A Tutorial for Calculating the Closeness Measure Proposed by Luechinger, Schelker, and Schmid (2024)

Lukas Schmid

December 9, 2025

Contents

Introduction

This tutorial introduces researchers to code that helps with the calculation of an electoral closeness measure for proportional representation systems. It is based on the paper by Luechinger, Schelker, & Schmid (2024) that calculates closed-form solutions for the vote shortfall of non-elected candidates and the vote surplus of elected candidates. This tutorial deviates from this approach by simulating those vote margins, which should lead to the same solutions as the analytical closed-form solutions provided in the paper. A simulation approach is required if certain electoral systems include elements, such as quotas for parties and/or candidates, which make the calculation of a closed-form solution challenging.

We illustrate the calculation of the closeness measure using the running example from our paper for a district with three seats and three parties with three candidates each.

Step 1: Data preparation

The first step is to prepare the data such that the naming of the variables is standardized for the later functions. This can be done using the function *PrepareData*.

Mandatory arguments

The user must specify the following arguments:

- *data*: Name of the dataset
- *election_cycle_name*: Variable name of the election cycle, typically the year
- *district_name*: Variable name that captures districts
- *system*: type of electoral system ("closed" for closed-list systems or "open" for open-list systems)
- *party_name*: Variable name that captures parties
- *votes_j_name*: Variable name that captures party votes
- *alliances*: TRUE if alliances (and potentially suballiances) are allowed, FALSE otherwise (default: FALSE)
- *additional_vars*: additional variables which may be required to calculate a threshold (default: " ", no additional vars)

In addition, the dataset *data* should include a variable name *elected*, which is 1 for elected candidates and 0 for non-elected candidates

Arguments for open-list systems

- *votes_h_name*: Variable name that captures candidate votes
- *cand_id_name*: Variable name that captures the ID of candidates

Arguments for closed-list systems

- *rank_name*: Variable name that captures the ordinal ranking assigned to candidates, according to which a party's seats are allocated to candidates (e.g., 1 for the first-ranked candidate, 2 for the second-ranked candidate and so forth)

Arguments for systems with alliances (and potentially suballiances)

- *alliance_name*: Variable name that captures alliances
- *suballiance_name*: Variable name that captures suballiances

Example

The following example is based on our running example in Luechinger, Schelker, and Schmid (2024) with the following distribution of party votes (*pvotes*) and candidate votes (*cvotes*):

```
data_example <- data.frame(district=rep(1,9),
                           year=rep(2024,9),
                           id=c(1:9),
                           party=c(rep("P1",3),rep("P2",3),rep("P3",3)),
                           elected=c(1,1,0,1,0,0,0,0,0),
                           pvotes=c(rep(45,3),rep(35,3),rep(20,3)),
                           cvotes=c(c(25,11,9),c(22,8,5),c(8,7,5))
                           )
print(data_example_prep)
```

```
##   district year party id_cand elected votes_j votes_h alliance suballiance
## 1        1 2024     1       1       1      45      25        1           1
## 2        1 2024     1       2       1      45      11        1           1
## 3        1 2024     1       3       0      45       9        1           1
## 4        1 2024     2       4       1      35      22        2           2
## 5        1 2024     2       5       0      35       8        2           2
## 6        1 2024     2       6       0      35       5        2           2
## 7        1 2024     3       7       0      20       8        3           3
## 8        1 2024     3       8       0      20       7        3           3
## 9        1 2024     3       9       0      20       5        3           3
##   alliance_dummy suballiance_dummy
## 1              0                 0
## 2              0                 0
## 3              0                 0
## 4              0                 0
## 5              0                 0
## 6              0                 0
## 7              0                 0
## 8              0                 0
## 9              0                 0
```

The transformed data with the key variables used for the calculation of the closeness measure looks as follows:

```
data_example_prep <- PrepareData(data=data_example,
                                 election_cycle_name = "year",
                                 district_name="district",
                                 system="open",
                                 party_name="party",
                                 votes_j_name="pvotes",
                                 cand_id_name="id",
                                 votes_h_name="cvotes",
                                 alliances=F)
print(data_example_prep)
```

```
##   district year party id_cand elected votes_j votes_h alliance suballiance
## 1        1 2024     1       1       1      45      25        1           1
## 2        1 2024     1       2       1      45      11        1           1
## 3        1 2024     1       3       0      45       9        1           1
## 4        1 2024     2       4       1       35       22         2            2
## 5        1 2024     2       5       0       35        8         2            2
## 6        1 2024     2       6       0       35        5         2            2
## 7        1 2024     3       7       0       20        8         3            3
## 8        1 2024     3       8       0       20        7         3            3
## 9        1 2024     3       9       0       20        5         3            3
##   alliance_dummy suballiance_dummy
## 1              0                 0
## 2              0                 0
## 3              0                 0
## 4              0                 0
## 5              0                 0
## 6              0                 0
## 7              0                 0
## 8              0                 0
## 9              0                 0
```

Step 2: Calculation of closeness measure

In the second step, researchers can calculate the closeness measure using the function *CalculateMargins*. It has the following arguments:

Mandatory arguments

- *data_input*: data for which vote margins should be calculated (should be output of function *Prepare-Data*)
- *system*: Electoral system, either "closed" for closed-list systems or "open" for open-list systems.
- *method*: either a pre-specified seat allocation method ("dHondt" or "SainteLague") or a user-written seat allocation method (see below for more details).
- *convcrit*: is a convergence criterion used by the binary search algorithm to determine when to stop adjusting vote counts.

Other arguments

- *return_option*: result data frame is returned if TRUE (default: FALSE)
- *outfile_name*: name (and path) of outfile where results should be stored
- *threshold*: type of threshold (default: "none")
- *calculate_all_seats*: calculate vote margin for all candidates and seats if TRUE (default: TRUE)
- *total_seats_name*: variable with total seats per districts, if not specified (default) the sum of elected per district is the number of
- *append*: if true, calculations will be appended to existing file with file name "outfile_name", otherwise (default) a new file witht the name "outfile_name" will be created
- *additional_vars*: additional variables for output
- *print_party*: print new party iteration in console if TRUE (default: FALSE)

Example

We use the following command for an open-list system using the d'Hondt method and save the output in the file "Example_votemargins.csv":

```
CalculateMargins(data_input=data_example_prep ,
                 method="dHondt",
                 convcrit=0.001,
                 additional_vars="id",
                 system="open",
                 return_option = TRUE,
                 outfile_name="Example_votemargins.csv")
```

```
## [1] "New year: 2024"
## [1] "New district: 1"

##   party id_cand elected votes_j votes_h alliance suballiance alliance_dummy
## 1     1       1       1      45      25        1           1              0
## 2     1       2       1      45      11        1           1              0
## 3     1       3       0      45       9        1           1              0
## 4     2       4       1      35      22        2           2              0
## 5     2       5       0      35       8        2           2              0
## 6     2       6       0      35       5        2           2              0
## 7     3       7       0      20       8        3           3              0
## 8     3       8       0      20       7        3           3              0
## 9     3       9       0      20       5        3           3              0
##   suballiance_dummy year district votemargin
## 1                 0 2024        1     14.001
## 2                 0 2024        1      2.001
## 3                 0 2024        1     -2.001
## 4                 0 2024        1     14.001
## 5                 0 2024        1    -10.001
## 6                 0 2024        1    -10.001
## 7                 0 2024        1     -2.499
## 8                 0 2024        1     -2.499
## 9                 0 2024        1     -3.001
```

Functions

Above, we have explained the *PrepareData* function that renames the variables for the function *CalculateMargins*. In this section, we will briefly explain the key functions.

CalculateMargins

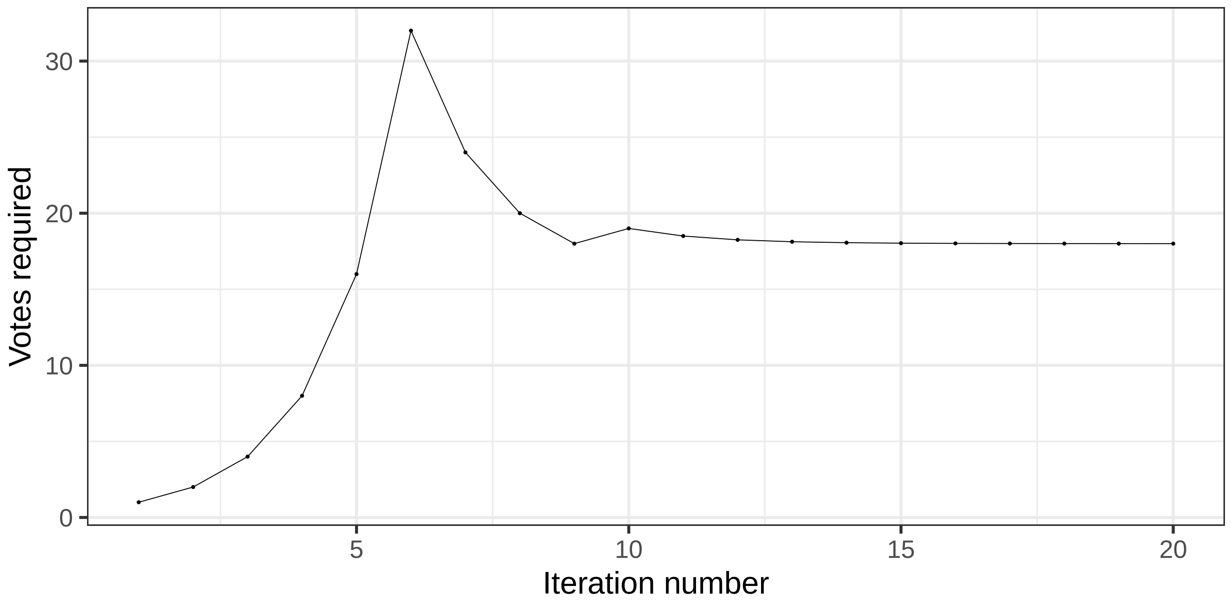
This function calculates the vote margin for each election and each district in the data. It loops over the function *GetRVSimulation__Candidate* .

GetRVSimulation__Candidate

This function calculates the vote margin for all parties and candidates per election and district using simulation. It calls *GetVotesRequiredClosedList* for closed-list systems and *GetVotesRequiredOpenList* for open-list systems.

GetVotesRequiredClosedList

Binary search method to calculate the votes required to be barely elected for a specific candidate in a party for closed-list systems. This can be illustrated using the following graph for the 2nd candidate of party P2 in the example in Luechinger, Schelker, and Schmid (2024):



The votes required are 18 and thus the vote margin is -10 (8 votes - 18 votes required).

GetVotesRequiredOpenList

Binary search method to calculate the votes required to be barely elected for a specific candidate in a party for open-list systems.

GetSeatsHighestAverage

Calculates the number of seats for a single party for highest average methods.

Further functions

- *CalculateRatios* calculates ratios (e.g., d'Hondt numbers) in long format which is used in function *GetSeatsHighestAverage*.
- *CollapseVotes* collapses party votes at the alliance or suballiance level which is used by *GetSeatsHighestAverage*.
- *CutDataByQuorum* removes all observations that do not meet a threshold.

Changing the electoral system

Researchers can code their own seat apportionment method. There are basically two different approaches to doing this:

1. If there is only threshold for certain parties, a new threshold can be added to the function *CutDataByQuorum*.

2. If the seat apportionment method is more complicated, it can be added to the functions *GetVotesRequiredOpenList* or *GetVotesRequiredClosedList*. The new function should have an output *elected* which is 1 if a candidate is elected and 0 otherwise and will be called after the if statement for this method. As an example, there is a template for Brazil in the function *GetVotesRequiredOpenList* (search for if (method==“Brazil”)). It is important to include the new method in the method argument when executing the function *CalculateMargins* (method=“yourmethod”).

Reference

Luechinger, S., Schelker, M., & Schmid, L. (2024). Measuring closeness in proportional representation systems. *Political Analysis*, *32*, 101–114.