

# JUDGEIT II: A Program for Evaluating Electoral Systems and Redistricting Plans<sup>1</sup>

Andrew Gelman

Gary King<sup>2</sup>

Andrew C. Thomas<sup>3</sup>

Version 1.3.3  
August 26, 2008

<sup>1</sup>Available from CRAN (<http://cran.r-project.org/>) and <http://GKing.Harvard.edu/judgeit>.

<sup>2</sup>David Florence Professor of Government, Harvard University (Institute for Quantitative Social Science, 1737 Cambridge Street, Harvard University, Cambridge MA 02138; <http://GKing.Harvard.Edu>, [King@Harvard.Edu](mailto:King@Harvard.Edu), (617) 495-2027).

<sup>3</sup>PhD Candidate, Department of Statistics, and Graduate Associate, Institute for Quantitative Social Science, Harvard University. <http://www.fas.harvard.edu/~acthomas/>; [acthomas@fas.harvard.edu](mailto:acthomas@fas.harvard.edu).

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Installation Instructions and Requirements</b>	<b>2</b>
<b>3</b>	<b>Usage Overview</b>	<b>3</b>
<b>4</b>	<b>Formatting Data Sets</b>	<b>3</b>
<b>5</b>	<b>Included Data Sets</b>	<b>3</b>
<b>6</b>	<b>Step 1: Create a JudgeIt object</b>	<b>4</b>
<b>7</b>	<b>Step 2: Conduct Analyses</b>	<b>6</b>
<b>8</b>	<b>Further JudgeIt output options</b>	<b>8</b>
<b>9</b>	<b>Accessing Raw judgeit.object Data</b>	<b>9</b>
<b>A</b>	<b>The Underlying Method</b>	<b>9</b>
<b>B</b>	<b>Demonstrations</b>	<b>11</b>

# 1 Introduction

JUDGEIT II brings the analytical routines of the original version of JUDGEIT to the R Project for Statistical Computing, while also greatly simplifying its interface and control system.<sup>1</sup> The methods implemented in this software were developed in Gelman and King (1990*b,a*); ?, 1994); King and Gelman (1991); Gelman, Katz and King (2004).

JUDGEIT allows a user to construct a model of a two-party election system over multiple election cycles, derive quantities of interest about the system through statistical estimation and simulation, and produce output summary statistics and graphical plots of those quantities. Some of the quantities of interest are based on partisan symmetry as a standard of fairness in legislative redistricting, such as *partisan bias* as the deviation from fairness and *electoral responsiveness* which indexes how party control of legislative seats responds to changes in a party's success at the polls even in a fair system. (A uniform consensus has existed in the academic literature since at least King and Browning (1987) on partisan symmetry as a standard for fairness, and even the U.S. Supreme Court now appears to agree; see Grofman and King 2007.) JUDGEIT also estimates and graphs seats-votes curves, make specific vote and seat predictions for individual districts, and calculate numerous other relevant statistics.

The program can evaluate electoral systems in three general situations:

1. When an election already has taken place,
2. When an election has not been held yet but a new redistricting plan (or plans) has been proposed or implemented, and
3. When you wish to assess what an election would have been like if held under certain specified counterfactual conditions (such as if no minority districts had been drawn, or term limitations had prevented incumbents from running for reelection).

For bias, responsiveness, seats-votes curves, and virtually every other estimate, JUDGEIT provides quantitative estimates of uncertainty (i.e., standard errors or confidence intervals). This is recognized as essential by social scientists and even the Supreme Court (see *Castaneda v. Partida*, 430 U.S. 482, 1977, and *Hazelwood School District v. United States*, 433 U.S. 299, 1977).

## 2 Installation Instructions and Requirements

As this version of JUDGEIT has been built as a package for the mathematical software R, it is necessary to install this software before beginning.

The latest version of R is available for free from <http://www.r-project.org>. After installation, retrieve the JUDGEIT package through the following command:

```
> install.packages("JudgeIt")
```

To use the commands within, type

```
> library(JudgeIt)
```

JUDGEIT requires the R package `mvtnorm` to function properly, and will load it automatically when needed.

### Case Syntax

In all instances, the stylized JUDGEIT refers to the software; the mixed case **JudgeIt** refers explicitly to the library name; and the lower case `judgeit` is used for the primary analysis function and any contained terms, such as `judgeit.object`.

---

<sup>1</sup>The original version of JudgeIt, written by Gary King and Andrew Gelman in versions available for DOS and in the language Gauss, is still available at <http://gking.harvard.edu/judgeit/judgeitI>.

### 3 Usage Overview

JUDGEIT can conduct three types of analyses:

1. Evaluation, wherein the information gathered from one year's elections is applied to itself. This is done to suggest the underlying structure of an electoral system.
2. Prediction, which takes a set of observed covariates and predicts the outcome of an election. This can be done with or without the previous election's results and covariates included explicitly.
3. Counterfactual analysis, which estimates the results if the election had been run under different circumstances (i.e. with different predictors.)

Two steps are required to perform a JUDGEIT analysis: election data must be loaded into a JUDGEIT object, and a choice of analysis must be specified along with its required options.

### 4 Formatting Data Sets

JUDGEIT requires each election to be reported in an object of type `data.frame`. This is natural if your data is in a comma-separated or tab-delimited file, since operations like

```
data <- read.csv("ohio.csv")
data2 <- read.table("cali98.txt")
```

will automatically put these tables into the data frame format. It is important to make sure that each variable is labeled before using it in the function `judgeit`, either by making the first line of the data file contain variable names, or by assigning them afterwards. For example:

```
data <- read.csv("ohio.csv",header=FALSE)
colnames(data) <- c("district","vote","inc")
```

Since most JUDGEIT analyses will involve a series of consecutive elections, `judgeit` accepts such a series as an object of type `list`. Suppose we have three elections, say Ohio's State Legislature in 1992, 1994 and 1996, and that each election is stored in a data frame as above. To integrate the separate data frames, use the command `elections <- list(oh1992,oh1994,oh1996)` which will create an object named "elections". Each year's election data *must* use the same variable names, so that JUDGEIT can recognize the same quantity across different elections.

### 5 Included Data Sets

JUDGEIT comes with data set ICSPR 6311, coded as `house6311`, containing the results for the U.S. House of Representatives from 1896 through 1992. It is formatted as a list of 49 data frames, each named for their election year, and containing six vectors:

- STATE, a numerical ICPSR state code for the state containing the district;
- DIST, a numerical indicator for the number of the district within the state;
- INC, an incumbency indicator,  $-1$  for Republican and  $1$  for Democrat;
- VOTE, the share of the vote received by the Democratic candidate;
- TURNOUT, the number of votes for the two candidates combined, and
- DELSOUTH, an indicator for whether the district is located in the South.

## 6 Step 1: Create a JudgeIt object

Now that there is an object containing the elections we wish to analyze, we can load this object into `JUDGEIT`. To do this:

```
judg.obj <-  
  judgeit(model.formula=vote ~ predictor1 + predictor2 + some.function(predictor3),  
          data=elections, vote.formula=cbind(turnout,eligible)~seats, ...) }
```

where the ellipsis “...” represents other possible commands used by the R command `model.frame`. The user may also apply a function to a variable in the data set as seen by `some.function(...)`

### Model Formula

The model formula, `model.formula`, specifies the system’s outcome variable — the *proportion* of the two-party vote received by the candidate for a specified party — and any desired predictors, such as incumbency status. The predictor names must be the same as the variable names within the data list (referred to here as `elections`.)

Functions can also be applied to the variables within the formula. For example, if we wish to create an indicator for whether a win is decisive (for example, the winner more than triples the loser’s vote share), we first create such a function,

```
some.function <- function(arg) -1*(arg<0.25)+1*(arg>0.75)
```

and then use it in the formula. For example, with data set `house6311` this formula works: `model.formula=VOTE~INC+some.function(VOTE)`.

### Voter Formula

We use the voter formula, `vote.formula`, to indicate the *number* of actual voters (also known as the turnout), the number of eligible voters, and/or the number of seats each district elects to the main assembly. If this is omitted, the number of seats is assumed to be one per district and the turnout is assumed to be equal in all districts.

For example,

- `vote.formula=cbind(turnout,eligible)~seats` specifies all three quantities.
- `vote.formula=cbind(turnout,eligible)~1` specifies turnout and eligible voters, and one seat per district.
- `vote.formula=turnout~1` specifies turnout. The number of eligible voters is ignored, and seats are specified at one per district.
- `vote.formula=~1` is the default option.

### Specifying Populations of Groups

The routine `voting.power` allows the user to investigate the probability that a member of a particular group would have on changing an election result. To specify the groups present in the system, use the formula option

```
pop.groups=~farmers+white.collars+clowns+statisticians
```

substituting whichever groups desired for the analysis. (The tilde ~ is necessary!)

## Other Options

### Selecting a data subset

It is possible to choose a subset of the data for analysis in each year. For example, the data set `house6311` contains an indicator `DELSOUTH`, which is 1 if a district is located in the south and 0 otherwise. To perform analyses on non-southern districts only, initialize a `JudgeIt` object (in this example, `jud.ob`) with the expression

```
jud.ob <- judgeit(...,subset=DELSOUTH==0,data=house6311)
```

### Automatic Use of Previous Year's Results

The results of a previous year's election is often among the best predictors of a current contest, and they are included by default. To remove this option, include the option `use.last.votes=FALSE` in your `judgeit` statement.

### Redistricting

At some point during a state's existence, the electoral map is redrawn to adjust for changes in the demographics of the districts. Specifying whether the districts in one election are identical to the previous one are vital to an accurate representation of the system.

By default, `judgeit` assumes that if two consecutive years have the same number of districts in the map, then no redistricting has taken place.

To set a redistricting occurrence manually, `judgeit` accepts as an option `same.districts`, which indicates whether the previous election used the same districting plan. In many states in the American system, the easiest way to do this is to have a variable specifying the election years and note whether the year ends in 2 (`year %% 10 == 2`). For other situations, you will need to add a specially coded variable to the data frame and set `same.districts` to this value.

### Uncontested Districts

`JUDGEIT` includes several routines to deal with uncontested districts. By default `JUDGEIT` assumes that a vote proportion below 0.05 or above 0.95 indicates an uncontested district and are fixed to default alternate values; these levels can be changes using the options `uncontested.low` and `uncontested.high`.

The option `uncontested.method` indicates which method is to be used to deal with districts tagged as uncontested:

- `"default"`, which replaces uncontested values with assumed vote totals specified by `uncontested.low.new` and `uncontested.high.new` (default values are 25% and 75%);
- `"impute"`, which uses the behaviour of contested districts to estimate unknown vote shares,
- `"remove"`, which simply eliminates uncontested districts from the analysis, or
- `"nochange"` which leaves the vote variable as entered.

### Simulation Parameters

`JUDGEIT` analyzes election systems by simulating a number of hypothetical draws of the election of interest, and computing quantities of interest from these simulations. The default number of simulated elections is 201; to change this, include the option `simulations` in the `judgeit` command.

## Applying District Weights: Some Count More than Others

Districts may have different influence than others when determining the parameters of a system. To reflect this, we can assign a `weight` parameter to command `judgeit` :

- `weight=constant` (default) in which all districts have equal weight;
- `weight=turnout`, corresponding to the two-party votes cast in each district;
- `weight=eligible.voters`, corresponding to the total number of eligible voters in each district, or
- `weight=seats`, the number of seats in each district.

This will have two distinct impacts on the analysis. First, in the linear modelling, each district will have a variance equal to a constant ( $\sigma^2$ ) times the specified weight. Second, the mean vote will be calculated with respect to this weight; a `constant` weight will yield the average district vote, while setting the weight according to `turnout` will yield the grand average or “popular” vote.

## 7 Step 2: Conduct Analyses

Once an election system has been loaded into a `JUDGEIT` object, quantities of interest can be simulated. There are three types of analyses than can be performed:

### Evaluation

The default setting of `JUDGEIT` for analysis, evaluation mode examines an election under the actual conditions that took place. Under these conditions, the systematic error component in simulation (the amount estimated by the parameter  $\lambda$ ) is identical, and total error estimates are therefore smaller.

By default, all substantive outputs from `judgeit` use evaluation mode. For example, using a previously created `JUDGEIT` object `judgeit.object`, we can obtain a seats-votes curve using the command

```
plot(seats(judgeit.object))
```

### Prediction

Prediction takes the information gleaned from one election and uses it to forecast the results of a new election. In this case, since the new election hasn’t taken place, the systematic error component is inestimable, and the total error component is generated.

To enable predictive mode, add the option `predict=TRUE` to a `judgeit` analysis command. To specify new covariates for this election, add the option `new.covariates` to specify the new conditions, or use `new.covariate.matrix` to draw a new electoral map entirely.

### Counterfactual Evaluation

This option supposes what might have happened if an election were rerun under different circumstances. The systematic error component is once again held constant to reflect that this is the same “voting apparatus” as was used originally.

To enable a counterfactual analysis, specify new covariates by adding the option `new.covariates` and changing the relevant values.

## Using `new.covariates` to recode data

In order to recode the data for counterfactual analysis or prediction, the option `new.predictors` must be added to a `judgeit` analysis command.

As an example, suppose we wish to find out what were happened if term limits were imposed and no incumbent could run for re-election, and that the incumbency indicator is labelled `INC`. The option

```
new.covariates=list("INC",0)
```

would instruct each routine to use this counterfactual data.

Suppose it came to pass that, instead, a group of representatives were forced to resign shortly before the election. If the new incumbency indicators are stored in a variable `new.incs`,

```
new.covariates=list("INC",new.incs)
```

would make that substitution. Make sure, however, that the substitution variable has the same number of districts as the model for that year; to confirm this, run the command

```
summary(judgeit.object,year).
```

## **district.report: Examine the estimation**

The output of this routine is a reckoning of each district, including the observed and hypothetical vote shares, the standard deviation of the hypothetical estimate, and the probability that a voter in this district would be able to change the outcome of the entire election by reversing their vote.

To run a district report, run the command

```
district.report(judgeit.object).
```

By default the command will run a report for the last year in the system; to specify a year to analyze, add the option `year=1992` substituting whichever year desired.

## **seats: Given a vote share, determine the seat share**

This is the basic tool used to determine the relationship between votes and seats in an electoral system.

**Example:** `seats(j.ob)` will produce estimations and errors for the fraction of seats received using the default options; in particular, in the final election in the system, at various values around the observed average vote.

### **Plotting a seats-votes curve**

If the output in a `JUDGEIT` object was produced by `seats`, the `plot()` command will recognize this and produce a seats-votes curve.

**Example:** `plot(seats(j.ob))` will plot the seats-votes curve for values in the last year in the system.

## **seats.with.prob: Determine the fraction of seats whose win probabilities lie within a chosen range.**

**Example:** `seats.with.prob(j.ob)` will produce estimations and errors for the fraction of seats received using the default options: in particular, in the final election in the system, with various average vote values between 0.45 and 0.55.

**results.prob:** Given a vote share or deviation, determine the probability that the seat share is within a particular margin

**Example:** `results.prob(j.ob)` does the default: For the vote as it is, what is the probability that the seat share will be between the default range of 0.45 to 0.55?

**Example:**

```
results.prob(j.ob,shift.in.votes=-0.05)
```

determines what the probability is that the seat share will be between the default range of 0.45 to 0.55, given that the resulting vote share was 5 points lower than actually experienced.

**Example:**

```
results.prob(j.ob,vote.range=c(0,0.5),mean.votes=0.5)
```

determines what the probability is that the seat share for Party 1 will be below 50 percent, given that the resulting vote share was 50 percent for each party.

**bias.resp, or svsum:** Determine partisan bias and responsiveness

The output is a table containing the estimates and errors for four quantities: Partisan bias at an even vote, both instantaneous and averaged over a 10-point swing, and responsiveness at the midpoint and the observed vote percentage.

**Example:**

```
bias.resp(j.ob,year=1976)
```

will output those properties for the election held in 1976. (This assumes that the judgeit object `j.ob` is specified to include the calendar years of each election.)

**votes.for.result, or winvote:** Given a probability of victory, determine the total vote share required to attain it

The reverse procedure of `results.prob`, if the seat share was between 0.5 and 1.

**Example:**

```
votes.for.result(j.ob,prob.win=0.7)
```

outputs the expected vote percentage needed for a 70% chance of victory.

**voting.power:** Determine the power that distinct groups have in changing an election result

Suppose there exists a distinct number of groups spread across the districts of an electoral map (by race, language, or some other delineation.) The routine `voting.power` calculates the power of an electoral group by estimating the probability that a vote from this group has in changing the result of an election, by flipping the outcome of districts in which they live.

If the option `pop.groups` is specified in the original `judgeit` command, no further options are required. If not, use the syntax

```
voting.power(j.ob,new.pop.groups=matrix1)
```

where `matrix1` represents the voters in each group within each district.

## 8 Further JudgeIt output options

### System summaries

Using the command `summary()` on a `JUDGEIT` object will give one of two results. Without a year given, the output will be the number of years, as well as the values of model parameters  $\lambda$  and  $\sigma$ .

With a year given, a report of the vote outcomes, predictors, seats and populations will be the output.

## 9 Accessing Raw `judgeit.object` Data

All data are stored in an object of class `judgeit`. If desired, the user can access each component within. Here is a list of components and their attributes:

- `covars` is a list of data frames comprising the predictors for each election in the system. So `judgeit.object$covars[[25]]` is a data frame with the covariates from the 25th election.
- `voteshare` is a list of vectors comprising the vote shares for each election in the system. So `judgeit.object$voteshare[[25]]` is a vector of the results of the 25th election.
- `turnout`, `elgvotes`, `seats` are lists of vectors comprising the actual turnout, the number of eligible voters, and the seats per district in the system for each election.
- `fullrow` is a list of vectors containing those rows whose primary elements (covariates, vote shares, eligible and actual voters and seats) contain complete data.
- `uncL`, `uncLR`, `uncU`, `uncUR` are the uncontested election detection thresholds and imputations as listed above.
- `svexpected.value.only` is the value of `expected.value.only` as given above.
- `simulations` is the number of simulations conducted by `JudgeIt` during each analysis.
- `weight` is the option selected by the user to indicate what weights should be used in the linear model, as described above.
- `distweights` is a list of the actual values of these weights.
- `covarsnew` is a list of data frames of counterfactual or future predictors as manipulated by the option `new.covariates`. It must have the same data type in each column as `covars` though not necessarily the same number of rows.
- `same.dists` is a vector indicating whether the previous election's district map is identical to the current one, as described above.
- `output` contains the output of the last analytical routine, and is displayed with the command `print(JudgeIt.object)`.
- `outputyear`, `outputclass` indicate the year and type of the last analysis conducted. These are used mainly in `plot(judgeit.object)`.
- `beta`, `vc` are the estimates given by the linear model for the system for the coefficients of the covariates and their covariance matrix.
- `sigma`, `lambda`, `sind`, `lind` are, respectively, the mean and year-by-year estimates of the standard error and systematic error fraction of the system.
- `years` is a vector of the names of each election variable in the inputted data frame list. In the case of `house6311`, this is a vector of the election years between 1896 and 1992.

## A The Underlying Method

### Sources of Error in Electoral Data

In principle, the source of error in an experimental outcome can be divided into systematic error, in which the apparatus for conducting an experiment skews the result in the same way each time it is run, and random error, which is different every time an experiment is conducted. In many cases in political science, the estimated error in an experiment cannot be divided into its two component types.

In the application of two-party elections, however, the “experimental apparatus” to be considered is an electoral system that changes minimally between elections in its construction. This gives us a method for estimating what fraction of an election’s variation is caused by properties of the system, and what changes can be considered to be random in nature.

Under this assumption, two different types of elections can be simulated: those in which the existing system is no longer in place, such as the prediction of future elections, and those in which we consider what would happen whether a particular election was run again using the same systematic conditions, two fundamentally different sets of analyses with a common method of generation.

Quantities of interest in the political science literature can often be explained in terms of the outcomes of hypothetical elections or properties of the election system.

## The Gelman-King Model

JudgeIt is used to model two-party electoral systems; choose one party to identify as “Party 1”. (All results for Party 2 are clearly the opposite for those of Party 1.)

In any particular election year, let  $v_i$  be the share of the two-party vote received by Party 1 in district  $i$ . We model the resulting vote share as

$$v_i = X_i' \beta + \gamma_i + \varepsilon_i$$

where  $X_i'$  is a vector of predictor variables with coefficient  $\beta$ , and  $\gamma_i \sim N(0, \lambda\sigma^2)$  and  $\varepsilon_i \sim N(0, (1 - \lambda)\sigma^2)$  are the systematic and random error terms. In this presentation,  $\sigma^2$  is the total error variance, and  $\lambda$  is the share attributed to the systematic error component. The error terms in each district are independent of each other and of those in each other district in the system.

The standard approach to estimating the unknown quantities is to model each year under the Bayesian framework, with noninformative priors on the  $\beta$  and  $\sigma^2$  parameters in each year. To estimate  $\sigma^2$ , take the total variance estimate in each year and pool those estimates together; then use the mean of the pooled estimates as the value of  $\sigma^2$  as the value for each election.

To estimate  $\lambda$  for the electoral system, note that the systematic component of the error is proportional to the votes received in each district in two subsequent elections, yielding

$$v_{i,t+1} = \lambda v_{i,t} + X_i' \beta + \gamma_i + \varepsilon_i,$$

where  $X_i'$  may include as many variables as are available from elections  $t$  and  $t + 1$ . The value of  $\lambda$  used is the mean of these estimated values. Note that estimates of  $\lambda$  can only be obtained when two subsequent elections use the same electoral map, i.e. no redistricting has taken place.

## Generating Hypothetical Elections

Now that the parameters in each election year are estimated, the model can be simulated. For a predictive analysis, the hypothetical vote share in each district is found to be

$$\tilde{v}_i = \tilde{X}_i' \beta + \tilde{\delta} + \tilde{\gamma}_i + \tilde{\varepsilon}_i$$

where  $\tilde{X}_i'$  is a (possibly new) vector of predictors corresponding to those terms in  $\beta$ . In this case, the two error terms are unidentifiable and once again combine so that

$$\tilde{\gamma}_i + \tilde{\varepsilon}_i \sim N(0, \sigma^2).$$

The  $\tilde{\delta}$  term is added under the *general uniform partisan swing* assumption; that is, for small deviations from the observed outcome, a swing in the overall vote share can be represented as the same swing in each district in the system. This allows the user to investigate two scenarios: what would happen if the average vote were to shift by a small amount, or what the electoral map would look like with a particular average vote share (corresponding to a particular shift in the average vote.)

For an evaluation of an election’s underlying properties, or to examine what would happen if we re-ran the election under counterfactual circumstances, we note that the systematic error

component  $\gamma_i$  can be estimated using the data. Since  $y_i$  and  $\gamma_i$  are simulated from the bivariate normal distribution,

$$\begin{bmatrix} y_i \\ \gamma_i \end{bmatrix} \sim N_2 \left( \begin{bmatrix} X_i' \beta \\ 0 \end{bmatrix}, \begin{bmatrix} \sigma^2 & \lambda \sigma^2 \\ \lambda \sigma^2 & \lambda \sigma^2 \end{bmatrix} \right),$$

we can obtain the conditional distribution,

$$\gamma_i | y_i \sim N(\lambda(y_i - x_i' \beta), \lambda(1 - \lambda)\sigma^2).$$

We then use this estimate of  $\gamma_i$  in the simulation equation

$$\tilde{v}_i = \tilde{X}_i' \beta + \tilde{\delta} + \gamma_i + \tilde{\varepsilon}_i.$$

Note that Party 1 wins the election if their share of the two-party vote is greater than one-half. Given  $\beta$  and  $\gamma$ , we then see that the expected seat share is

$$P(v_i > 0.5 | \beta, \gamma, \delta) = 1 - \Phi \left( \frac{0.5 - \tilde{X}_i' \beta - \tilde{\delta} + \gamma_i}{\sqrt{(1 - \lambda)\sigma^2}} \right).$$

To generate the probability distribution for this quantity, we then draw values for  $\beta$  and  $\gamma$  given their conditional distributions, and set  $\delta$  to its required value given the application.

## Application: Calculating Partisan Bias

One definition of partisan bias is the extra seat share received by one party when the average vote is split equally between both. We estimate the distribution of partisan bias conditional on  $\beta$  and  $\gamma$ , then use their distributions to determine a probability interval.

To get one draw from the distribution of partisan bias:

- Draw  $\beta$  and  $\gamma$  from their respective distributions. These represent conditions in the electoral system up until the election.<sup>2</sup>
- Calculate the mean and variance of the vote share in each district conditional on the draws of  $\beta$  and  $\gamma$ .
- Calculate the grand mean vote (with whatever weights are being used) and subtract it from 0.5; this is the value of  $\delta$  we use to adjust the mean vote to 0.5.
- Determine the expected seat share  $P(v_i > 0.5 | \beta, \gamma, \delta)$  for each district and take the weighted mean. Twice the (weighted mean minus 0.5) is the partisan bias conditional on  $\beta$  and  $\gamma$ .

Repeating this procedure gives us the distribution of the partisan bias.

## B Demonstrations

After loading the JudgeIt library, the demonstrations `seatsdemo`, `probdemo`, `distreportdemo` and `svsumdemo` are available through the command `demo`.

## References

- Gelman, Andrew and Gary King. 1990a. "Estimating Incumbency Advantage Without Bias." *American Journal of Political Science* 34(4, November):1142–1164. <http://gking.harvard.edu/files/abs/inc-abs.shtml>.
- Gelman, Andrew and Gary King. 1990b. "Estimating the Electoral Consequences of Legislative Redistricting." *Journal of the American Statistical Association* 85(410, June):274–282. <http://gking.harvard.edu/files/abs/svstat-abs.shtml>.

<sup>2</sup>The routine used to generate multivariate normal random variables is taken explicitly from the R library MASS.

- Gelman, Andrew and Gary King. 1994. "Enhancing Democracy Through Legislative Redistricting." *American Political Science Review* 88(3, September):541–559. <http://gking.harvard.edu/files/abs/red-abs.shtml>.
- Gelman, Andrew, Jonathan Katz and Gary King. 2004. *Rethinking the Vote: The Politics and Prospects of American Electoreal Reform*. New York: Oxford University Press chapter Chapter 5, Empirically Evaluating the Electoral College, pp. 75–88. <http://gking.harvard.edu/files/abs/rethink-abs.shtml>.
- Grofman, Bernard and Gary King. 2007. "The Future of Partisan Symmetry as a Judicial Test for Partisan Gerrymandering after LULAC v. Perry." *Election Law Journal* 6(1, January):2–35. <http://gking.harvard.edu/files/abs/jp-abs.shtml>.
- King, Gary and Andrew Gelman. 1991. "Systemic Consequences of Incumbency Advantage in the U.S. House." *American Journal of Political Science* 35(1, February):110–138. <http://gking.harvard.edu/files/abs/sysconseq-abs.shtml>.
- King, Gary and Robert X Browning. 1987. "Democratic Representation and Partisan Bias in Congressional Elections." *American Political Science Review* 81(4, December):1252–1273. <http://gking.harvard.edu/files/abs/sv-abs.shtml>.