohio State University

[This page Intentionally blank]

THE POLITICAL METHODOLOGIST
Department of Political Science
Massachusetts Institute of Technology
Cambridge, MA 02139

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.