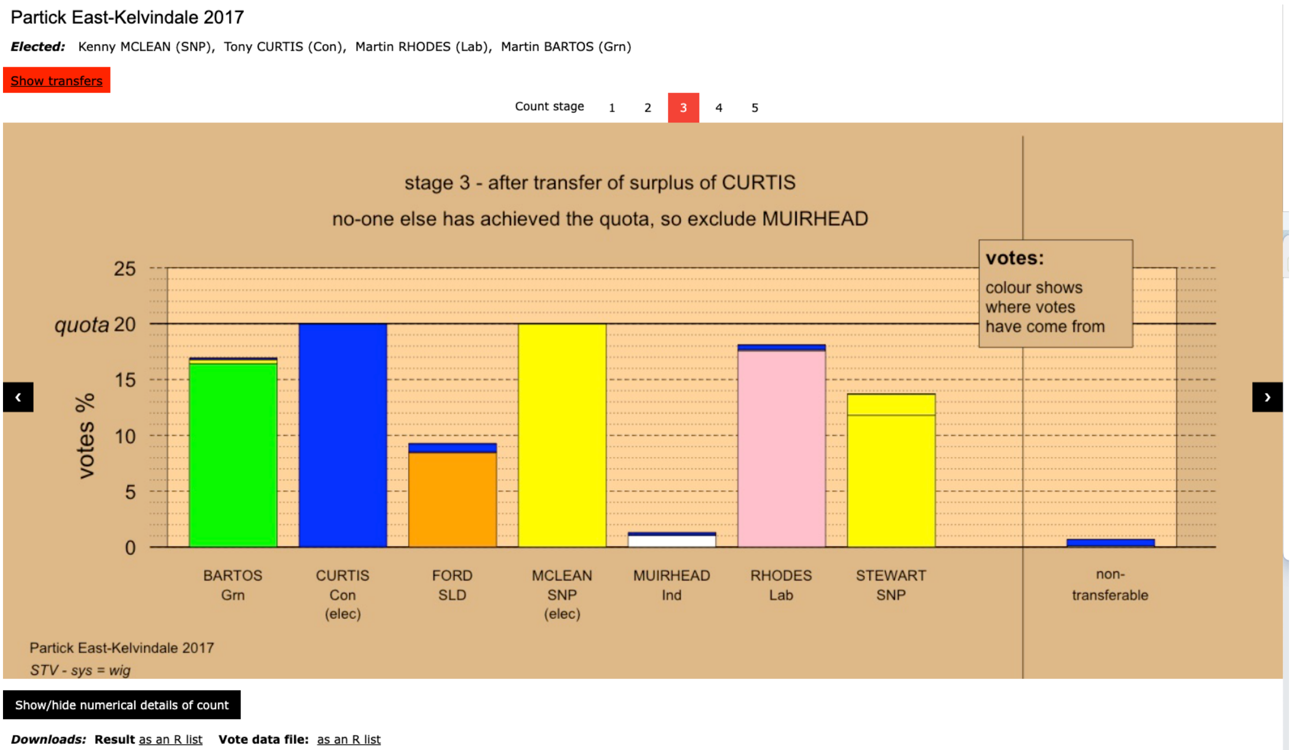
pref R package manual *0.3.0, 6 Jan 2024*

Denis Mollison (Heriot-Watt University)

Summary *page*

Summary

Implements the Single Transferable Vote (STV) electoral system, with clear explanatory graphics. The core function is Meek's method, the purest expression of the simple principles of STV, but which does require electronic counting. It can handle votes expressing equal preferences for subsets of the candidates. A function stv.wig implementing the Weighted Inclusive Gregory (WIG) method is also provided. The principles of STV and an outline of the steps required to implement it are described in $7.



Example of output for a Scottish Council election

The required vote data format is as an R list (see $ 4 below). A function *pref.data* is provided to transform some commonly used data formats into this format.

1 Package description

The goal of the function ***stv*** (see $3) is to count votes from an STV election, and to provide a clear and full description of the count and the result, using both numbers and graphics. It uses Meek STV, and allows for votes that include equal preferences. Graphical output includes expository web pages; the function ***stv.result*** prints a summary of the election count and result.

The function ***stv.wig*** ($3) implements the Weighted Inclusive Gregory STV method, as used in Scottish council elections since 2007; it has the same options as *stv*.

[A function, *stv.batch*, to run counts for multiple elections, as used to process the complete 2012, 2017 and 2022 Scottish Council elections, is too fragile to include here, but is available from the author.]

Election data for preference voting come in many formats, ranging from a simple vote matrix to the incluson of full candidate names and party affiliations. A function ***pref.data*** is provided to translate various common vote file formats into a flexible R list format, whose only essential element is a matrix of vote data ($4). This flexible R list format provides the input for *stv* and *stv.wig*. A range of examples from actual elections is included ($6).

The most recent stable version of pref can be installed from CRAN (https://cran.r-project.org/) using `install.packages(pref)`.

The development version can be installed from Github:

```
library(devtools)
install_github("denismollison/pref")
```

Either way, the package can then be loaded into an R session using `library(pref)`.

2 Usage

First, if vote data are not in the recommended format they should be converted into an R list, *elecdata* say, using the function *pref.data* (see $4 below). Then the election count is run using the function *stv*:

```
res=stv(elecdata,outdirec="stv_out")
```

(or similarly for *stv.wig*). This sets an output directory, but otherwise uses the default options for *stv*: a brief summary of the result *res* will be printed, while full details and plots with webpages to display them will be stored in *outdirec*. If you follow the above literally, *outdirec* will be a directory called "stv_out" within your working directory; more likely, you will prefer to replace "stv_out" with the path to a directory of your choice.

Other options (see $3) include pausing the election at each stage with fuller details and a plot of the current state of the count displayed, or omitting graphical output altogether; while the output directory can be left as the default value of tempdir() so that it is ephemeral.

To print fuller details of the election count use *stv.result(results)*. To display webpages load *stv_out/index.html* into a browser.

3 Functions associated with the count

This section describes the two count functions, *stv* and *stv.wig*. Each has both input and output in the form of an R list; and also, as default, makes plots of the count stages, and webpages to view them.

The required data input format is described in $4.

For*stv*, which implements Meek STV allowing equal preferences, the options and their default values are as follows:

res = **stv**(elecdata,outdirec=tempdir(),verbose=FALSE,plot=TRUE,
webdisplay=FALSE)

The value returned here is a list *res* of results, containing the following items:

res$elecname - election name
res$sys - the STV system used (="meek")
res$elec - the names of those elected, in order of election
res$itt - candidates in order of election/exclusion, reported at each stage
res$counttext - the results of each stage, in words
res$votes - a matrix of the votes at each stage and the final keep values
res$quotatext - the total number of votes, and initial and final values of the quota
res$va - a 3D array showing how votes have transferred (from first to current preference) for each stage
res$keep - the keep values (as %s) at each stage

This list, and the input vote data list *elecdata*, are stored as compressed R data files (.rda format) in the directory *outdirec*. The default option for *outdirec* is an ephemeral directory *tempdir()*; this choice is to avoid writing to a user's filespace without warning. More usually, users will wish to keep election results, and should make their own choice of *outdirec* accordingly.

Note that output file names will include the election name: it is therefore important that the election name should not include any of the special characters that operating systems, especially Windows, do not allow in file names (*e.g.* quotes or brackets); note that spaces(" ") in the election name will be replaced by underscores ("_").

With the default options, the function *stv* will also make webpages in *outdirec* with plots showing each stage of the count . If you set *webdisplay=TRUE* they should be displayed automatically - but note that this feature is system-dependent and might fail. The safer option is to stay with the default option *webdisplay=FALSE,* and open the web pages from *outdirec* later. Note that these webpages include a numerical display of full details of the count and plots of transfers, each as an option that requires clicking a button to display.

A full expository option is stv(elecdata,outdirec="out",verbose=TRUE, webdisplay=TRUE); this displays progress in numbers and a plot at each stage of the count, requiring the user to press 'return' to ask for the next stage when ready, and concludes by displaying the full results as web pages. Alternatively, combining the options verbose=FALSE and plot=FALSE calculates the detailed result with minimal printout and no graphical output.

Internal functions used in the vote count calculations (which can be found in *functions_meek.R*) include ***transfer***, which revises the keep values k at each stage. Following transfers, the stage is completed by the function ***decision***: if any additional candidates have sufficient votes they are deemed elected, otherwise the candidate with fewest votes is excluded. A function ***decision_text*** expresses the decision in words (a function ***plural*** helps with grammar).

The other STV count function is *stv.wig*; this has the same default options as *stv*, thus:

```
res = stv.wig(elecdata,outdirec=tempdir(),verbose=FALSE,
  plot=TRUE,webdisplay=FALSE)
```

This function implements the `Weighted Inclusive Gregory' (WIG) algorithm, which has been used for Scottish Council elections since 2007. This differs from Meek in not allowing transfers to already elected candidates. While that may seem harmless - or even desirable - it introduces discontinuities, and the quota cannot be adjusted satisfactorally: a significant factor in political elections and those with large numbers of candidates. The sole justification for preferring it to Meek is that it can if necessary be counted by hand (which is probably not relevant if you're reading this). The differences in practice between the WIG and Meek algorithms can be explored at www.macs.hw.ac.uk /~denis/stv_elections, which cross-references each Scottish council count (for 2012 and 2017) with its Meek version.

The value returned by the *stv.wig* function is a list containing the same items (*mutatis mutandis*) as for the *stv* function, except for the last (*keep*, which is not relevant for WIG).

4 Functions associated with data input

The input format required by *stv* and *stv.wig* is a list *elecdata*, or *ed* for short, which can have the following elements:

ed$e - election name
ed$s - number of representatives to be elected
ed$c - number of candidates
ed$nv - number of votes
ed$v - matrix of votes (*ed$nv* × *ed$c*)
ed$m - multiplicity for each vote (=1 if just one vote per row)
edn, edf, ed$n2 - name, first name, and abbreviated name for each candidate
edp, edcol - party acronyms and party colours of candidates (if appropriate - otherwise = "")

The first thing to say is that the only essential element here is the vote matrix ed$v. If the candidates' names are not given, they will be assumed to be *dimnames(ed$v)[[2]]*, and if this is empty capital letters A,B,... will be used.

An election name and the number of representatives to be elected are also essential, but if these are not given the user will be asked for them (see first example, *Yale*, in $6).

The function ***pref.data*** converts some common preference data formats into the required format. Its defaults -

pref.data(datafile,mult=FALSE,details=TRUE,parties=FALSE, ballot=FALSE,friendly=FALSE,header=FALSE)

- correspond to the common .blt format, which is for example used for Scottish council election data; except that for those data one needs mult=TRUE, and should also set parties=TRUE.

In more detail, the options are:

mult - if TRUE, the multiplicity of each row of the vote matrix is given as its first element

parties - either FALSE, or the name of a file with party acronyms nd colours

ballot - if TRUE, *i*th entry in a row is the preference number for candidate *i*

- if FALSE, rows are candidate numbers in order of preference, with bracketing indicating equal preference

friendly - if TRUE, file starts with title, then *ed$s* and *ed$c*, then candidate names

- if FALSE, starts with *ed$c* and *ed$s*, with candidate names and election title at end of file (the common `.blt' format)

details - if FALSE, data are simply a vote matrix, with the first column containing multiplicities if mult=TRUE and the first line giving candidate names if *header*=TRUE

Other internal input functions (which can be found in *functions_input.R*) are ***abbrev***, to calculate a suitable abbreviated name, *name2*; ***party_colour***, to associate party colours with candidates if appropriate; and ***cap_words***, for consistency in name style.

Other data formats

Hopefully, users with data in other formats will not find it too hard to convert their data into an R list with components as for *ed* above. Some points to note are:

ed$nv is the number of lines of vote data: this will be the actual vote total if these are all individual votes; otherwise, the vote total is *sum(ed$m)*.
The variable *ed$n2* is used to avoid names of excessive length, and to distinguish two candidates with the same surname; if these are not a problem for your data, just set *ed$n2 = ed$n*.

If party acronyms and colours are not relevant, just set *ed$p* and *ed$col* to the empty character (*i.e. = rep("",ed$c)*). The program will then use its `rainbow' default to choose colours for the graphics.

STV allowing equal preferences

It should be noted that *elecdata* does not explicitly include information on whether equal preferences are allowed. The numbers in each row of the vote matrix establish that voter's order of preference: if some of the numbers are equal, *stv* treats them as equal preferences. On the other hand, *stv.wig* expects orders of preference to be strict, and will crash if given data including equal preferences.

5 Functions associated with output

The function ***stv.results*** gives a fairly full description of the count in narrative order. It prints the main elements of a results list produced by either *stv* or *stv.wig*,: *elecname, sys, counttext, elec, votes* and *quotatext*.

The graphics are produced by internal functions ***voteplot*** and ***webpages***. They are not intended to be used separately from the main count functions, but some users may find them of interest. They are unusual uses of R, particularly the latter, which uses R to automate the writing of web pages to present plots of the count. I would be interested to hear of others who have used R to write web pages. And suggestions as to how these functions might be made more elegant will be gratefully received.

Note that the plots are saved as jpegs and then displayed using a function *plot_jpeg* that relies on the package *jpeg*. This is to avoid dependence on local R plotting parameters. If *jpeg* is not available, a warning message will be printed and *plot* set = FALSE to disable plotting.

In case of any other problems with live plots and browser opening, ensure that the (default) options plot=TRUE and webdisplay=FALSE are set, allowing you to edit the webpages before trying to display them.

6 Examples

This section lists three examples where data require preprocessing into an R list, using the function *pref.data*, and six where the data are already available as an R list (use *data(package="pref")* for a list of these), so that a vote counting function (*stv* or *stv.wig*) can be applied directly.

Input function *pref.data*

Yale

An example with input data simply a vote matrix; you will be asked for an election name ("Yale") and the number to elect (4). [For more details of this election see the last of the count function examples.]

```
datafile=system.file("extdata","yale.dat",package="pref")
yale=pref.data(datafile,details=FALSE)
```

Jedburgh and District 2012

An example converting Scottish Council election data from their original ".blt" format with full details, including party names and colours, into an R list .

```
datafile=system.file("extdata","Jedburgh2012.blt",package="pref")
parties12=system.file("extdata","parties_SC2012.txt",package="pref")
jed12=pref.data(datafile,mult=TRUE,parties=parties12)
```

Then running jed12result=stv.wig(jed12) should reproduce exactly the official result of this election.

John Muir Trust 2002

An example in ".blt" format with equal preferences indicated using brackets. Note that this file has been reordered in "friendly" format, *i.e.* with the election title and candidate names at the beginning of the file.

```
datafile=system.file("extdata","jmt2002.blt",package="pref")
j02=pref.data(datafile,friendly=TRUE)
```

Then running j02result=stv(j02) should reproduce exactly the official result of this election.

Count functions *stv, stv.wig*

The first four examples below come from Scottish Council elections; using *stv.wig* reproduces the official result exactly. The full results for 2012-22 can be found at www.macs.hw.ac.uk/~denis/stv_elections. For the elections of 2012 and 2017, both the official count and one using Meek STV are given, with links to switch between the two for ease of comparison.

Helensburgh Central 2012 hc12result=stv.wig(hc12)

A very simple example of an election to choose 4 representatives, where 4 candidates had first preference totals of over 20%, and so were elected at the first stage. Note that in this simple case there is no difference between different STV methods.

Partick East - Kelvindale 2017 p17result=stv.wig(p17)

A more typical STV election (see page 1 for a sample of the output web pages).

North West and Central Sutherland 2017
nws17wig=stv.wig(nws17)

An example exposing one of the flaws of WIG STV: none of the elected candidates achieved the quota.

Cumnock and New Cumnock 2017 cnc17wig=stv.wig(cnc17)

An example where WIG and Meek give different results: here one of the two leading parties has fewer first preferences but they are more equally divided between their two candidates; under Meek this matters less because reductions in the quota allow more transfers between a party's candidates.

Council 1999 c99result=stv(c99)

An example from an election allowing equal preferences.

Yale yaleresult=stv(yale) (or =stv.wig(yale))

Run the first input example first, to generate *yale*.
An example (from a faculty election at Yale university) with a large number of candidates, 44 for 4 places. While the graphics cope, they are difficult to read (but you can look at plots separately). Note that this election was held under \`Cambridge rules', so neither *stv* nor *stv.wig* reproduces the

official outcome. Indeed they both agree in filling the 4th place differently from the official result.

'Yale - last 12'   y12result=stv(y12)   (or =stv.wig(y12))

A version of the previous example, starting after 32 candidates have been excluded (but noone yet elected), to give clearer plots of the later (and decisive) stages of the election.

Output function *stv.results*

stv.result is a simple function to print out the results of an election. It takes one argument, an R list *res* of results, as generated by an STV count function (*stv* or *stv.wig*), and has no options. It prints out the main components of *res* in an order that provides a narrative of the election, namely: *elecname, sys, counttext, elec, votes, quotatext*. For example:

c99result=stv(c99); stv.result(c99result)

nws17wig=stv.wig(nws17); stv.result(nws17wig)]

7 Background: STV

The Single Transferable Vote is a system designed to elect representatives in such a way that each represents the same number of voters. For its relation to core democratic principles, and comparison with alternative forms of proportional representation, see Mollison (2023).

In STV, each voter provides their order of preference; these preferences are used to transfer unused votes or parts of votes from earlier to later preferences, according to the following algorithm:

(a) Votes are initially assigned to the voter's first choice

(b) The number of votes required to ensure election is calculated; this is called the *quota*. When the total number of votes is v and there are s seats to be filled, the quota is v/(s+1)

(c) Any candidate whose total reaches the quota is elected; if they have more than the quota, the surplus is transferred to their voters' next preferences

(d) If not all seats are filled, the candidate with fewest votes is excluded and all their votes are transferred to their voters' next preferences

Steps (b-d) are repeated, as necessary, until all seats are filled.

In counting the votes the only significant difficulty is in distributing the surplus σ_i of an elected candidates with vote v_i, where fairness suggests that equal fractions σ_i/v_i of each vote should be transferred, and that, as a slightly less obvious part of that fairness, transfers should go exactly where the voter has requested even if the recipient has sufficient votes already. The latter feature has the knock-on effect of requiring further transfers, which is why (a) the count requires a computer, and (b) why this feature was not implemented until computers were available (Meek published his algorithm in 1969/70).

Meek's method has a number of other advantages, conceptual and practical (Mollison 2024), and is widely regarded as the best form of STV. There is really no good reason why any other method should be used if the vote data can be gathered securely as a computer file. It is therefore surprising that it has not previously been available on CRAN.

History of use

The original idea of STV goes back to Thomas Wright Hill (1819), with various improvements (use of preferential voting (Andrae, Hare), quota $\sim v/(s+1)$ (Droop) rather than $\sim v/s$ (Hare), and fractional transfers (Gregory)) introduced between 1855 and 1881, after which there was little change until Meek's reassessment for the computer age nearly 100 years later. STV has been used for political elections in various countries and regions, including Tasmania (since 1909) and Ireland (since its independence in 1921). In the UK it is currently used for the Northern Ireland Assembly and for Scottish Council elections - the latter provide the best source of STV data currently available.

STV is also increasingly used for electing the governing bodies of non-political organisations. Meek STV allowing voters to express equality of preference has been used by the John Muir Trust and the London Mathematical Society (since 1998 and 1999 respectively). The program used for those elections, the first publicly available for STV allowing equal preferences, was written in Pascal by the late David Hill; it was originally available through ERS Services.

Other STV methods and software

The key motive for this package was to put Meek STV allowing equal preferences into the public, open software, domain together with output, particularly graphics, that help the public to understand how STV works. The hope is that this should make what is widely regarded as the best form of STV easily available to all for.the long term.

Hill's program remains available, currently through Civica, but not as open software; also, as a Pascal program, its prospects of long term support are limited. A wide range of STV methods are available - commercially - through Opavote; see https://www.opavote.com/methods/single-transferable-vote for descriptions of these. The package *STV* on CRAN makes available Cambridge (Massachusetts) STV, but this is widely regarded as obsolete - its legislation prescribes that only counting methods available in 1941 can be used. The package *Vote* on CRAN presents an elegantly written STV program that does allow equal preferences. However the system used, though it is described as being `developed from' Hill's 1987 Meek STV program is in fact a generalised version of WIG, in which the quota is allowed to reduce during the count; but this revision of the quota is incomplete, being only for those elected later in the count, with the consequence that the method fails to share several key advantages of Meek.

8 Development and extensions

Much of the motivation for developing this software lay in the study of the large sets of data available from Scotland's council elections since 2007, comprising over 1000 individual elections in all. The development therefore included programs to tidy up such data for processing, and a program *stv.batch* to run counts for such large data sets. This software is not currently at a stage suitable for the public domain, but is available from the author on request.

Future planned developments include adapting the web display functions so that they can be used on their own to display already calculated vote results.

The more radical future ambition is to add software for Condorcet's method (`majority rule'), for calculating and analysing results of elections to make a single choice from preference data.

At a more basic level, it would be useful to include implementations of other STV counting methods, and to provide more data handling options. Feedback from users on what would be useful will be welcome.

References

Meek, B. L. (1969) `Une nouvelle approche du scrutin transférable', *Mathématiques et sciences humaines* **25**, 13-23.

Meek, B. L. (1970) `Une nouvelle approche du scrutin transférable (fin)'. *Mathématiques et sciences humaines* **29**, 33-39.

Mollison, Denis (2023) `Fair votes in practice', https://arxiv.org/abs/2303.15310.

Mollison, Denis (2024) `Why Meek?' *(draft under development)* https://www.macs.hw.ac.uk/ denis/stv/why_meek.pdf.

Feedback

Comments, and expressions of interest in collaboration, will be very welcome.

denis.mollison (at) gmail.com