Linear edit manipulation and error localization with the `editrules` package

Edwin de Jonge and Mark van der Loo

May 11, 2011

Abstract

This vignette is far from finished. Version 1.0 fo the package will have the full vignette.

Contents

1 Introduction

The value domain of real numerical data records with n variables is often restricted to a subdomain of $\mathbb{R}^n$ due to linear equality and inequality relations which the variables in the record have to obey. Examples include equality restrictions imposed by financial balance accounts, positivity demands on certain variables or limits on the ratio of variables.

Any such restriction is of the form

$$\mathbf{a}\cdot\mathbf{x}\odot b \text{ with } \odot\in\{<,\leq,=\}, \tag{1}$$

where $\mathbf{x}$ is a numerical data record, $\mathbf{a}$, $\mathbf{x}\in\mathbb{R}^n$ and $b\in\mathbb{R}$. In data editing literature, data restriction rules are refered to as *edits*, or *edit rules*. We will call edits, written in the form of Eq. (1), edits in *normal form*.

Large complex surveys are often endowed with dozens or even hundreds of edit rules. For example, the Dutch Structural Business Survey, which aims to report on the financial structure of companies in the Netherlands, contains on the order of 100 variables, endowed with a similar number of linear equality and inequality restrictions.

Defining and manipulating large edit sets can be a daunting task , while edit violations gives rise to the error localization problem, which can quite simply be stated as *which variables contain the errors that cause a record to violate certain edits rules?*.

The `editrules` package for the R statistical computing environment (R Development Core Team, 2011) aims to provide an environment to conveniently define, parse and check linear (in)equality restrictions, perform common edit manipulations and offer error localization functionality based on the (generalized) paradigm of Fellegi and Holt (1976). This paradigm is based on the assumption that errors are distributed randomly over the variables, and there is no detectable cause of error. The paradigm also decouples the detection from correction of corrupt variables. Certain causes of error, such as sign flips, typing errors or rounding errors can be detected and are closely related to their resolution. The reader is referred to the `deducorrect` package (van der Loo et al., 2011; Scholtus, 2008, 2009) for treating such errors.

The following chapters demonstrates the functionality of the `editrules` package with coded examples as well a description of of the underlying theory and algorithms. For a detailed per-function description the reader is referred to the reference manual accompanying the package. Unless mentioned otherwise, all code shown in this paper can be executed from the R commandline after loading the `editrules` package.

2 Defining and checking numerical restrictions

2.1 The `editmatrix` object

For computational processing, a set of edits of the form

$$\mathbf{a}\cdot\mathbf{x}\odot b \text{ with } \odot\in\{<,\leq,=,\geq,>\}. \tag{2}$$

is most conveniently represented as a matrix. In the `editrules` package, a set of linear edits is stored as an `editmatrix` object. This object stores the linear relations as an augmented matrix $[\mathbf{A}, \mathbf{b}]$, where $\mathbf{A}$ is the matrix obtained by combining the $\mathbf{a}$ vecors of Eq. (2) in rows of $\mathbf{A}$ and constants b in $\mathbf{b}$. A second attribute holds the comparison operators as a `character` vector. Formally, we denote that every `editmatrix` E is defined by

$$E = \langle [\mathbf{A}, \mathbf{b}], \odot \rangle \text{ with } [\mathbf{A}, \mathbf{b}] \in \mathbb{R}^{m \times n+1},\ \odot \in \{<, \leq, =, \geq, >\}^m, \tag{3}$$

where n is the number of variables, m the number of edit rules and the notation $\langle\ ,\ \rangle$ denotes a combination of objects. Retrieval functions for various parts of an `editmatrix` are available, see Table 1 (p. 6) for an overview. Defining augmented matrices by hand is tedious and prone to error, which is why the `editmatrix` function derives edit matrices from a textual representation of edit rules. Since most functions of the `editrules` package expect an `editmatrix` in normal form (that is $\odot \in \{<, \leq, =\}$), the `editmatrix` function by default transforms all linear edits to normal form.

As an example, consider the set of variables

turnover	t
personell cost	c_p
housing cost	c_h
total cost	c_t
profit	p,

subject to the rules

$$\begin{aligned}
t &= c_t + p && (4)\\
c_t &= c_h + c_p && (5)\\
p &\leq 0.6t && (6)\\
c_t &\leq 0.3t && (7)\\
c_p &\leq 0.3t && (8)\\
t &> 0 && (9)\\
c_h &> 0 && (10)\\
c_p &> 0 && (11)\\
c_t &> 0. && (12)
\end{aligned}$$

Clearly, these can be written in the form of Eq. (1). Here, the equality restrictions correspond to balance accounts, the 3rd, 4th and 5th restrictions are sanity checks and the last four edits demand positivity. Figure 1 shows how these edit rules can be transformed from a textual representation to a matrix representation with the `editmatrix` function.

As Figure 1 shows, the `editmatrix` object is shown on the console as a matrix, as well as a set of textual edit rules. The `editrules` package is capable of coercing a set of R expressions to an `editmatrix` and *vice versa*. To coerce text to a matrix, the `editmatrix` function processes the R language parsetree of the textual R expressions as provided by the R internal `parse` function. To coerce the matrix representation to textual representation, an R character string is derived from the matrix which can be parsed to a language object.

In the example, the edits were automatically named `e1`, `e2`, ..., `e9`. It is possible to name and comment edits by reading them from a `data.frame`.

```
> E <- editmatrix(c(
+ "t  == ct + p" ,
+ "ct == ch + cp",
+ "p  <= 0.6*t",
+ "cp <= 0.3*t",
+ "ch <= 0.3*t",
+ "t  >  0",
+ "ch >  0",
+ "cp >  0",
+ "ct >  0"), normalize=TRUE)
> E

Edit matrix:
   ct  p    t ch cp Ops  CONSTANT
e1 -1 -1  1.0  0  0  ==         0
e2  1  0  0.0 -1 -1  ==         0
e3  0  1 -0.6  0  0  <=         0
e4  0  0 -0.3  0  1  <=         0
e5  0  0 -0.3  1  0  <=         0
e6  0  0 -1.0  0  0   <         0
e7  0  0  0.0 -1  0   <         0
e8  0  0  0.0  0 -1   <         0
e9 -1  0  0.0  0  0   <         0

Edit rules:
e1 : t == ct + p
e2 : ct == ch + cp
e3 : p <= 0.6*t
e4 : cp <= 0.3*t
e5 : ch <= 0.3*t
e6 : 0 < t
e7 : 0 < ch
e8 : 0 < cp
e9 : 0 < ct
```

Figure 1: Defining an `editmatrix` from a `character` vector containing verbose edit statements. The option `normalize=TRUE` ensures that all comparison operators are either $<$, $\leq$ or $==$.

The ability to read edit sets from a `data.frame` facilitates defining and maintaining the rules outside of the R environment by storing them in a user-filled database or textfile. Manipulating and combining edits, for example through variable elimination methods will cause `editrules` to drop or change the names and drop the comments, as they become meaningless after certain manipulations.

2.2 Basic manipulations and edit checking

Table 1 shows simple manipulation functions available for an `editmatrix`. Basic manipulations include retrieval functions for the augmented matrix, coefficient matrix, constant vector and operators of an `editmatrix`. There are functions to test for and transform to normality. The function `violatedEdits` expects

```
> # generate a csv text string
> E.csv <-
+ 'name , edit       , description
+ "b1"  ,    t == ct + p    ,  "total balance"
+ "b2"  ,    ct == ch + cp  ,  "cost balance"
+ "s1"  ,    p <= 0.6*t     ,  "profit sanity"
+ "s2"  ,    cp <= 0.3*t    ,  "personell cost sanity"
+ "s3"  ,    ch <= 0.3*t    ,  "housing cost sanity"
+ "p1"  ,    t >0           ,  "turnover positivity"
+ "p2"  ,    ch > 0         ,  "housing cost positivity"
+ "p3"  ,    cp > 0         ,  "personel cost positivity"
+ "p4"  ,    ct > 0         ,  "total cost positivity"'
> # read into a data.frame
> E.df <- read.csv(textConnection(E.csv))
> # transform to an editmatrix
> editmatrix(E.df)

Edit matrix:
     ct  p    t ch cp Ops CONSTANT
b1   -1 -1  1.0  0  0  ==        0
b2    1  0  0.0 -1 -1  ==        0
s1    0  1 -0.6  0  0  <=        0
s2    0  0 -0.3  0  1  <=        0
s3    0  0 -0.3  1  0  <=        0
p1    0  0 -1.0  0  0   <        0
p2    0  0  0.0 -1  0   <        0
p3    0  0  0.0  0 -1   <        0
p4   -1  0  0.0  0  0   <        0

Edit rules:
b1   : t == ct + p [   total balance ]
b2   : ct == ch + cp [   cost balance ]
s1   : p <= 0.6*t [   profit sanity ]
s2   : cp <= 0.3*t [   personell cost sanity ]
s3   : ch <= 0.3*t [   housing cost sanity ]
p1   : 0 < t [   turnover positivity ]
p2   : 0 < ch [   housing cost positivity ]
p3   : 0 < cp [   personel cost positivity ]
p4   : 0 < ct [   total cost positivity ]
```

Figure 2: Declaring an editmatrix with a `data.frame`. The input `data.frame` is required to have three columns named `name`,(edit name, stored as rowname of augmented matrix) `edit` (textual representation of the edit rule) and `description` (a comment stating the intent of the rule). All must be of type `character`.

an `editmatrix` and a `data.frame` or a named numeric vector. It returns a `logical array` where every row indicates which edits are violated (`TRUE`) by records in the `data.frame`. Figure 3 demonstrates the result of checking two records against the editrules defined in Eqs. (4)–(12). Indexing of edits with the `[` operator is restricted to selection only. No assignment can be made to indexed `editmatrix` objects. In stead, `as.editmatrix` should be used.

Table 1: Simple manipulation functions for objects of class `editmatrix`

function	description
`getA(E)`	Get matrix $\mathbf{A}$
`getb(E)`	Get constant vector $\mathbf{b}$
`getAb(E)`	Get augmented matrix $[\mathbf{A},\mathbf{b}]$
`getOps(E)`	Get comparison operators
`E[i,]`	Select edit(s)
`as.editmatrix(A,b,ops)`	Create an `editmatrix` from it's attributes
`normalize(E)`	Transform `E` to normal form
`isNormalized(E)`	Check whether `E` is in normal form
`violatedEdits(E, x)`	Check which edits are violated by `x`
`isObviouslyRedundant(E)`	Check for tautologies in rows of `E`
`isObviouslyUnfeasible(E)`	Check for contradictions in rows of `E`

```
> # define two records in a data.frame
> dat <- data.frame(
+   t = c(1000, 1200),
+  ct = c(400,  200),
+  ch = c(100,  350),
+  cp = c(500,  575),
+  p =  c(500,  652 ))
> # check for violated edits
> violatedEdits(E,dat)

       e1   e2    e3   e4    e5    e6    e7    e8    e9
[1,] TRUE TRUE FALSE TRUE FALSE FALSE FALSE FALSE FALSE
[2,] TRUE TRUE FALSE TRUE FALSE FALSE FALSE FALSE FALSE
```

Figure 3: Checking which edits are violated for every record in a `data.frame`. The first record violates `e1` and `e2`, the second record violates `e1`,`e2`, and `e4`.

2.3 Obvious redundancy and infeasibility

When manipulating linear edit sets by value substitution and/or variable elimination, the edit set can become polluted with redundant edits or, when variable values are substituted, become infeasible. The `editrules` package has two methods available which check for easily detectable redundancies or infeasibility. The fourier-motzkin elimination method has auxilary built-in redundancy removal, which is described in Section 3.3.

A system of inequalities $\mathbf{Ax} \leq \mathbf{b}$ is called infeasible when there is no real vecor $\mathbf{x}$ satisfying it. It is a consequence of Farkas' lemma (Farkas (1902), but see Schrijver (1998) and/or Kuhn (1956)) on feasibility of sytems of linear equalities, that a system is infeasible if and only if $0 \leq -1$ can be derived by taking positive linear combinations of the rows of the augmented matrix $[\mathbf{A},\mathbf{b}]$. The function `isObiouslyinfeasible` returns a `logical` indicating whether such a contradiction is present. Subsitution of values may also lead to equalities of the form $0 = 1$, which also indicate that the system has become infeasible. Being obviously infeasible is sufficient for an `editmatrix` to be infeasible, but not necessary. Algorithm 1 gives the pseudocode for reference purposes.

When new edits are derived, either by value substitution or by variable

Algorithm 1 isObviouslyInfeasible(E)

```
Input: a normalized editmatrix $E$
  for $\mathbf{a}\cdot\mathbf{x}\odot b \in E$ do
    if $\mathbf{a} = \mathbf{0}$ then
      if $(\odot \in \{=\} \wedge b \neq 0) \vee (\odot \in \{\leq, <\} \wedge b < 0)$ then
        return TRUE
      end if
    end if
  end for
  return FALSE
Output:                ▷ logical indicating if $E$ is obviously infeasible.
```

Algorithm 2 isObviouslyRedundant(E)

```
Input: a normalized editmatrix $E$, with $m$ edits
  $\mathbf{v} \leftarrow (\text{FALSE})^{\times m}$
  $i \leftarrow 0$
  for $\mathbf{a}\cdot\mathbf{x}\odot b \in E$ do
    $i \leftarrow i + 1$
    if $\mathbf{a} = \mathbf{0}$ then
      if $(\odot \in \{=\} \wedge b = 0) \vee (\odot \in \{\leq, <\} \wedge b > 0)$ then
        $v_i \leftarrow$TRUE
      end if
    end if
  end for
Output: $\mathbf{v}$      ▷ logical vector indicating which rows of $E$ are obviously
  redundant.
```

elimination, redundant rules of the form $0 \leq 1$ or $0 = 0$ can be generated. The function `isObviouslyRedundant` detects such rules and returns a `logical` vector indicating which rows of an `editmatrix` are redundant. Pseudocode is given in Algorithm 2.

3 Manipulation of linear restrictions

There are two fundamental operations possible on edit sets, both of which (possibly) reduce the number of variables involved in the edit set. The first, most simple one is when a value is substituted into an edit. The second possibility is variable elimination. For a set of linear equalities, one can apply Gaussian elimination, while for sets of inequalities or mixed sets of equalities and inequalities Fourier-Motzkin elimination is applied. While variable substitution and Gaussian elimination guarantee that the eliminated variable is not involved in the derived edit set anymore, this is not necessarily the case for Fourier-Motzkin elimination.

```
Algorithm 3 REPLACEVALUE($E, j, x$)
Input: $E = \langle[\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_j, \ldots, \mathbf{a}_n], \mathbf{b}], \odot\rangle$, $x \in \mathbb{R}$, $j \in \{1, 2, \ldots n\}$
Output: $\langle[\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{j-1}, \mathbf{0}, \mathbf{a}_{j+1}, \ldots \mathbf{a}_n], \mathbf{b} - \mathbf{a}_j x], \odot\rangle$
```

```
> replaceValue(E, "t", 10)

Edit matrix:
   ct  p t ch cp Ops CONSTANT
e1 -1 -1 0  0  0  ==      -10
e2  1  0 0 -1 -1  ==        0
e3  0  1 0  0  0  <=        6
e4  0  0 0  0  1  <=        3
e5  0  0 0  1  0  <=        3
e7  0  0 0 -1  0   <        0
e8  0  0 0  0 -1   <        0
e9 -1  0 0  0  0   <        0

Edit rules:
e1 : 0 == ct + p + -10
e2 : ct == ch + cp
e3 : p <= 6
e4 : cp <= 3
e5 : ch <= 3
e7 : 0 < ch
e8 : 0 < cp
e9 : 0 < ct
```

Figure 4: Substituting the value 10 for the turnover variable using the `replaceValue` function.

3.1 Value substitution

Given a set of m linear edits as defined in Eq. (3). For any record $\mathbf{x}$ it must hold that

$$\mathbf{A}\mathbf{x} \odot \mathbf{b}, \quad \odot \in \{<, \leq, =, \geq, >\}^m. \tag{13}$$

Substituting one of the unknowns x_j by a certain value x amounts to replacing the j column of $\mathbf{A}$ with $\mathbf{0}$ and $\mathbf{b}$ with $\mathbf{b} - \mathbf{a}'_j x$. After this, the reduced record of unknowns, with x_j replaced by x has to obey the adapted system (13). For reference purposes, Algorithm 3 spells out the substitution routine. The function was named `replaceValue` since `substitute` is already defined in the R-base. Figure 4 shows how `replaceValue` can be called from the R environment.

3.2 Gaussian elimination

The well-known Gaussian elimination routine has been implemented here as a utility function, enabeling users to reduce the equality part of their edit matrices to reduced row echelon form. The `echelon` function has been overloaded to take either an R `matrix` or an `editmatrix` as argument. In the latter case, the equalities are transformed to reduced row echelon form, while inequalities are left untreated.

```
> echelon(E)

Edit matrix:
   ct p    t ch cp Ops CONSTANT
e1  1 0  0.0 -1 -1  ==        0
e2  0 1 -1.0  1  1  ==        0
e3  0 1 -0.6  0  0  <=        0
e4  0 0 -0.3  0  1  <=        0
e5  0 0 -0.3  1  0  <=        0
e6  0 0 -1.0  0  0   <        0
e7  0 0  0.0 -1  0   <        0
e8  0 0  0.0  0 -1   <        0
e9 -1 0  0.0  0  0   <        0

Edit rules:
e1 : ct == ch + cp
e2 : p + ch + cp == t
e3 : p <= 0.6*t
e4 : cp <= 0.3*t
e5 : ch <= 0.3*t
e6 : 0 < t
e7 : 0 < ch
e8 : 0 < cp
e9 : 0 < ct
```

Figure 5: The `echelon` function transforms the linear equalities of an editmatrix to reduced row echelon form. See Figure 1 for the original definition of `E`.

3.3 Fourier-Motzkin elimination

4 Error localization for numerical data

4.1 The generalized Fellegi-Holt paradigm

4.2 General binary search with the `choicepoint` algorithm

4.3 Error localization with `cp.editmatrix`

5 Conclusions

References

Farkas, G. (1902). Über die theorie der einfachen ungleichungen. *Journal für die Reine und Angewandte Mathematik 124*, 1–27.

Fellegi, I. P. and D. Holt (1976). A systematic approach to automatic edit and imputation. *Journal of the Americal Statistical Association 71*, 17–35.

Kuhn, H. W. (1956). Solvability and consistency for linear equations and inequalities. *The American Mathematical Monthly 63*, 217–232.

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.

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

Schrijver, A. (1998). *Theory of linear and integer programming.* Wiley-Interscience series in descrete mathematics and optimization. New York: John Wiley and Sons.

van der Loo, M., E. de Jonge, , and S. Scholtus (2011). *deducorrect: Deductive correction of simple rounding, typing and sign errors.* R package version 0.9-2.