Introduction to the `MethComp` package

(compiled Friday 4th February, 2011, 22:37)

Bendix Carstensen Steno Diabetes Center, Gentofte, Denmark
& Department of Biostatistics, University of Copenhagen
`bxc@steno.dk`
`www.biostat.ku.dk/~bxc`

Contents

The purpose of the MethComp package is to provide computational tools to manipulate, display and analyze data from method comparison studies. A method comparison study is a study where two methods of quantitative measurement are compared by measuring the same set of items with both methods.

There may be more than two methods, and there may be replicate measurements on each item by each method.

1 Data structures

In general we are concerned with measurements by different methods, on different items (persons, samples), possibly replicated.

Often such data are represented by a row of measurements for each item, with possible replicates listed either below or beside each other. This implicitly assumes that some replicate measurements belong together, which is not necessarily the case in all situations.

All functions in MethComp assume data to be represented in the "long" form, with one measurement on each row, and columns to indicate method, item and replicate. Specifically, we assume the following columns are available in a data frame:

- `meth` The measurement method. Numeric or factor.
- `item` Identification of item (person, sample). Numeric or factor.
- `repl` Replicate number. Numeric or factor.
- `y` The measurement by method `meth` on item `item`, replicate number `repl`.

There is a class, "`Meth`" for this kind of data frame. A dataframe is converted to a `Meth` object by using the `Meth` function on it:

```
> data( ox )
> str( ox )

'data.frame':        354 obs. of  4 variables:
 $ meth: Factor w/ 2 levels "CO","pulse": 1 1 1 1 1 1 1 1 1 1 ...
 $ item: num  1 1 1 2 2 2 3 3 3 4 ...
 $ repl: num  1 2 3 1 2 3 1 2 3 1 ...
 $ y   : num  78 76.4 77.2 68.7 67.6 68.3 82.9 80.1 80.7 62.3 ...

> ox <- Meth( ox )

The following variables from the dataframe
"ox" are used as the Meth variables:
meth: meth
item: item
repl: repl
   y: y
```

```
        #Replicates
Method    1   2   3 #Items #Obs: 354 Values:  min  med  max
  CO      1   4  56     61       177          22.2 78.6 93.5
  pulse   1   4  56     61       177          24.0 75.0 94.0
```

```
> summary( ox )
```

```
        #Replicates
Method    1   2   3 #Items #Obs: 354 Values:  min  med  max
  CO      1   4  56     61       177          22.2 78.6 93.5
  pulse   1   4  56     61       177          24.0 75.0 94.0
```

If these variable are not availabe in the data frame we may create them on the fly or by giving the variable positions as arguments to the `Meth` function:

```
> data( fat )
> str( fat )
```

```
'data.frame':        258 obs. of  5 variables:
 $ Id : num  1 1 1 3 3 3 5 5 5 11 ...
 $ Obs: Factor w/ 2 levels "KL","SL": 1 1 1 1 1 1 1 1 1 1 ...
 $ Rep: num  1 2 3 1 2 3 1 2 3 1 ...
 $ Sub: num  1.6 1.7 1.7 2.8 2.9 2.8 2.7 2.8 2.9 3.9 ...
 $ Vic: num  4.5 4.4 4.7 6.4 6.2 6.5 3.6 3.9 4 4.3 ...
```

```
> sc <- Meth( fat, 2, 1, 3, 4 )
```

```
The following variables from the dataframe
"fat" are used as the Meth variables:
meth: Obs
item: Id
repl: Rep
   y: Sub
       #Replicates
Method         3 #Items #Obs: 258 Values:  min med max
    KL        43     43       129          0.39 1.7 4.2
    SL        43     43       129          0.51 1.7 4.1
```

```
> str( sc )
```

```
Classes âĂŸMethâĂŹ and 'data.frame':        258 obs. of  5 variables:
 $ meth: Factor w/ 2 levels "KL","SL": 1 1 1 1 1 1 1 1 1 1 ...
 $ item: Factor w/ 43 levels "1","2","3","4",..: 1 1 1 3 3 3 5 5 5 11 ...
 $ repl: Factor w/ 3 levels "1","2","3": 1 2 3 1 2 3 1 2 3 1 ...
 $ y   : num  1.6 1.7 1.7 2.8 2.9 2.8 2.7 2.8 2.9 3.9 ...
 $ Vic : num  4.5 4.4 4.7 6.4 6.2 6.5 3.6 3.9 4 4.3 ...
```

```
> summary( sc )

       #Replicates
Method          3 #Items #Obs: 258 Values:  min med max
    KL         43     43       129          0.39 1.7 4.2
    SL         43     43       129          0.51 1.7 4.1
```

We may even give some of them as names of the columns in the dataframe:

```
> vi <- Meth( fat, 2,1,"Rep","Vic" )

The following variables from the dataframe
"fat" are used as the Meth variables:
meth: Obs
item: Id
repl: Rep
   y: Vic
       #Replicates
Method          3 #Items #Obs: 258 Values:  min med max
    KL         43     43       129           2.0 3.9 6.5
    SL         43     43       129           2.3 4.1 6.7
```

However, more complicated operations on the dataframe is best done on the fly using the `with` function (from the `base` package):

```
> data( hba1c )
> str( hba1c )

'data.frame':        835 obs. of  6 variables:
 $ dev   : Factor w/ 3 levels "BR.V2","BR.VC",..: 2 2 2 2 2 2 2 2 1 1 ...
 $ type  : Factor w/ 2 levels "Cap","Ven": 2 2 2 2 1 1 1 1 2 2 ...
 $ item  : num  12 12 12 12 12 12 12 12 12 12 ...
 $ d.samp: num  1 1 1 1 1 1 1 1 1 1 ...
 $ d.ana : num  2 3 4 5 2 3 4 5 2 3 ...
 $ y     : num  8.7 8.7 8.7 8.7 9.2 9 8.8 8.7 9.4 9.3 ...

> hb1  <- with( hba1c,
+               Meth( meth = interaction(dev,type),
+                     item = item,
+                     repl = d.ana-d.samp,
+                        y = y, print=TRUE ) )

            #Replicates
Method          3     4 #Items #Obs: 835 Values:  min med  max
  BR.V2.Cap     0    38     38       152          5.3 8.0 12.6
  BR.VC.Cap    19    19     38       133          5.3 8.2 12.1
  Tosoh.Cap     0    38     38       152          5.0 7.8 11.8
  BR.V2.Ven    19    19     38       133          5.5 8.1 12.0
  BR.VC.Ven    19    19     38       133          5.3 8.0 11.6
  Tosoh.Ven    20    18     38       132          5.3 8.0 12.1
```

```
> str( hb1 )

Classes âĂŸMethâĂŹ and 'data.frame':	835 obs. of  4 variables:
 $ meth: Factor w/ 6 levels "BR.V2.Cap","BR.VC.Cap",..: 5 5 5 5 2 2 2 2 4 4 ...
 $ item: Factor w/ 38 levels "1","2","3","4",..: 12 12 12 12 12 12 12 12 12 12 ...
 $ repl: Factor w/ 5 levels "0","1","2","3",..: 2 3 4 5 2 3 4 5 2 3 ...
 $ y   : num  8.7 8.7 8.7 8.7 9.2 9 8.8 8.7 9.4 9.3 ...
```

Objects of class `Meth` (which inherits from `data.frame`) has methods such as `summary`, `plot`, `subset` and `transform`. The functions mostly do not require the data to be in `Meth` format — if a dataframe with the right columns is supplied, it is normally converted internally to `Meth` format.

2 Function overview

The following is a brief overview of the functions in the `MethComp` package. The full documentation is in the help pages for the functions, and an illustration of the way they work can be obtained by referring to the printed manual at the end of this document or on the fly by typing e.g.:

```
> ?plot.Meth
```

which will bring up the manual page for the function `plot.meth`. The example code from the manual page can be run directly by:

```
> example( plot.Meth )
```

2.1 Graphical functions

`BA.plot` Makes a Bland-Altman plot of two methods from a data frame with method comparison data, and computes limits of agreement. The plotting is really done by a call to the function `BlandAltman`.

`BlandAltman` draws a Bland-Altman plot and computes limits of agreement.

`plot.Meth` Plots all methods against all others, both as a scatter plot and as a Bland-Altman plot.

`bothlines` Adds regression lines of y on x and vice versa to a scatter plot. Optionally, the Deming regression line can be added too.

2.2 Data manipulating functions

`make.repl` Generates (or replaces) a `repl` column in a data frame with columns `meth`, `item` and `y`.

`perm.repl` Randomly permutes replicates within (method,item) and assigns new replicate numbers.

`to.wide` Transforms a data frame in the long form to the wide form where separate columns for each method are generated, with one row per (item,replicate).

`to.long` Reverses the result of `to.wide`. The function can also generate a long form dataset from a dataset with different methods beside each other.

`summary.Meth` Tabulates items by method and no. replicates for a `Meth` object.

`Meth.sim` Simulates a dataset from a method comparison experiment for given parameters for bias, exchangeability and variance component sizes.

2.3 Analysis functions

`BA.est` Estimates in the variance components models underlying the concept of limits of agreement, and returns the bias and the variance components. Assumes constant bias between methods.

`Deming` Performs Deming regression, i.e. regression with errors in both variables.

`DA.reg` Regresses the differeneces between methods on the averages and derives approximate linear conversion equations, based on [**?**].

`AltReg` Estimates via alternating regressions in the general model. Returns estimates of mean conversion parameters and variance components.

`MCmcmc` Estimates via `BUGS` in the general model with non-constant bias (and in the future) possibly non-constant standard deviations of the variance components. Produces a `MCmcmc` object, which is an `mcmc.list` object with some extra attributes. `mcmc.list` objects are handeled by the `coda` package, so this is required when calling `MCmcmc`.

2.4 Reporting functions

Some of these functions all take a `MCmcmc` object as input, others will postprocess the output of `DA.reg`, `BA.est` or `AltReg`.

The functions `BA.est`, `AltReg` return objetcs that have class `MethComp`, whereas the result of `MCmcmc` can be converted to an object of this type by the `MethComp` function. The reason for this is that the results of the `MCmcmc` function is output from an MCMC-simulation which we may want to monitor by specisl functions. The `MethComp` function only takes the central summaries from the `MCmcmc` object assuming the chains have reached convergence.

`print.MethComp` Prints a table of conversion equation between methods analyzed, with prediction standard deviations.

`print.MCmcmc` Prints a table of conversion equation between methods analyzed, with prediction standard deviations, but also gives summaries of the posteriors for the parameters that constitute the conversion algorithms.

`plot.MethComp`, `plot.MCmcmc` Plots the conversion lines between methods with prediction limits.

`post.MCmcmc` Plots smoothed posterior densities for the estimates. Primarily of interest for the variance components, but it has aruments to produce the posterior of the intercepts and the slopes of the conversion lines between methods too.

`check.MCmcmc` Makes diagnistic plots of the traces of the chains included in the `MCmcmc` object.