The Stata Journal (2017)
17, Number 3, pp. 652–667

Dealing with misfits in random treatment assignment

Alvaro Carril
University of Chile
Santiago, Chile
acarril@fen.uchile.cl

Abstract. In this article, I discuss the "misfits" problem, a practical issue that arises in random treatment assignment whenever observations cannot be neatly distributed among treatments. I also introduce the `randtreat` command, which performs random assignment of unequal treatment fractions and provides several methods to deal with misfits.

Keywords: st0490, randtreat, random assignment, misfits, randomized control trial

1 Introduction

Random treatment assignment is presented as a simple exercise in theory, but practitioners know that there are usually several subtleties to deal with. In particular, when the number of observations in a given stratum is not a multiple of the number of treatments to be assigned, then one has to deal with the remainder observations—the "misfits"—while trying to maintain treatment allocation balance both within and across strata.

When one deals with unequal treatment fractions, the misfits problem arises whenever the number of observations in a given stratum is not a multiple of the least common multiple (LCM) of those fractions' denominators. This generalizes the issue as discussed by Bruhn and McKenzie (2011).

While the misfits problem may seem trivial, it can become significantly large when misfits add up across strata. If this is not taken into account, one may observe covariate imbalance even in variables for which one has stratified. Additionally, misfits may be systematically assigned to one particular treatment, leading to a deviation from the intended treatment proportions.

In this article, I present a simple command, `randtreat`, that performs random treatment assignment. It can handle multiple treatments and unequal treatment fractions, both of which are usually encountered in randomized control trials (RCTs). Stratified randomization can be achieved by optionally specifying a variable list that defines multiple strata. `randtreat` performs all of these tasks by marking misfit observations and provides several methods to deal with these observations.

The rest of this article is structured as follows: Section 2 explains the misfits problem and characterizes how and to what extent it can harm treatment allocation balance.

Section 3 details how `randtreat` handles misfits conceptually, and section 4 presents `randtreat` itself.

2 Problems with treatment assignment

The practical problem I discuss can be conceptually divided in two: the basic misfits problem and the unequal treatment fractions problem. Both are related; in fact, the latter only generalizes the former. Note that the "random" part of treatment assignment is irrelevant to these issues, so I omit it from the discussion in the beginning.

2.1 Misfits

The basic misfits problem arises whenever the size of the sample (or a given stratum) is not a multiple of the number of treatments to be allocated.[1] Whenever this happens, there will be some remaining observations, and there will be no obvious way to assign them to any of the available treatments.

This situation can happen often in practice and is common among researchers that conduct RCTs. Because RCTs are being increasingly used in a wide variety of fields, the need to rigorously and transparently deal with this problem also increases. However, to the best of my knowledge, few researchers have handled misfits explicitly. David McKenzie's World Bank blog post, written jointly with Miriam Bruhn (Bruhn and McKenzie 2011), is the most systematic analysis of the issue I have found, and the usage of the word "misfit" is due to them.[2]

It helps to have a concrete example to understand the issue. Consider allocating 3 treatments in a sample of 20 observations. Of course, one cannot assign 3 treatments evenly across 20 observations, but the usual algorithms for treatment assignment do not deal with this situation. For instance, consider using `egen`'s `cut()` function:

```
. set obs 20
. gen id = _n
. egen treatment = cut(id), group(3)
```

In this example, we know that up to 18 observations can be evenly allocated in a way that preserves treatment allocation fractions, namely, one-third for each treatment. However, this method of treatment assignment—or others like it—do not deal with this issue because they divide the whole sample (or strata) crudely, using the `floor()` or `ceil()` function either implicitly (as in this example) or explicitly. Because up to 18 observations can be evenly allocated, we will have 2 misfit observations regardless of whether they are explicitly accounted for. The latter case might produce problems in our assignment, as will be discussed in section 2.3.

1. Throughout this article, I adhere to the convention of referring to any value of the `treatment` variable as "treatment", because whether that value actually means "treatment" or "control" in the context of an experimental study is irrelevant. That being said, the `treatment` variable generated by `randtreat` takes nonnegative integer values, and 0 is usually assumed to mark the control group.
2. When formalizing the concept, I coined a woefully politically incorrect term for it, so I preferred using Bruhn and McKenzie's term.

Even though this is an overly simple example, it helps us to understand the fundamental problem. Although 1 sample with 20 observations can be rare, it can be common to have multiple strata of that size. Because each stratum is subject to an independent treatment assignment, the misfits problem applies to all stratum, thus multiplying the potential number of misfits in the whole sample. Section 2.4 explores this point further.

2.2 Unequal treatment fractions

A common requirement in real-world RCTs is that treatment arms be of unequal proportions; that is, different fractions of the sample should be assigned to each treatment. In this case, whenever the LCM of those fractions' denominators is not a divisor of the number of observations in any given stratum, we will have misfits in that stratum.

For example, consider a sample of 21 observations where half are assigned to control and then 3 different treatments are assigned in equal proportions among the other half. The naïve way of doing this would be to simply divide the sample in half, assigning the first half to control and then one-sixth to each treatment. A simple method for achieving that would be

```
. clear
. set obs 21
. generate treatment = .
. replace treatment = 0 if _n <= _N/2
. replace treatment = 1 if _n >  _N/2   & _n <= _N*2/3
. replace treatment = 2 if _n >  _N*2/3 & _n <= _N*5/6
. replace treatment = 3 if _n >  _N*5/6
```

It is evident that one cannot exactly assign these treatments in the desired proportions. There are different ways of handling the rounding of observations, but again, if the number of observations is not a multiple of the LCM of the treatment fractions' denominators, there will be misfit observations. In the above example, the corresponding LCM is 6, so up to 18 observations can be neatly allocated, while 3 misfit observations will have to be dealt with.

2.3 Why misfits are a problem

Until now, one might have wondered why we should take special care in handling misfits. After all, even though the method of crudely cutting the sample (or strata) and rounding up does not account for misfits, their allocation to treatments is still random because the sorting of observations is random. As long as our chosen method assigns every observation to a treatment, we should be fine, right?

Well, not always. One problem we might encounter is that an algorithm that does not account for misfits might systematically assign them to one particular treatment in every stratum, thus unbalancing the desired treatment proportions. This is the most direct problem that arises when using code that seems correct but inadvertently assigns misfits without taking them into account.

To be more concrete, let's consider `bpwide.dta`, which has fictional blood pressure data for 120 patients. Suppose we want to randomly allocate three treatments, stratifying by sex and age group. This setup has 6 strata, each with exactly 20 observations:

```
. sysuse bpwide, clear
. egen stratum = group(sex agegrp)
```

We know in advance that this configuration will produce 2 misfits per stratum for a total of 12 misfit observations. We could then proceed to perform a random treatment assignment with the following code:

```
. clear
. set seed 1102
. generate rannum = uniform()
. bysort stratum: egen rank = rank(rannum)
. generate treatment = .
. bysort stratum: replace treatment = 0 if rank <= _N/3
. bysort stratum: replace treatment = 1 if rank > _N/3 & rank <= 2*_N/3
. bysort stratum: replace treatment = 2 if rank > 2*_N/3
```

Tabulating `treatment` shows that even though we specified that each treatment had to have one-third of the observations, the control group (`treatment==0`) is actually under-represented because, in each stratum, the two misfits were systematically assigned to `treatment==1` and `treatment==2`.

This deviation from the intended treatment fractions may prove to be a problem in several ways. First, it can negatively affect the experiment's statistical power, because the groups are not balanced. Also, the treatment distribution may be unfeasible because treatments are usually more expensive than controls and the number of treatments available is restricted. There may also be political or ethical constraints that do not permit any unbalance in the treatment distribution.

Moreover, by failing to account for the misfits, we could inadvertently assign them in a way that harms the balance of our original stratification. For example, if we have stratified by sex and age group, our algorithm could assign all misfit women to control and all misfit men to treatment. This systematic misallocation of misfits may significantly unbalance any stratified assignment, defeating its purpose.

The misfits problem could be negligible in some simple experimental setups. However, in experiments with multiple treatments, unequal treatment fractions and various strata, the number of misfits and the problems associated with them may escalate very quickly. Because the number of misfits is crucial for assessing the severity of the problem they may cause, we now examine this matter.

2.4 Characterizing the number of misfits

I have described simple treatment assignments that produce a certain number of misfit observations and analyzed the ways those misfits could be a problem. It should be evident by now that whether misfits are going to be a problem in any treatment assignment—and to what extent—depends on their relative number, so it is interesting to analyze in further detail exactly how many misfits a given assignment can yield.

I first consider the case with equal treatment fractions discussed in section 2.1. We can see that for any number T of treatments we want to assign, we need the sample size to be divisible by T to not have misfits. If that is not the case, then the number of misfits will be equal to the remainder of the division of the sample size over T. For example, when assigning three treatments in a sample with nine observations, we will have no misfits. If the sample size was 10, we would get 1 misfit; if the sample size was 11, we would have 2 misfits. However, if the sample size was 12, then again, we would have no misfits. This situation holds true for each stratum in which the assignment is carried out.

The intuitive reasoning explained above can be formalized as follows: Let the strata of a sample be indexed by s, with $s = 1, \ldots, S$. In an assignment with multiple treatments of equal proportions, we will see that

$$m'_s = v_s \bmod T \quad \forall s$$

where m'_s is the number of misfits in stratum s, v_s is the number of observations (or "size") of stratum s, and T is the number of treatments. So it follows that

$$0 \leq m'_s \leq T - 1 \quad \forall s$$

Therefore, with equal treatment fractions, the total number of misfit observations in a sample with S strata is simply the sum of m'_s over s,

$$0 \leq \sum_{s=1}^{S} m'_s \leq S(T-1) \tag{1}$$

which means that, with equal treatment fractions, the total number of misfits in a sample has an upper bound of $(T-1)$ times the number of strata.

To put this result in perspective, in a sample with two treatments and no strata, this upper bound is 1, which will almost always be negligible. On the other hand, in a sample with 6 strata and 4 treatments, the upper bound is 18, which might be an issue in some experimental setups.

These calculations can be generalized to allow for unequal treatment fractions, as discussed in section 2.2. This case is very common in real-world RCTs, and, as we will see, the number of misfits produced in these setups can be substantially large.

Again, let T be the total number of treatments, with each treatment indexed by $t = 1, \ldots, T$. Denote treatment t's allocation fraction as a_t/b_t. Let J be equal to the LCM of the treatment fractions' denominators; that is,

$$J = \operatorname{lcm}(b_1, \ldots, b_T)$$

Now, let m_s be the number of misfits with unequal treatment fractions in any given stratum s. We see that

$$m_s = v_s \bmod J \quad \forall s$$

where v_s is the number of observations in stratum s. This means that the number of misfits in each stratum is equal to the remainder resulting from the division of the number of observations in that stratum and the least common denominator of the treatment allocation fractions. For instance, in the example presented in section 2.2, we see that $J = \text{LCM}(2, 6, 6, 6)$, so $m_1 = 21 \bmod 6 = 3$.

To see that the case with unequal treatment fractions is a generalization of the case with equal fractions, we must consider that by defining $T =$ treatments, we implicitly defined treatment fractions' denominators $b_1, \ldots, b_T = T \ \forall t$. In that case, the LCM of those denominators is also T, so $J = T$. From number theory, we know that this particular case also represents the lower bound for J; that is, $J \geq T$.

In this general case, we see that the number of misfits in any particular stratum has an upper bound of $J - 1$; that is,

$$0 \leq m_s \leq J - 1 \quad \forall s$$

Given this result, and recalling that $m'_s \leq J - 1 \ \forall s$ and that $J \geq T$, we see that

$$m'_s \leq m_s \quad \forall s$$

meaning that the number of misfits in the generalized case of unequal treatment fractions is at least as big as the number of treatments when dealing with the particular case of equal treatment fractions.

Finally, the total number of misfits is equal to the sum of m_s over s, so we see that

$$0 \leq \sum_{s=1}^{S} m_s \leq S(J - 1)$$

which means that the total number of misfits in any given sample has an upper bound equal to the number of strata multiplied by the least common denominator of the treatments allocation fractions minus one. This result generalizes the analogue obtained in (1) for equal treatment fractions.

Again, let's put this result in perspective. If we consider a sample with 120 observations and a treatment allocation with 5 strata and treatment fractions $1/2, 1/3$, and $1/6$, we see that the total number of misfits may be up to 25, which is a considerable percentage of the total number of observations.

More extreme cases are certainly possible. With large enough treatment allocation fractions' denominators, for example, $19/60$ and $41/60$, we have $J = 60$. This means that if after stratifying for some variables, we find that a stratum has fewer than 60 observations, then all those observations are going to be misfits. In these situations, dealing with misfits is not only an additional precaution but also an important necessity.

3 Dealing with misfits

Now that we know when the misfits problem arises and to what extent it represents a threat to a treatment assignment, we must consider methods of handling it. Because

misfits are inherent to the characteristics of a particular sample and treatment allocation design, the problem cannot be completely overcome. Nevertheless, once misfits are accounted for, we can consider various methods to handle them, each one with different (and somewhat symmetrical) advantages and disadvantages.

However, before exploring methods to deal with misfits, I introduce the randpack, a device that is fundamental to understanding how `randtreat` handles treatment assignment to account for misfits.

3.1 The randpack

Because we have established that the misfits problem's most general case arises when unequal treatment fractions are specified, any method to deal with misfits has to be conceived in a way that handles these types of treatment allocations.

I now introduce the basic device `randtreat` uses to perform random treatment assignment, the randpack.[3] The randpack is my invention and exists only to differently conceptualize how a random treatment assignment is performed, both theoretically and practically, and how it can account for misfits.

To understand what the randpack is, consider a simple treatment assignment where half a sample with nine observations must be assigned to treatment and the other half to control. The usual way of thinking about this assignment is as if it were a partitioning of the whole sample in half, assigning each half of the observations to a treatment status and letting some kind of rounding take care of the misfit.

However, we can also carry out that treatment assignment in a repeating pattern. We start by marking the first observation as control and the second one as treated. Then, we repeat this pattern for the third and fourth observations, and so on. After the eighth observation has been marked as treated, we will not be able to repeat the full pattern (that is, one control and one treated), so we mark the last observation as a misfit.

In a dataset, the repeating pattern just described can be considered a repeating sequence of integers. If we adhere to the convention of assigning a 0 to control observations and a 1 to treated ones, we see that the pattern just described is equivalent to the sequence $(0, 1)$. The treatment variable will repeat this sequence until all observations have a value assigned to them, very much like `egen`'s `fill()` function. However, if the sequence cannot be fully repeated (that is, the number of integers in the sequence is greater than the number of unassigned observations), then all remaining observations are marked as misfits.

Both approaches to treatment assignment are represented below. Treatment assignment by simple partition is represented in column `A`, while treatment assignment by repeating a sequence of integers is represented in column `B`.

3. After coming up with this (rather unimaginative) name, an anonymous referee pointed out that there is also an R package called `randPack`. However, there is no relationship between that R package and this Stata program.

```
n    A    B
1    0    0
2    0    1
3    0    0
4    0    1
5    ?    0
6    1    1
7    1    0
8    1    1
9    1    .  <-- misfit
```

The repeating sequence of integers used to assign treatment statuses in column `B` (that is, the sequence (0,1)) is what I refer to as the randpack. Notice that in column `A`, the fifth observation has a question mark, indicating it could be assigned either a 0 or a 1 depending on the rounding method. Conversely, one advantage of using a randpack is that it marks misfit observations mechanically.

Notice how the number of integers in the randpack is directly linked to the number of misfits that a given treatment allocation can give rise to. In what follows, I will formalize this intuition, detailing how to construct a randpack for various different assignment setups and establishing the relationship between that randpack and the number of misfits we will have to handle.

How to construct the randpack

When one assigns treatments of equal proportions, it should be immediately obvious that the number of integers in the randpack must be equal to the number of treatments to be assigned. For example, to assign 4 treatments in equal proportions, we can use the randpack $(0, 1, 2, 3)$. Notice that each treatment's code appears only once in the randpack because each must be equally represented.

The randpack's structure is more interesting if we analyze assignments with unequal treatment fractions, as discussed in section 2.2. For instance, consider a setup where two-thirds of the observations are controls and one-third is treated. There are only 2 treatment statuses, so we know the randpack will contain only integers 0 and 1. However, to preserve the specified fractions (that is, 2/3 and 1/3), we must ensure that the number of integers in the randpack is equal to the LCM of the fractions' denominators, which is 3. To construct the randpack, we must repeat each treatment code in a way that satisfies the desired treatment fractions, so the randpack has to have 2/3 of controls and 1/3 of treated, which is $(0, 0, 1)$.

In other words, the number of times each treatment code is contained in the randpack is given by the product of the code's corresponding allocation fraction with the total number of elements of the randpack. In the example above, code `0` (control) is contained $2/3 \cdot 3 = 2$ times and code `1` is contained $1/3 \cdot 3 = 1$ time.

More generally, the randpack is the smallest sequence (or vector) of treatment codes that maintains the desired treatment allocation fractions. The total number of treatment codes contained in the randpack is equal to the LCM of those fractions' denomi-

nators, and each treatment code must appear in the randpack a number of times equal to the product of its allocation fraction multiplied by the LCM of all the fractions' denominators. This sequence would be $(0, 1, 1)$ in a treatment assignment where 1/3 of the observations are controls and 2/3 are treated. An assignment where half the observations are controls and the other half are divided evenly among 3 treatments will have a sequence like $(0, 0, 0, 1, 2, 3)$, and so on.

I present a formal construction of the randpack in matrix notation in the appendix.

3.2 How is the randpack related to misfits?

The randpack has the virtue of connecting the characteristics of a treatment assignment to the number of misfit observations that assignment can yield. To see this, consider an allocation of 3 treatments with the following distribution: `treatment==0` (control) to 1/2 of the observations, `treatment==1` to 1/3 of the observations, and `treatment==2` to 1/6 of them.

First, the number of elements in the randpack is key because it indicates how many misfit observations per stratum the assignment can yield. In particular, the number of misfits in each stratum is strictly less than the number of elements in the randpack. As we have seen, the number of elements in the randpack is equal to the LCM of the treatment assignment fractions' denominators. So in the setup just described, the LCM of the fractions—hence, the number of elements of the randpack—is 6, which means that these treatment allocation fractions can produce as many as 5 misfits per stratum.

Once the randpack is defined, it is used to assign a treatment status to observations. Because these observations are already randomly sorted, the elements inside the randpack do not need to be.[4] The program's algorithm fills out the strata sequentially with the elements of the randpack, but if a whole randpack cannot fit in the remaining observations of a given stratum, then those observations' treatment status remains missing. This marks the misfits.

3.3 Methods for dealing with misfits

When choosing a method to deal with misfits, keep in mind that you cannot completely avoid them because their existence is inherent to the sample and the parameters of the desired treatment allocation. You must understand the tradeoffs of each method of handling them and apply the one that better suits the research requirements and constraints.

In stratified treatment assignment, the first question that arises is whether to deal with misfits by stratum or globally. That is, should we go into each stratum and deal with misfits locally, or should we pool all misfits across strata and deal with them globally?

4. Of course, one can also achieve random assignment by randomizing the elements inside the randpack and not the observations.

One approach is randomly allocating the misfits within each stratum. This method ensures that the difference in the number of misfits allocated to the treatments is not greater than one. While this ensures balance within strata, it may lead to treatment unbalance at the coarser levels. This method corresponds to the `misfits(strata)` option in the program.

The opposite approach is to group all misfits in a new "stratum" of their own and then randomly allocate treatments within it. The advantage of this method is that it preserves treatment fractions globally in the sample, because the final number of unbalancing misfits (the misfits of the misfits, so to speak) cannot be larger than the number of elements in the randpack. The downside is that this method does not guarantee balance within strata. This method corresponds to the `misfits(global)` option in the program.

Both approaches can be generalized to account for unequal treatment fractions. The main idea behind this is to assign misfits, either by stratum or globally, in a way that accounts for the probability of being assigned to any treatment, based on the specified fractions. This is achieved by shuffling the elements of the randpack and using it to sequentially fill out the misfit observations. Evidently, these variants make no difference if the specified treatment fractions are equal.

If done by stratum, this approach does not prevent treatments being allocated more than once within the stratum's misfits, but misfits are assigned to treatments with a weighted probability based on the treatment fractions. While this may lead to unbalanced treatments within strata, it can result in better global balance of treatments when dealing with unequal allocation fractions. This method corresponds to the `misfits(wstrata)` option in the program. Note that `randtreat` shuffles the randpack for each stratum with misfits to avoid the initial shuffle unbalancing the weighted allocation of misfits.

Finally, if misfits are assigned globally and unequal treatment fractions are specified, a weighted option is usually preferable. This will assign all misfits globally to treatments in accordance to the distribution dictated by the treatment fractions. Again, `randtreat` will shuffle the elements of the randpack and use it to assign treatments to all pooled misfits sequentially. This method can be used with the `misfits(wglobal)` option.

4 The randtreat command

4.1 Syntax

The `randtreat` command has been developed under Stata 11. Its syntax is

```
randtreat [if] [in], generate(newvar) [replace setseed(#)
    strata(varlist) multiple(#) unequal(fractions)
    misfits(missing|strata|wstrata|global|wglobal)]
```

4.2 Description

`randtreat`'s purpose is twofold: to easily randomize multiple unequal treatments across strata and to provide methods to deal with misfits. `randtreat` presumes that the current dataset corresponds to units (for example, individuals, firms, etc.) to be randomly allocated to treatment statuses.

When run, it creates a new variable encoding treatment status, which is randomly assigned. Although `randtreat` defaults to two treatments, more equally proportioned treatments can be specified with the `multiple()` option. Alternatively, multiple treatments of unequal proportions can be specified with the `unequal()` option. A stratified assignment can be performed using the `strata()` option.

Whenever the number of observations in a given stratum is not a multiple of the number of treatments or the LCM of the treatment fractions, then that stratum will have misfits, that is, observations that cannot be neatly distributed among the treatments. Misfits are automatically marked with missing values in the `treatment` variable, but `randtreat` provides several methods to deal with them. The method can be specified with the `misfits()` option.

4.3 Options

Treatment variable

`generate(`*newvar*`)` creates a new variable encoding randomly assigned treatment status. Treatment values are consecutive nonnegative integers. `generate()` is required.

`replace` replaces the treatment variable if it already exists.

`setseed(#)` specifies the initial value of the random-number seed used to assign treatments. It can be set so that the random treatment assignment can be replicated.

Assignment parameters

`strata(`*varlist*`)` performs a stratified allocation on the variables in *varlist*. If specified, the random assignment will be carried out in each stratum identified by the unique combination of the *varlist* variables' values. Notice that this option is almost identical to using `by` (see [D] **by**), except that the command is not independently run for the specified variables, because global existence of misfits across strata must be accounted for.

`multiple(#)` specifies the number of treatments to be assigned. The default (and minimum) is `multiple(2)`: one control group and one treatment group, unless the `unequal()` option is specified, in which case this option is redundant and should not be specified. All treatments will be allocated in equal fractions. For example, specifying `multiple(5)` means each treatment will be assigned to 20% of the observations.

`unequal(`*fractions*`)` specifies unequal fractions for treatments. Each fraction must be of the form a/b and must be separated by spaces. Each fraction must belong in $(0, 1)$, and the sum must be equal to 1. For example, specifying `unequal(1/2 1/4 1/4)` will randomly assign half the observations to the control group and then divide evenly the rest of the observations among two treatment groups.

Notice that this option implicitly defines the number of treatments. For example, in the aforementioned specification, we implicitly defined three treatments. So when `unequal()` is specified, `multiple(#)` is redundant and should be avoided.

`misfits(missing|strata|wstrata|global|wglobal)` specifies a method to deal with observations that cannot be neatly distributed among treatments, as discussed in section 2. The methods are the following:

`missing` leaves misfit observations as missing values in the treatment variable, so one can later deal with misfits as one sees fit. The default is `misfits(missing)`.

`strata` randomly allocates misfits independently across all strata, without weighting treatments as specified in `unequal()`. This method prioritizes balance of misfits' treatment allocation within each stratum (they cannot differ by more than 1) but may harm original treatment fractions if the number of misfits is large.

`wstrata` randomly allocates misfits independently across all strata, weighting treatments as specified in `unequal()`. This ensures that the fractions specified in `unequal()` affect the within-distribution of treatments among misfits, so overall balance of unequal treatments should be (almost) attained. However, this method does not ensure the balance of misfits' treatment allocation within each stratum (they could differ by more than 1).

`global` randomly allocates all misfits globally, without weighting treatments as specified in `unequal()`. This method prioritizes global balance of misfits' treatment allocation (they cannot differ by more than 1) but may harm original treatment fractions if the number of misfits is large.

`wglobal` randomly allocates all misfits globally, weighting treatments as specified in `unequal()`. This ensures balance at the global level and also respects unequal fractions of treatments, even when the number of misfits is large. However, this method does not ensure the global balance of misfits' treatment allocation (they could differ by more than 1). The downside is that this method could produce even greater imbalance at the finer level (in each stratum), especially if the number of misfits is relatively large.

Note that if all treatments have equal fractions, then the weighted variants of the methods are equivalent to the unweighted ones. This is also true when the assignment is not stratified global and local methods are equivalent.

4.4 Output

`randtreat` preserves the active dataset and the current sorting of its observations. It creates a new variable with randomly assigned treatment values, which range from 0 to $T-1$, where T is the specified number of treatments (see the `multiple()` option) or the number of treatment fractions (see the `unequal()` option).

Regardless of the method chosen to deal with misfits (see the `misfits()` option), the command always displays the number of misfits that the specified treatment allocation produces.

5 Acknowledgments

I am indebted to several "random helpers" at the Random Help Google user group, as well as Nick Cox from Statalist, who provided advice and code snippets that I used for this program. I also thank colleagues at the J-PAL LAC office, especially Olivia Bordeu and Diego Escobar, for valuable discussion on the implementation and application of `randtreat` in real-world RCTs. Finally, I am grateful for the useful comments of an anonymous referee, which helped to improve the structure and clarity of this article. All remaining errors are my own.

6 References

Brualdi, R. A. 2006. Algorithms for constructing $(0,1)$-matrices with prescribed row and column sum vectors. *Discrete Mathematics* 306: 3054–3062.

Bruhn, M., and D. McKenzie. 2011. The World Bank: Impact Evaluations Blog: Tools of the trade: Doing stratified randomization with uneven numbers in some strata. http://blogs.worldbank.org/impactevaluations/tools-of-the-trade-doing-stratified-randomization-with-uneven-numbers-in-some-strata.

About the author

Alvaro Carril is a research associate at J-PAL LAC, where he randomizes his choice of cereal every morning.

Appendix

Formal statement of the randpack

Given that the randpack is the central object used by `randtreat` to perform a treatment assignment, we need to understand exactly how it is constructed. All methods for dealing with misfits implemented within the command are, in a way, manipulations of the randpack. This appendix defines it in a more rigorous fashion.

Let T be the total number of treatments to be assigned, with each treatment indexed by $t = 1, \ldots, T$. Denote treatment t's assignment fraction as a_t/b_t. Now, let J be equal to the LCM of the fractions' denominators; that is,

$$J = \text{LCM}(b_1, \ldots, b_T)$$

which corresponds to what we have called the randpack's size.

The randpack is constructed as the product of two matrices, $\mathbf{A}$ and $\mathbf{B}$. First, let $\mathbf{A}$ be a $J \times T$ matrix such that

$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1T} \\ A_{21} & A_{22} & \cdots & A_{2T} \\ \vdots & \vdots & \ddots & \vdots \\ A_{J1} & A_{J2} & \cdots & A_{JT} \end{pmatrix}$$

where A_{jt} is either 1 or 0, with the number of 1s appearing in column t indicating how many times treatment t's code number features in the randpack. Additionally, let $\mathbf{B}$ be a T-dimensional vector such that

$$\mathbf{B} = \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_T \end{pmatrix}$$

where $B_t = t - 1$, which means $\mathbf{B}$ is a vector containing each treatment's code number.

While $\mathbf{B}$ is a simple vector, some effort is needed to construct $\mathbf{A}$, which is a (0–1)-matrix indicating the number of times each treatment's code number is repeated in the randpack. This is truly the key part of constructing the randpack.

To define a matrix like $\mathbf{A}$, I consider a simplified version of Ryser's algorithm as described by Brualdi (2006). First, let the row sum vector $\mathbf{R} = (r_1, r_2, \ldots, r_J)$ and the column sum vector $\mathbf{S} = (s_1, s_2, \ldots, s_T)$ be positive integral vectors that satisfy

$$r_1 + r_2 + \cdots + r_J = s_1 + s_2 + \cdots + s_T$$

Let $\mathcal{A}(\mathbf{R}, \mathbf{S})$ denote the class of all logical matrices for which the row sum vector equals $\mathbf{R}$ and the column sum vector equals $\mathbf{S}$. Let $A(\mathbf{R}; T)$ denote the perfectly nested $J \times T$ logical matrices with row sum vector $\mathbf{R}$ with the property that the 1s in each row occur in the initial positions. The Gale–Ryser theorem implies that all the matrices in $\mathcal{A}(\mathbf{R}, \mathbf{S})$ can be obtained from $A(\mathbf{R}; T)$ by shifting 1s in rows in ways that satisfy column sum vector $\mathbf{S}$, which is what interests us.

We assume now that $\mathbf{R}$ is constant and $r_j = 1\ \forall j$, so the above restriction can be rewritten as

$$\sum_{t=1}^{T} s_t = J$$

which means that $\mathbf{R}$ is a J-dimensional vector of all 1s.

Now, let $\mathbf{S}$ be defined as

$$\mathbf{S} = J\widetilde{\mathbf{S}}$$

where $\widetilde{\mathbf{S}}$ is the T-dimensional vector of treatment fractions; that is,

$$\widetilde{\mathbf{S}} = \begin{pmatrix} a_1/b_1 \\ a_2/b_2 \\ \vdots \\ a_T/b_T \end{pmatrix}$$

Ryser's is a recursive algorithm that begins with $A(\mathbf{R}; T)$ and shifts 1s to the right to achieve the desired column sums $s_T, s_{T-1}, \ldots, s_1$ in the following order:

1. Shift a 1 in s_T rows of $A(\mathbf{R}; T)$ to column T, starting from the lower rows.
2. The matrix left in columns $1, \ldots, T-1$ of $A(\mathbf{R}; T)$ is a matrix $A(\mathbf{R}'; T-1)$ with row sum vector determined by $\mathbf{R}$ and the 1s shifted. Now step 1 must be repeated but beginning with $A(\mathbf{R}'; T-1)$ and using s_{T-1} instead of s_T.

Using this algorithm, we can construct our matrix $\mathbf{A}$. Having already defined $\mathbf{B}$, we can now formally define the randpack, denoted as $\mathbf{x}$, as the matrix product $\mathbf{AB}$:

$$\mathbf{x} = \begin{pmatrix} (\mathbf{AB})_{11} \\ (\mathbf{AB})_{12} \\ \vdots \\ (\mathbf{AB})_{1J} \end{pmatrix}$$

Example

To help clarify how $\mathbf{x}$ is constructed, consider the recurring example in which the treatment fractions are 1/2, 1/3, and 1/6. We can immediately define

$$J = \text{LCM}(2, 3, 6) = 6$$

so

$$\mathbf{S} = 6 \begin{pmatrix} 1/2 \\ 1/3 \\ 1/6 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$$

Now, to construct $\mathbf{A}$ using that $\mathbf{R} = (1,1,1,1,1,1)$ and $\mathbf{S} = (3,2,1)$, we start with $A(\mathbf{R};T)$ and use Ryser's algorithm:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

So we finally see that $\mathbf{x} = \mathbf{AB}$ is

$$\mathbf{x} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 2 \end{pmatrix}$$