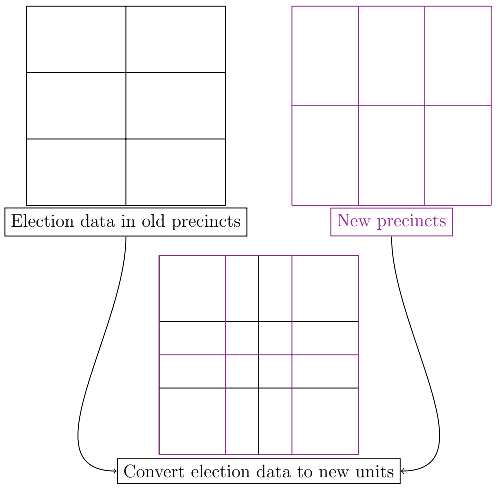
PRORATION AND ROUNDOFF

ASSAF

1. Introduction

This document is written to be a brief explanation of two processes which we call *proration* and *roundoff*. *Proration* is the process of taking electoral data given in terms of one partition and assigning it to another partition of a state. For example, if election data is given in terms of old precincts, and we want to see it in terms of current precincts, which might be geographically different from the past precincts.


Election data in old precincts
New precincts
Convert election data to new units

Roundoff is the process of assigning small units to larger units that may not match. For example, if district lines cut through counties, and you want to assign counties to districts, roundoff is a way to make the assignment in a (mostly) deterministic and complete manner.

2. Proration

2.1. **A Visual Explanation.** Proration uses two central assumptions:

- The old partition and new partition have a mutual refinement[1] which contains population data.
- The voting population, and the votes for each candidate are all uniformly distributed in each region of the old partition.

The first assumption is almost always satisfied if the old partition and the new partition were drawn in the same census cycle. In this case, the mutual refinement is simply census blocks. The second assumption is problematic, but at the writing of this document, this is necessary

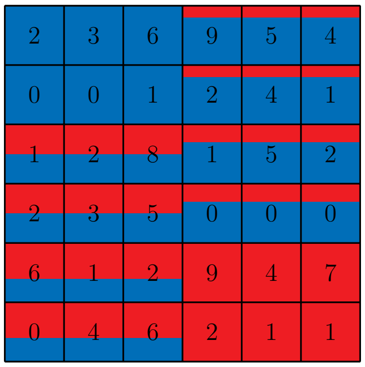
[1]in other words, they are composed of the same basic building blocks.

in order to assign election data to the new partition in a reasonably fast time. At the end of this section are some ideas on how to bypass this assumption, and to assign election data in a different, more computationally expensive, way.

In the above example, assume that we have the following election results, given in old precincts:

0%R 100%D	20%R 80%D
50%R 50%D	30%R 70%D
60%R 40%D	100%R 0%D

We take the old precincts, and the new precincts, and look at the pieces which constitute them - the blocks. We overlay the blocks on the old precincts, and use the second assumption to compute the election results in each block. Note that each block has some population, written in black.

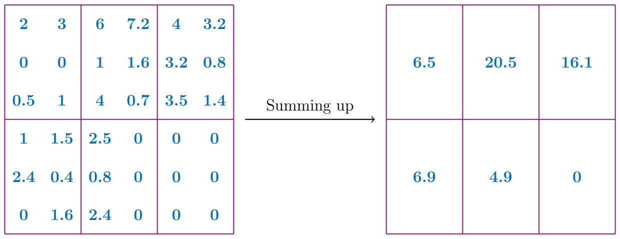


2	3	6	9	5	4
0	0	1	2	4	1
1	2	8	1	5	2
2	3	5	0	0	0
6	1	2	9	4	7
0	4	6	2	1	1

We then multiply the vote ratios by the populations of each block, in order to get the vote totals in each block for each party. For example, below are the computed **D** votes:

2	3	6	7.2	4	3.2
0	0	1	1.6	3.2	0.8
0.5	1	4	0.7	3.5	1.4
1	1.5	2.5	0	0	0
2.4	0.4	0.8	0	0	0
0	1.6	2.4	0	0	0

Next, we overlay the new precincts, and simply add up the votes in order to get the new totals:



Note that some of the vote totals are not integers, but are rather estimates of the real vote totals.

2.2. **Some psudocode.** The proration process can be also understood in terms of a pseudocode.

```
for i = 1 to (number of blocks):
  if block(i) is in old_precinct(j) and new_precinct(k):
    votes_in_block = (block population)*(old_precinct(j) vote percent)
    add votes_in_block to (new_precinct(k) votes)
```

2.3. **Some Problems and Generalizations.** The above proration process does not work if the precincts do not have a good mutual refinement. In this case, there are a few solutions:

(1) The first possibility is to roundoff *current* blocks to the *old* precincts, and to proceed as usual. The roundoff method is explained in the next section of this document.
(2) The second option is to prorate by area. This is equivalent to assuming that voters are equally distributed within a precinct, and then using any mutual partition of the old and new precincts as a base for proration (with populations added proportionally to the area).

    More concretely, in pseudocode, this is simply:

```
for i = 1 to (number of new precincts):
  if new_precinct(i) intersects old_precinct(j):
    area(i,j) = area(new_precinct(i) intersection new_precinct(j))
    area_ratio = area(i,j)/area(old_precinct(j))
    intersection_votes = area_ratio*(old_precinct(j) vote count)
    add intersection_votes to new_precinct(i)
```

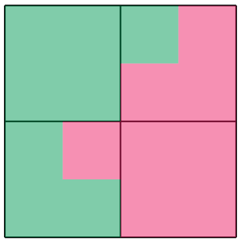
    Note that in this option, vote totals may end up as real numbers, and not as pretty fractions, like in the other proration methods.

There is also a method to do proration that is more accurate than anything written above, but is much more data intensive, is technically more involved, and requires a lot more computation power. Every county has a voter registration file, which in some cases has information on who voted in which election. These files contain the mailing address of every registered voter, and thus can give you a distribution of where voters are located within a county, and hence within a precinct.

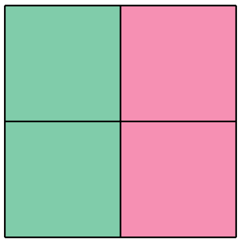
Thus, using voter registration files, it might be possible to use vote ratios of election data in old districts to prorate *voters* instead of *blocks*.

3. Roundoff

3.1. **Roundoff By Area.** The simplest way to round off is to take every small unit (we will use counties in this example) and find the larger unit (districts) which contains the largest portion of its area. For example, consider two districts which partition a square region split up into four counties:

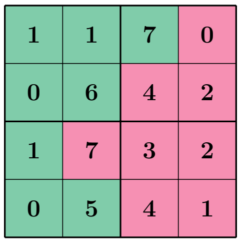


Under the area method of roundoff, the district association of the counties becomes:



Note that there is a minute chance that two districts may contain an equal amount of a county's area, and no other districts contain as much area. In this case, the algorithm will just toss a coin. However, in real data, this is extremely rare.

3.2. **Roundoff by Population.** In this method of roundoff, we assume that both the districts and the counties are composed of the same base building blocks, with populations attached. In this case, we associate a county to the district containing the largest portion of its population. For example, in the above, if the counties were further subdivided into squares with populations:



then the roundoff by population would look like: