Correction of rounding, typing, and sign errors with the `deducorrect` package

Mark van der Loo, Edwin de Jonge and Sander Scholtus

April 9, 2011

Abstract

This vignette is unfinished. Version 1.0 of the package will contain a full vignette.

Since raw (survey) data usually has to be edited before statistical analysis can take place, the availability of data cleaning algorithms is important to many statisticians. In this paper the implementation of three data correction methods in R. The methods of this package can be used to correct numerical data under linear restrictions for typing errors, rounding errors, sign errors and value interchanges. The algorithms, based on earlier work of Scholtus, are described as well as implementation details and coded examples. Although the algorithms have originally been developed with financial balance accounts in mind the algorithms are formulated genericly and should find a wider range of application.

Contents

List of Algorithms

1 Introduction

Raw statistical data is often plagued with internal inconsistencies and errors which inhibit reliable statistical analysis. Establishment survey data is particularly prone to in-record inconsistencies, because the numerical variables contained in these data are usually interrelated by many mathematical relationships. Before statistical analysis can take place, these relationships have to be checked and violations should be resolved as much as possible. While establishing that a record violates certain relationships is straightforward, deciding which fields in a record contain the actual errors can be a daunting task. In the past, much attention has been paid to this decision problem, often using Fellegi and Holt's principle (Fellegi and Holt, 1976) as the point of departure. This principle states that for non-systematic errors, and with no information on the cause of errors, one should try to make a record consistent by changing as few variables as possible.

This principle precludes using the data available in the (possibly erroneous) fields to detect and correct the error. In certain cases, naively applying Fellegi and Holt's principle will yield consistent records with nevertheless faulty data. As an example, consider a survey record with three variables x, y and z, which have to obey the relationship $x = y - z$. Such relationships frequently occur in financial profit-loss accounts. If a record happens to have values such that $x = z - y$, then Fellegi and Holt's principle suggests that either the numerical value of x, y or z should be adapted in such a way that the relationship holds, while the values in the record suggest that the values in fields y and z might have been interchanged. Swapping the values of z and y therefore seems a reasonable solution although it formally means changing two values.

1.1 Deductive correction

We use the term deductive correction to indicate methods wich use information available in inconsistent records to deduce and solve the probable cause of error. Recently, a number of algorithms for deductive correction have been proposed by Scholtus (2008, 2009). These algorithms can solve problems not uncommon in numerical survey data, namely

- Rounding errors.
- Simple typing errors.
- Sign swaps and/or value interchanges.

The algorithms focus on solving problems in records with linear relationships, which can be written in any (combination of) the forms

$$\begin{aligned} Ax &= b \quad (1)\\ Ax &< b \quad (2)\\ Ax &\leq b \quad (3)\\ Ax &> b \quad (4)\\ Ax &\geq b \quad (5) \end{aligned}$$

Here, every A is a matrix, x a numerical data record and b a constant vector. Every row of the combined matrix $[A, b]$ represents one linear restriction. In data-editing literature the restrictions imposed on records are often called edit rules, or edits in short. If an edit describes a relationship between a number of variables $\{x_j\}$, we say that the edit *contains* the variables $\{x_j\}$. Conversely, when x_j is part of a relationship defined by an edit we say that x_j *occurs* in the edit.

In this paper, we describe the `deducorrect` package for R (R Development Core Team, 2011), which implements (slight) generalisations of the algorithms proposed by Scholtus (2008, 2009). The purpose of this paper is to provide details on the algorithms and to familiarize users with the syntax of the package. The correction algorithms in the package report the results in a uniform matter. Section 1.2 provides details on the `deducorrect` output object which stores information on corrected records, applied corrections, and more. Sections 2, 3 and 4, provide details on the classes of problems that may be treated with the package, an exposition of the algorithms used and coded examples with analysis of the results. It is also shown how the examples from Scholtus (2008) and Scholtus (2009) can be treated with this software. The package requires that linear relationships are defined with the `editrules` package (de Jonge and van der Loo, 2011). Unless noted otherwise, all R-code examples in this paper can be executed from the R commandline after loading the `deducorrect` and `editrules` package.

1.2 The `deducorrect` object and `status` values

Apart from the corrected records, every `correct-` function of the `deducorrect` package returns some logging information on the applied corrections. Information on applied corrections, a status indicator per record, a timestamp and user information are included and stored uniformily in a `deducorrect` object. See Table 1 for an overview of the contents of a `deducorrect` object. Because of the large amount of information in a `deducorrect` object, the contents are summarized for printing to screen. In the example below, we define one record of data, a linear restriction in the form of an `editmatrix`, and apply the `correctSigns` correction method[1].

[1]sometimes extra brackets are included to force R to print the result

Table 1: Contents of the `deducorrect` object. All slots can be accessed or reassigned through the `$` operator.

`corrected`	The input data with records corrected where possible.
`corrections`	A `data.frame` describing the corrections. Every record contains a row number, labeling the row in the input data, a variable name of the input data, the old value and the new value.
`status`	A `data.frame` with at least one column giving treatment information of every record in the input data. Depending on the `correct` function, some extra columns may be added.
`timestamp`	The date and time when the `deducorrect` object was created.
`generatedby`	The name of the function that called `newdeducorrect` to create the object.
`user`	The name of the user running R, deduced from the environment variables of the system using R.

```
> (d <- data.frame(x = 1, y = 0, z = 1))

  x y z
1 1 0 1

> require(editrules)
> E <- editmatrix("x==y-z")
> sol <- correctSigns(E, d)
> sol

 deducorrect object generated by 'correctSigns' on Sat Apr  9 09:52:28 2011
 slots:  $corrected, $corrections, $status, $timestamp, $generatedby, $user

 Record status:
  invalid   partial corrected     valid       Sum
        0         0         1         0         1

 Variables corrected:
  x Sum
  1   1
```

The individual components of `sol` can be retrieved with the dollar-operator. The slot `corrected` is the same as the input data, but with corrected records, where possible:

```
> sol$corrected

   x y z
1 -1 0 1
```

Table 2: The number of equalities n and inequalities m violated by an edit, before and after treatment with one of the correct-functions of `deducorrect`. The label N/A indicates that this exit status does not occur in the function.

Before		After		status		
Eqs	Ineqs	Eqs	Ineqs	`correctSigns`	`correctRounding`	`correctTypos`
0	0	0	0	valid	valid	valid
0	m	0	m	invalid	invalid	invalid
n	0	n	0	invalid	invalid	invalid
n	0	$< n$	0	N/A	partial	partial
n	0	0	0	corrected	corrected	corrected
n	m	n	m	invalid	invalid	invalid
n	m	$< n$	0	N/A	partial	partial
n	m	$< n$	$< m$	N/A	partial	partial
n	m	0	0	corrected	corrected	corrected

The applied corrections are stored in the `corrections` slot.

```
> sol$corrections

  row variable old new
1   1        x   1  -1
```

Every row in `corrections` tells wich variable in which row of the input data was changed, and what the old and new values are. The `status` slot gives details on the status of the record.

```
> sol$status

     status weight degeneracy nflip nswap
1 corrected      1          2     1     0
```

The first column is an indicator which can take five different values, indicating whether validity could be established, and/or if the record could be (partially) corrected by the method which created the deducorrect object. See Table 2 for details. The rest of the columns depend on the function which created the object and can provide more details on the chosen solutions. These are described in the following sections.

1.3 Balance accounts and totally unimodular matrices

Most algorithms described here have been designed with financial balance accounts in mind. The balance accounts encountered in establishment surveys mostly involve integer records since financial amounts are usually reported in currency (kilo-)units. Therefore, linear editrules of the form

$$Ax = b \text{ with } A \in \{-1, 0, 1\}^{m\times n},\ x \in \mathbb{Z}^n,\ \text{ and } b \in \mathbb{Z}^m. \tag{6}$$

are frequently encountered. In all the examples of financial balance accounts encountered by the authors, the matrix A happened to be totally unimodular. A (not necessarily square) matrix is called *totally unimodular* when every square submatrix has determinant -1, 0, or 1. The scapegoat algorithm (Scholtus, 2008), which is used in the `correctRounding` function, requires A to be totally unimodular. See appendix B of Scholtus (2008) for a further discussion of total unimodularity. The `deducorrect` package offers the function `isTotallyUnimodular` which checks if a matrix is totally unimodular. The algorithm follows a recursive procedure given below.

```
procedure isTotallyUnimodular(A)
    A ←reduceMatrix(A)
    if A = ∅ then
        return TRUE
    else if Each column of A has exactly 2 nonzero elements then
        return hellerTompkins(A)
    else
        𝓑 ←raghavachari(A)
        if Every B ∈ 𝓑 isTotallyUnimodoular(B) then
            return TRUE
        else
            return FALSE
        end if
    end if
end procedure
```

Here, reduceMatrix iteratively removes all rows and columns of A which have a single nonzero element (an operation of $\mathcal{O}(n)$ in the number of columns and rows). When possible, the criterium of Heller and Tompkins (1956), which is $\mathcal{O}(2^n)$ in the number of columns is used to determine unimodularity. If this is not possible, a series of smaller matrices $\mathcal{B}$ is derived with the method of Raghavachari (1976). Every matrix in $\mathcal{B}$ is subsequently checked for total unimodularity by calling isTotallyUnimodular. In the worst case, Rachavachari's method must be called recursively and checking for unimodularity is $\mathcal{O}(n!)$ in the number of columns. In practical applications A is often fairly sparse and only a small portion of A has to be treated with the Rachavachari method.

2 `correctRounding`

2.1 Area of application

This function can be used to correct records which violate linear equality restrictions because of rounding errors in one or more variables. The linear

Algorithm 1 Scapegoat algorithm

Input: Equality restriction matrix A and constant vector b, record x, rounding tolerance ε.

1: Remove rows from the system $Ax = b$ not satisfying $|A_{i\cdot}x - b_i| < \varepsilon$.
2: **if** $A \neq \varnothing$ **and** $||Ax - b|| > 0$ **then**
3: Randomly permute columns of A. Permute x and b accordingly.
4: Use QR decomposition to partition A columnwise in a square invertible matrix A_1 and remaining columns A_2. Partition x in x_1 and x_2, and b in b_1 and b_2 accordingly.
5: $x_1 \leftarrow A_1^{-1}(b - A_2x_2)$
6: Unpermute $[x_1, x_2]$
7: **end if**
8: Restore x by adding the previously removed elements.

Output: x

equality restrictions must be of the form

$$Ax = b \text{ with } A \in \{-1, 0, 1\}^{m\times n},\ x \in \mathbb{Z}^n, \text{ and } b \in \mathbb{Z}^m.$$

where A is a totally unimodular matrix (see Section 1.3), which can be tested with the function `isTotallyUnimodular`.

2.2 How it works

The `correctRounding` function uses the scapegoat algorithm described in Scholtus (2008) to suggest corrections for linear equality violations. Linear inequalities are ignored, except that corrections which cause new inequality violations are not accepted. The algorithms selects editrules violated by rounding errors. Rounding errors cause small deviations from equality and therefore deviations smaller than some ε (say, $\varepsilon = 2$) are assumed to stem from rounding errors. Next, a number of variables –called scapegoat variables– is selected randomly in such a way that rounding errors can be solved exactly and uniquely. If the chosen solution happens to cause new inequality violations, the solution is rejected and a new set of scapegoat variables is drawn. This is repeated at most k times. See Algorithm 1 for a consice description of the basic procedure (without checking for inequalities).

2.3 Examples

Here, we will reproduce the example of Scholtus (2009), Section 5.3.2. Consider an integer-valued record with 11 variables, subject to the rules:

```
> E <- editmatrix( c("X1 + X2 == X3"
+                   ,"X2 == X4"
+                   ,"X5 + X6 + X7 == X8"
```

```
+                    ,"X3 + X8 == X9"
+                    ,"X9 - X10 == X11"))
```

Consider also the following inconsistent record:

```
> (dat <- data.frame(t(c(12, 4, 15, 4, 3, 1, 8, 11, 27, 41, -13))))

  X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11
1 12  4 15  4  3  1  8 11 27  41 -13

> violatedEdits(E, dat)

       e1    e2   e3   e4   e5
[1,] TRUE FALSE TRUE TRUE TRUE
```

As reported by the `violatedEdits` function, this record violates editrules 1, 3, 4, and 5. Using R's built-in matrix operations, we may check which edits might be violated because of rounding errors:

```
> E %*% t(as.matrix(dat))

rules [,1]
   e1    1
   e2    0
   e3    1
   e4   -1
   e5   -1
```

which, in this case is the same since all violations fall within the limit we expect rounding errors. Repairing the record can be done with

```
> set.seed(1)
> sol <- correctRounding(E, dat)
> cbind(sol$corrected, sol$status)

  X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11    status attempts
1 12  4 16  4  3  1  8 12 28  41 -13 corrected        1

> sol$corrections

   row variable old new
X3   1       X3  15  16
X8   1       X8  11  12
X9   1       X9  27  28
```

Here, we used `set.seed` to make results reproducible. The result is not exactly the same as the solution found in the reference. Here, variables x_3, x_8 and x_9 have been adapted, while in the reference x_3, x_8 and x_9 were adapted. Since corrections are very small, smearing out the effect of adaptations over a number of variables is a reasonable option.

3 `correctTypos`

3.1 Area of application

This function can be used to correct typographical errors in an integer record violating linear equality constraints as in Eq. (6):

$$Ax = b \text{ with } A \in \{-1, 0, 1\}^{m\times n},\ x \in \mathbb{Z}^n,\ \text{ and } b \in \mathbb{Z}^m.$$

The algorithm was developed with sets of financial balance equations in mind, where these type of problems are very common. As far as inequalities are concerned, they are currently ignored by the algorithm, in the sense that no attempt is made to repair inequality violations. However, the algorithm does not allow solutions causing extra inequality violations.

Records which violate the equality restrictions are treated. There is an option `eps` which allows for a tolerance in checking if records should be treated. This way, records containing ony rounding errors can be ignored but do note that they will retrieve the status `valid`.

3.2 How it works

In short, the algorithm first computes a list of suggestions which correct one or more violated edits (Algorithm 2). The corrections not corresponding to a typographical error are removed, after which the set of suggestions that maximize the number of satisfied editrules is determined (Algorithm 3).

Suggestions are generated for the set of variables which *only* occur in violated edits since altering these variables will have no effect on already satisfied edits. For every variable x_j, define the matrix $A^{(j)}$ who's rows represent edits containing x_j. Suggestions $\tilde{x}_j^{(i)}$ for every row i of $A^{(j)}$ can be generated by solving for x_j:

$$\tilde{x}_j^{(i)} = \frac{1}{A_{ij}^{(j)}}\left(b_i - \sum_{j'\neq j} A_{ij'}^{(j)} x_j\right). \tag{7}$$

We keep only the unique suggestions, and reject solutions which are more than a certain Damerau-Levenshtein distance removed from the original value. The *Damerau-Levenshtein* distance d_{DL} between two strings s and t is the minimum number of character insertions, deletions, substitutions and transpositions necessary to change s into t or *vice versa* (Damerau, 1964; Levenshtein, 1966). The remaining set of suggestions $\{x_j^{(i)}\}$ will in general contain multiple suggestions for each violated edit i and multiple suggestions for each variable x_j. Using a tree search algorithm, a subset of $\{x_j^{(i)}\}$ is selected which maximizes the number of resolved edits. The tree search is sped up considerably by pruning branches which resolve the same edit multiple times or use multiple suggestions for the same variable.

Algorithm 2 Generate solution candidates

Input: Record x, a set of linear equality restrictions and a list of variables to `fixate`. A maximum Damerau-Levenshtein distance `maxdist`.

$L \leftarrow \varnothing$
Determine $J_0 = \{j : x_j$ occurs *only* in violated edits and not in `fixate`$\}$
for $j \in J_0$ **do**
    Determine the matrix $A^{(j)}$ of violated edits containing x_j and associated constant vector $b^{(j)}$
    for every row i of $A^{(j)}$ **do**
        $\tilde{x}_j^{(i)} \leftarrow \left(b_i^{(j)} - \sum_{j' \neq j} A_{ij'}^{(j)} x_{j'}\right) / A_{ij}^{(j)}$
        $L \leftarrow L \cup \tilde{x}_j^{(i)}$
    end for
end for
Remove $\tilde{x}_j^{(i)}$ from L for which $d_{\mathrm{DL}}(\tilde{x}_j^{(i)}, x_j) >$ `maxdist`

Output: List L of m unique solution suggestions for record x.

This algorithm generalizes the algorithms of Scholtus (2009) in the following two ways: first, the imposed linear restrictions are generalised from $Ax = 0$ to $Ax = b$. Secondly, the original algorithm allowed for a single *digit* insertion, deletion, transposition or substitution. The more general Damerau-Levenshtein distance used here treats the digits as characters, allowing for sign changing, which is forbidden if only digit changes are allowed. Also, by applying a standard Damerau-Levenshtein algorithm it is easy to allow for corrections spanning larger values d_{DL}. That is, one could allow for multiple typos in a single field. Moreover, the Damerau-Levenstein distance as implemented in the `deducorrect` package allows one to define different weights to the four types of operations involved, adding some extra flexibility to the method.

3.3 Examples

In this section we show the most important options of the `correctTypos` function. After a simple, worked-out example we reproduce the results in Chapter 4 of Scholtus (2009).

First, define a simple one-record dataset with an associated edit rule.

```
> dat <- data.frame(x = 123, y = 192, z = 252)
> (E <- editmatrix("z == x + y"))

Edit matrix:
    x  y z CONSTANT
e1 -1 -1 1        0

Edit rules:
e1 : z == x + y
```

Algorithm 3 Maximize number of resolved edits

Input: Record x, a list of linear equality restrictions and a list of solution suggestions $L = \{L_\ell = \tilde{x}_{j_\ell}^{(i_\ell)} : \ell = 1, 2, \ldots, m\}$

$k \leftarrow 0$
$s \leftarrow$ NULL
procedure TREE(x, L)
  if $L \neq \varnothing$ **then**
    TREE$(x, L \backslash L_1)$ ▷ Left branche: don't use suggestion
    $x_{j_1} \leftarrow L_1$ ▷ Right branche: use suggestion
    $L \leftarrow L \backslash \{x_{j_\ell}^{(i_\ell)} \in L : j_\ell = j_1$ **or** $x_{j_\ell}^{(i_\ell)}$ resolves the same edit as $L_1\}$
    TREE(x, L)
  else
    if Number of edits n resolved by x larger then k **then**
      $k \leftarrow n$
      $s \leftarrow x$
    end if
  end if
end procedure

Output: (partial) solution s, resolving maximum number of edits.

Obviously, the edit in `E` is not satisfied since $123+192 = 315$. As can be seen from the output of `editmatrix`, we have $b = 0$, so the correction candidates here are:

$$\tilde{x}^{(1)} = 0 - \frac{-1 \cdot 192 + 1 \cdot 252}{-1} = 60 \tag{8}$$

$$\tilde{y}^{(1)} = 0 - \frac{-1 \cdot 123 + 1 \cdot 252}{-1} = 129 \tag{9}$$

$$\tilde{z}^{(1)} = 0 - \frac{-1 \cdot 123 - 1 \cdot 192}{1} = 315 \tag{10}$$

The Damerau-Levenshtein distances between the candidates and their originals are given by:

$$d_{\mathrm{DL}}(\tilde{x}^{(1)}, x) = 3 \text{ (two substitutions and an insertion)} \tag{11}$$

$$d_{\mathrm{DL}}(\tilde{y}^{(1)}, y) = 1 \text{ (one transposition)} \tag{12}$$

$$d_{\mathrm{DL}}(\tilde{z}^{(1)}, z) = 3 \text{ (three substitutions)} \tag{13}$$

In this case, there is just one candidate with $d_{\mathrm{DL}} = 1$, solving the inconsistency with just one digit transposition. Running the record through `correctTypos` indeed finds the digit transposition:

```
> correctTypos(E, dat)$corrected

    x   y   z
1 123 129 252
```

Scholtus (2009) (Chapter 4) treats a series of examples which we will reproduce here. We consider a dataset with 11 variables, subject to the following edit rules.

```
> E <- editmatrix( c("x1 + x2 == x3"
+                   ,"x2 == x4"
+                   ,"x5 + x6 + x7 == x8"
+                   ,"x3 + x8 == x9"
+                   ,"x9 - x10 == x11"))
```

The following dataframe contains the correct record (`example 4.0`) as well as the manipulated erroneous records.

```
> dat

              x1  x2   x3  x4  x5 x6 x7  x8    x9  x10 x11
example 4.0 1452 116 1568 116 323 76 12 411  1979 1842 137
example 4.1 1452 116 1568 161 323 76 12 411  1979 1842 137
example 4.2 1452 116 1568 161 323 76 12 411 19979 1842 137
example 4.3 1452 116 1568 161   0  0  0 411 19979 1842 137
example 4.4 1452 116 1568 161 323 76 12   0 19979 1842 137
```

This `data.frame` can be read into R by copying the code from the `correct-Typos` help page. As can be seen, example 4.1 has a single digit transposition in x_4, example 4.2 has the same error, and an extra 1 prefixed to x_9, example 4.3 contains multiple extra errors (in x_5, x_6 and x_7 which cannot be explained bby simple typing errors. Finally, example 4.4 also has multiple errors which cannot all be explained by simple typing errors. This example has multiple solutions which solve an equal amount of errors.

The violated edit rules may be listed with the function

```
> violatedEdits(E, dat)

        e1    e2    e3    e4    e5
[1,] FALSE FALSE FALSE FALSE FALSE
[2,] FALSE  TRUE FALSE FALSE FALSE
[3,] FALSE  TRUE FALSE  TRUE  TRUE
[4,] FALSE  TRUE  TRUE  TRUE  TRUE
[5,] FALSE  TRUE  TRUE  TRUE  TRUE
```

Now, to apply as many typo-corrections as possible:

```
> sol <- correctTypos(E, dat)
> cbind(sol$corrected, sol$status)

              x1  x2   x3  x4  x5 x6 x7  x8    x9  x10 x11    status
example 4.0 1452 116 1568 116 323 76 12 411  1979 1842 137     valid
example 4.1 1452 116 1568 116 323 76 12 411  1979 1842 137 corrected
example 4.2 1452 116 1568 116 323 76 12 411  1979 1842 137 corrected
example 4.3 1452 116 1568 116   0  0  0 411  1979 1842 137   partial
example 4.4 1452 116 1568 116 323 76 12   0 19979 1842 137   partial
```

Our implementation finds the exact same solutions as in the original paper of Scholtus (2009). Also see this reference for a through analysis of the outcomes.

4 `correctSigns`

4.1 Area of application

This function can be used to solve sign errors and value swaps which cause linear equalities (Eq. 1) to fail. Possible presence of linear inequalities [Eq. (2)-(5)] are taken into account when resolving errors, but they are not part of the error detection process.

4.2 How it works

The function `correctSigns` tries to change the sign of (combinations of) variables and/or swap the order of variables to repair inconsistent records. Sign flips and value swaps are closely related since

$$-(x - y) = y - x, \tag{14}$$

These simple linear relations frequently occur in profit-loss accounts for example. Basically, `correctSigns` first tries to correct a record by changing one sign. If that doesn't yield any solution, it tries changing two, and so on. If the user allows value swaps as well, it starts by trying to correct the record with a single sign flip or variable swap. If no solution is found, a combination is tried, and so on. The algorithm only treats the variables which have nonzero coefficients in one of the violated rows of Eq. (1). Since the number of combinations grows exponentially with the number of variables to treat, the user is given some control over the volume of the search space to cover in a number of ways. First of all, the variables which are allowed to flip signs or variable pairs which may be interchanged simultaneously can be determined by the user. Knowledge of the origin of the data will usually give a good idea on which variables are prone to sign errors. For example, in surveys on profit-loss accounts, respondents sometimes erroneously submit the cost as a negative number. Secondly, the user may limit the maximum number of simultaneous sign flips and or value swaps that may be tested. This is controled by the `maxActions` parameter in Algorithm 4. The third option limiting the search space is to break when the number of combinations, given a number of actions to try becomes too large. This is controlled by the `maxCombinations` parameter in Algorithm 4.

To account for sign errors and variable swap errors which are masked by rounding errors, the user can provide a nonegative tolerance ε, so the set of equality constraints are checked as

$$|Ax - b| < \varepsilon, \tag{15}$$

Algorithm 4 Record correction for `correctSigns`

Input: A numeric record x, a tolerance ε. A set of equality and inequality constraints of the form

$$\begin{aligned} Ax - b &= 0 \\ Bx - c &\geq 0, \end{aligned}$$

A list `flip` of variables whos signs may be fliped, a list `swap` of variable pairs whos values may be interchanged, an integer `maxActions`, an integer `maxCombinations` and a weight vector.

Create a list `actions`, of length n containing those elements of `flip` and `swap` that affect variables that occur in violated rows of A.
Create an empty list S.
$k \leftarrow 0$
while $S = \varnothing$ **and** $k < \min(\texttt{maxActions}, n)$ **do**
    if not $\binom{n}{k} > \texttt{maxCombinations}$ **then**
        $k \leftarrow k + 1$
        Generate all $\binom{n}{k}$ combinations of k actions.
        Loop over those combinations, applying them to x. Add solutions obeying $|Ax - b| < \varepsilon$ and $Ax - c \geq 0$ to S.
    end if
end while
if not $S = \varnothing$ **then**
    Compute solution weights and choose solution with minimum weight. Choose the first solution in the case of degeneracy.
end if

where $|\cdot|$ indicates the elementwise absolute value. The default value of ε is the square root of machine accuracy which amounts to approximately 10^{-8} on a 32-bit architecture.

The purpose of this algorithm is to find and apply the minimal number of actions (sign flips and/or variable swaps) necessary to repair the record. It is not guaranteed that a solution exists, nor that the solution is unique. If multiple solutions are found, the solution which minimizes a weight is chosen. The user has the option to assign weights to every variable, or to every action. The total weight of a solution is the sum over the weights of the altered variables or the sum over the weight of the actions performed. Actions with heigher weight are therefore less likely to be performed and variables with higher weight are less likely to be altered.

This algorithm is a generalization of the original algorithms in Scholtus (2008) in two ways. First, the original algorithm was designed with a specific type of profit-loss account in mind, while the algorithm of `deducorrect` can handle any set of linear equalities. Second, the original algorithm was not designed to take account of inequality restrictions, which is a feature of the algorithm in this work. In Section 4.4 it is shown how the results of the original example can be reproduced.

4.3 Some simple examples

In this section we walk through most of the options of the `correctSigns` function. We will work with the following six records as example.

```
> (dat <- data.frame(
+      x = c( 3, 14, 15,  1, 17, 12.3),
+      y = c(13, -4,  5,  2,  7, -2.1),
+      z = c(10, 10,-10, NA, 10, 10 )))

     x    y   z
1  3.0 13.0  10
2 14.0 -4.0  10
3 15.0  5.0 -10
4  1.0  2.0  NA
5 17.0  7.0  10
6 12.3 -2.1  10
```

We subject this data to the rule

$$z = x - y. \tag{16}$$

With the `editrules` package, this rule can be parsed to an `editmatrix`.

```
> require(editrules)
> E <- editmatrix(c("z == x-y"))
```

Obviously, not all records in `dat` obey this rule. This can be checked with a function from the `editrules` package:

```
> cbind(dat, violatedEdits(E, dat))

     x    y   z    e1
1  3.0 13.0  10  TRUE
2 14.0 -4.0  10  TRUE
3 15.0  5.0 -10  TRUE
4  1.0  2.0  NA    NA
5 17.0  7.0  10 FALSE
6 12.3 -2.1  10  TRUE
```

Records 1, 2, 3 and 6 violate the editrule, record 5 is valid and for record 4 validity cannot be established since it has no value for z. If `correctSigns` is called without any options, all variables x, y and z can be sign-flipped:

```
> sol <- correctSigns(E, dat)
> cbind(sol$corrected, sol$status)
```

```
     x    y   z    status weight degeneracy nflip nswap
1  3.0 13.0 -10 corrected      1          1     1     0
2 14.0  4.0  10 corrected      1          1     1     0
3 15.0  5.0  10 corrected      1          1     1     0
4  1.0  2.0  NA      <NA>      0          0     0     0
5 17.0  7.0  10     valid      0          0     0     0
6 12.3 -2.1  10   invalid      0          0     0     0
```

```
> sol$corrections
```

```
  row variable old new
1   1        z  10 -10
2   2        y  -4   4
3   3        z -10  10
```

So, the first three records have been correcte by flipping the sign of z, y and z respectively. Since no weight parameter was given, the weight is just the number of variables whose have been sign-flipped. Record 4 is not treated, since validity could not be established, record 5 was valid to begin with and record 6 could not be repaired with sign flips. However, record 6 seems to have a rounding error. We can try to accomodate for that by allowing a tolerance when checking equalities.

```
> sol <- correctSigns(E, dat, eps = 2)
> cbind(sol$corrected, sol$status)
```

```
     x    y   z    status weight degeneracy nflip nswap
1  3.0 13.0 -10 corrected      1          1     1     0
2 14.0  4.0  10 corrected      1          1     1     0
3 15.0  5.0  10 corrected      1          1     1     0
4  1.0  2.0  NA      <NA>      0          0     0     0
5 17.0  7.0  10     valid      0          0     0     0
6 12.3  2.1  10 corrected      1          1     1     0
```

```
> sol$corrections
```

```
  row variable   old   new
1   1        z  10.0 -10.0
2   2        y  -4.0   4.0
3   3        z -10.0  10.0
4   6        y  -2.1   2.1
```

Indeed, changing the sign of y in the last record brings the record within the allowed tolerance. Suppose that we have so much faith in the value of z, that we do not wish to change it's sign. This can be done with the `fixate` option:

```
> sol <- correctSigns(E, dat, eps = 2, fixate = "z")
> cbind(sol$corrected, sol$status)
```

```
      x     y   z    status weight degeneracy nflip nswap
1  -3.0 -13.0  10 corrected      2          1     2     0
2  14.0   4.0  10 corrected      1          1     1     0
3 -15.0  -5.0 -10 corrected      2          1     2     0
4   1.0   2.0  NA      <NA>      0          0     0     0
5  17.0   7.0  10     valid      0          0     0     0
6  12.3   2.1  10 corrected      1          1     1     0
```

```
> sol$corrections
```

```
  row variable  old   new
1   1        x  3.0  -3.0
2   1        y 13.0 -13.0
3   2        y -4.0   4.0
4   3        x 15.0 -15.0
5   3        y  5.0  -5.0
6   6        y -2.1   2.1
```

Indeed, we now find solutions whitout changing z, but at the price of more sign flips. By the way, the same result could have been obtained by

```
> correctSigns(E, dat, flip = c("x", "y"))
```

The sign flips in record 1 and three have the same effect of a variable swap. Allowing for swaps can be done as follows.

```
> sol <- correctSigns(E, dat, swap=list(c("x","y")),
+     eps=2, fixate="z")
> cbind(sol$corrected, sol$status)
```

```
     x    y   z    status weight degeneracy nflip nswap
1 13.0  3.0  10 corrected      1          1     0     1
2 14.0  4.0  10 corrected      1          1     1     0
3  5.0 15.0 -10 corrected      1          1     0     1
4  1.0  2.0  NA      <NA>      0          0     0     0
5 17.0  7.0  10     valid      0          0     0     0
6 12.3  2.1  10 corrected      1          1     1     0
```

```
> sol$corrections
```

```
  row variable  old  new
1   1        x  3.0 13.0
2   1        y 13.0  3.0
3   2        y -4.0  4.0
4   3        x 15.0  5.0
5   3        y  5.0 15.0
6   6        y -2.1  2.1
```

Notice that apart from swapping, the algorithm still tries to correct records by flipping signs. What happened here is that the algorithm first tries to flip the sign of x, then of y, and then it tries to swap x and y. Each is counted as a single action. If no solution is found, it starts trying combinations. In this relatively simple example the result turned out well. In cases with more elaborate systems of equalities and inequalities, the result of the algorithm becomes harder to predict for users. It is therefore in general advisable to

- Use as much knowledge about the data as possible to decide which variables to flip sign and which variable pairs to swap. The problem treated in section 4.4 is a good example of this.
- Keep `flip` and `swap` disjunct. It is better to run the data a few times times through `correctSigns` with different settings.

Not allowing any sign flips can be done with the option `flip=c()`.

```
> sol <- correctSigns(E, dat, flip = c(), swap = list(c("x", "y")))
> cbind(sol$corrected, sol$status)

     x    y   z    status weight degeneracy nflip nswap
1 13.0  3.0  10 corrected      1          1     0     1
2 14.0 -4.0  10   invalid      0          0     0     0
3  5.0 15.0 -10 corrected      1          1     0     1
4  1.0  2.0  NA      <NA>      0          0     0     0
5 17.0  7.0  10     valid      0          0     0     0
6 12.3 -2.1  10   invalid      0          0     0     0

> sol$corrections

  row variable old new
1   1        x   3  13
2   1        y  13   3
3   3        x  15   5
4   3        y   5  15
```

This yields less corrected records. However running the data through

```
> correctSigns(E, sol$corrected, eps = 2)$status

     status weight degeneracy nflip nswap
1     valid      0          0     0     0
2 corrected      1          1     1     0
3     valid      0          0     0     0
4      <NA>      0          0     0     0
5     valid      0          0     0     0
6 corrected      1          1     1     0
```

will fix the remaining edit violations, and yields code which is a lot easyer to interpret.

4.4 Sign errors in a profit-loss account

Here, we will work through the example of chapter 3 of Scholtus (2008). This example considers 4 records, labeled case a, b, c, and d, which can be defined in R as

```
> dat <- data.frame(
+    case = c("a","b","c","d"),
+    x0r = c(2100,5100,3250,5726),
+    x0c = c(1950,4650,3550,5449),
+    x0  = c( 150, 450, 300, 276),
+    x1r = c(   0,   0, 110,  17),
+    x1c = c(  10, 130,  10,  26),
+    x1  = c(  10, 130, 100,  10),
+    x2r = c(  20,  20,  50,   0),
+    x2c = c(   5,   0,  90,  46),
+    x2  = c(  15,  20,  40,  46),
+    x3r = c(  50,  15,  30,   0),
+    x3c = c(  10,  25,  10,   0),
+    x3  = c(  40,  10,  20,   0),
+    x4  = c( 195, 610,-140, 221))
```

A record consists of 4 balance accounts wose results have to add up to a total. Each $x_{i,r}$ denotes some kind of return, x_{ic} some kind of cost and x_i the difference $x_{i,r} - x_{i,c}$. There are operating, financial, provisions and exeptional incomes and expenditures. The differences x_0, x_1, x_2 and x_3 have to add up to a given total x_4. These linear restrictions can be defined with the use of the editrules package.

```
> require(editrules)
> E <-editmatrix(c(
+      "x0 == x0r - x0c",
+      "x1 == x1r - x1c",
+      "x2 == x2r - x2c",
+      "x3 == x3r - x3c",
+      "x4 == x0 + x1 + x2 + x3"))
> E

Edit matrix:
   x0 x0c x0r x1 x1c x1r x2 x2c x2r x3 x3c x3r x4 CONSTANT
e1  1   1  -1  0   0   0  0   0   0  0   0   0  0        0
e2  0   0   0  1   1  -1  0   0   0  0   0   0  0        0
e3  0   0   0  0   0   0  1   1  -1  0   0   0  0        0
e4  0   0   0  0   0   0  0   0   0  1   1  -1  0        0
e5 -1   0   0 -1   0   0 -1   0   0 -1   0   0  1        0

Edit rules:
e1 : x0 == x0r - x0c
e2 : x1 == x1r - x1c
e3 : x2 == x2r - x2c
```

```
e4 : x3 == x3r - x3c
e5 : x4 == x0 + x1 + x2 + x3
```

Checking which records violate what edit rules can be done with the `violatedEdits` function of `editrules`.

```
> violatedEdits(E, dat)

        e1    e2    e3    e4    e5
[1,] FALSE  TRUE FALSE FALSE  TRUE
[2,] FALSE  TRUE FALSE  TRUE FALSE
[3,]  TRUE FALSE  TRUE FALSE  TRUE
[4,]  TRUE  TRUE  TRUE FALSE  TRUE
```

So record 1 (case a) for example, violates the restrictions e_1: $x_1 = x_{1,r} - x_{1,c}$ and e_5, $x_1 + x_2 + sx_3 = x_4$. We can try to solve the inconsistencies by allowing the following flips and swaps:

```
> swap <- list(
+     c("x1r","x1c"),
+     c("x2r","x2c"),
+     c("x3r","x3c"))
> flip <- c("x0","x1","x2","x3","x4")
```

Trying to correct the records by just flipping and swapping variables indicated above corresponds to trying to solve the system of equations

$$
\begin{cases}
x_0 s_0 & = & x_{0,r} - x_{0,c} \\
x_1 s_1 & = & (x_{1,r} - x_{1,c}) t_1 \\
x_2 s_2 & = & (x_{2,r} - x_{2,c}) t_2 \\
x_3 s_3 & = & (x_{3,r} - x_{3,c}) t_3 \\
x_4 s_4 & = & x_0 s_0 + x_1 s_1 + x_2 s_2 + x_3 s_3 \\
(s_0, s_1, s_2, s_3, s_4, t_1, t_2, t_3) \in \{-1, 1\}^8 & , &
\end{cases}
\tag{17}
$$

where every s_i corresponds to a sign flip and t_j corresponds to a value swap, see also Eqn. (3.4) in Scholtus (2008). Using the `correctSigns` function, we get the following.

```
> cor <- correctSigns(E, dat, flip = flip, swap = swap)
> cor$status

     status weight degeneracy nflip nswap
1 corrected      1          1     1     0
2 corrected      2          1     0     2
3 corrected      2          1     1     1
4   invalid      0          0     0     0
```

As expected from the example in the reference, the last record could not be corrected because the solution is masked by a rounding errors. This can be solved by allowing a tolerance of two measurements units.

```
> cor <- correctSigns(E, dat, flip = flip, swap = swap, eps = 2)
> cor$status

     status weight degeneracy nflip nswap
1 corrected      1          1     1     0
2 corrected      2          1     0     2
3 corrected      2          1     1     1
4 corrected      2          1     2     0

> cor$corrected

  case  x0r  x0c   x0 x1r x1c  x1 x2r x2c  x2 x3r x3c x3   x4
1    a 2100 1950  150   0  10 -10  20   5  15  50  10 40  195
2    b 5100 4650  450 130   0 130  20   0  20  25  15 10  610
3    c 3250 3550 -300 110  10 100  90  50  40  30  10 20 -140
4    d 5726 5449  276  17  26 -10   0  46 -46   0   0  0  221
```

The latter table corresponds exactly to Table 2 of Scholtus (2008).

5 Final remarks

This paper demonstrates our implementation of three data correction methods, initially devised by one of us (Scholtus (2008, 2009)). With the `deducorrect` R package, users can correct numerical data records which violate linear equality restrictions for rounding errors, typographical errors and sign errors and/or value transpositions. Since both the algorithms correcting for typographical and sign errors can take rounding errors into account, a typical data-cleaning sequence would be to start with correcting for sign- and typographical errors, ignoring rounding errors and subsequently treating the rounding errors. We note that data cleaning can be sped up significantly if independent blocks of editrules are treated separately. Two sets of editrules A and B are independent when variables occuring in rules of A do not occur in rules of B. The `editrules` package offers functionality to split editmatrices into blocks (with the `findblocks` function).

References

Damerau, F. (1964). A technique for computer detection and correction of spelling errors. *Communications of the ACM 7*, 171–176.

de Jonge, E. and M. van der Loo (2011). *editrules: R package for parsing and manipulating edit rules.* R package version 0.4-1.

Fellegi, P. and D. Holt (1976). A systematic approach to automatic edit and imputation. *Journal of the Americal Statistical Association 71*, 353.

Heller, I. and C. Tompkins (1956). An extension of a theorem of dantzig's. In H. Kuhn and A. Tucker (Eds.), *Linear inequalities and related systems*, pp. 247–254. Princeton University Press.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady 10*, 707–710.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing.* Vienna, Austria: R Foundation for Statistical Computing. ISBN 3-900051-07-0.

Raghavachari, M. (1976). A constructive method to recognize the total unimodularity of a matrix. *Zeitschrift für operations research 20*, 59–61.

Scholtus, S. (2008). Algorithms for correcting obvious inconsistencies and rounding errors in business data. Technical Report 08015, Statistics Netherlands, Den Haag. *Accepted for publication in the Journal of Official Statistics.* The papers are available in the inst/doc directory of the R package or via the website of Statistics Netherlands.

Scholtus, S. (2009). Automatic correction of simple typing error in numerical data with balance edits. Technical Report 09046, Statistics Netherlands, Den Haag. *Accepted for publication in the Journal of Official Statistics.* The papers are available in the inst/doc directory of the R package or via the website of Statistics Netherlands.

A Some notes on the `editrules` package

The `editrules` package (de Jonge and van der Loo, 2011) is a package for reading, parsing and manipulating numerical and categorical editrules. It offers functionality to conveniently construct edit matrices from verbose edit rules, stated as R statements. As an example consider the following set of edits on records with profit p, loss l, and turnover t.

$$\begin{cases} t & \geq & 0 \\ l & \geq & 0 \\ t & = & p+l \\ p & < & 0.6t. \end{cases} \tag{18}$$

The first two rules indicate that turnover and loss must be positive numbers, the third that the profit-loss account must balance, and the last rule indicates that profit cannot be more thatn 60% of the turnover. Denoting a record as a vector (p, l, t), these rules can be denoted as matrix equations:

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p \\ l \\ t \end{bmatrix} \geq \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{19}$$

$$\begin{bmatrix} 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} p \\ l \\ t \end{bmatrix} = 0 \tag{20}$$

$$\begin{bmatrix} 1 & 0 & -0.6 \end{bmatrix} \begin{bmatrix} p \\ l \\ t \end{bmatrix} < 0 \tag{21}$$

In the `editrules` package, these linear rules are all stored in a single object, called an `editmatrix`. It can be constructed as follows:

```
> (E <- editmatrix(c(
+     "t >= 1",
+     "l >= 0",
+     "t == p + l",
+     "p < 0.6*t")))

Edit matrix:
      t  l  p CONSTANT
e1  1.0  0  0        1
e2  0.0  1  0        0
e3  1.0 -1 -1        0
e4 -0.6  0  1        0

Edit rules:
e1 : t >= 1
e2 : l >= 0
e3 : t == p + l
e4 : p < 0.6*t
```

An `editmatrix` object stores a stacked matrix representation of linear edit resrictions. There are more storage modes in `editrules` which we will not detail here. Users can extract (in)equalities through the `getOps` function which returns a vector of comparison operators for every row. For example:

```
> E[getOps(E)==">=", ]

Edit matrix:
   t l p CONSTANT
e1 1 0 0        1
e2 0 1 0        0

Edit rules:
e1 : t >= 1
e2 : l >= 0
```

Alternatively, an the comparison operators of an editmatrix may be normalized:

```
> editmatrix(editrules(E), normalize = TRUE)

Edit matrix:
      t  l  p CONSTANT
e1 -1.0  0  0        1
e2  0.0 -1  0        0
e3  1.0 -1 -1        0
e4 -0.6  0  1        0

Edit rules:
e1 : -1 <= t
e2 : 0 <= l
e3 : t == l + p
e4 : p < 0.6*t
```

The `editrules` package offers functionality to check data against any set of editrules. The function `violatedEdits`, for example returns a boolean matrix indicating which record violates what editrules. `editrules` also offers editrule manipulation functionality, for example to split editmatrices into independent blocks. For further functionality of the `editrules` package, refer to the package documentation.