Package 'ODBbB'

Version: 0.1.0

Date: May 2020

Author: Glen LLaci

Maintainer: Glen Llaci glen_llaci@yahoo.com (mailto:glen_llaci@yahoo.com)

Description

The `ODBbB` package has three core functions. Transposing the data table Block by block, Inversing and taking two different data tables or matrices and executing the product between their blocks. There are also some other functions that return to the user the Mean or the SD of a matrix block by block (even if we have a large data table).

License GPL-2

Introductory example

Let's see a simple example in the case of transposing block by block. the other fonctions follow the same steps, but with different operation.
Example 0:

```
print(dt)
```

```
##     x  y  z
## 1  a1  1 11
## 2  b1  2 12
## 3  c1  3 13
## 4  d1  4 14
## 5  a2  5 15
## 6  b2  6 16
## 7  c2  7 17
## 8  d2  8 18
## 9  a3  9 19
## 10 b3 10 20
## 11 c3 11 21
## 12 d3 12 22
```

can be transposed as:

```
##   1    2    3    4
## x "a1" "b1" "c1" "d1"
## y "1"  "2"  "3"  "4"
## z "11" "12" "13" "14"
## x "a2" "b2" "c2" "d2"
## y "5"  "6"  "7"  "8"
## z "15" "16" "17" "18"
## x "a3" "b3" "c3" "d3"
## y " 9" "10" "11" "12"
## z "19" "20" "21" "22"
```

We can use this pacage as well to mesure the mean or the standart deviation of each colum or row of all the blocks that we transpose. It can also give the total mean or standart deviation of each block one by one.

Contents of the package:

The functions that compose this package are

- **TrDtBbB**: It makes the transpose block by block of the data table or matrix.
- **InvDtBbyB**: This function takes any NxN data table and inverse its data block by block.
- **ProdDtBbB**: This function is used in the case when we need to execute the product block by block of two different data tables or matrices.
- **BlockTrMean**: It give the `Mean` of each block.
- **BlockTrSD**: It give the `Standart Deviation` of each block.
- **LargeTrMean**: It will transpose large data and give the `Mean` by block.
- **LargeTrSD**: It will transpose large data and give the `Standart Deviation` by block.

TrDtBbB

Transpose data table block by block

Description

This function transpose any data table or matrix block by block.

Usage

TrDtBbB(Data=data(),lengthBlock=numeric(),cls=c(),rs=c(),time_col=NULL)

Arguments

Data	The data table or matrix that we're going to transpose block by block.
lengthBlock	The length of the blocks that will be transposed.
cls	The columns that we want to include in the transposition
rs	The rows of the first block. Based on this rows the function will select identically the rows of every other block.
time_col	The *time parameter* is optional. We can ignore it if our data has no time values.

Example

For the example we're going to use a part of the monthly audience data of five France's television chanels and the time that they dedicate to 14 different themes The source data has 5 channels (in columns) and 14 themes (in rows) that are repeated in every month from january 2005 to jun 2018. In our example we will use only the 3 first months of 2005 and the 7 first themes.

Example 1:

The original table before the transposation by block is:

```
print(table1)
```

```
##        MOIS       THEMATIQUES TF1 France_2 France_3 Canal_plus Arte
## 1  janv.-05      Catastrophes 214      191       88         18   40
## 2  janv.-05   Culture_loisirs  27       42       35          4    0
## 3  janv.-05          Economie  35       18       10          1    8
## 4  janv.-05         Education  14       12        4          3    3
## 5  janv.-05     Environnement  31       25       15          1    3
## 6  janv.-05      Faits divers  24       19        9          1    1
## 7  janv.-05 Histoire_hommages  36       38       24         10   23
## 8  févr.-05      Catastrophes  42       30       13          4   14
## 9  févr.-05   Culture_loisirs  41       66       23         12    5
## 10 févr.-05          Economie  46       43       24          2    7
## 11 févr.-05         Education  27       31       22          7    7
## 12 févr.-05     Environnement  46       25       12          0    6
## 13 févr.-05      Faits divers  17       16       13          8    0
## 14 févr.-05 Histoire_hommages  14       12       12          1    9
## 15  mars-05      Catastrophes  47       44       27          1   10
## 16  mars-05   Culture_loisirs  48       81       38         14    2
## 17  mars-05          Economie  92       50       27          2   13
## 18  mars-05         Education  33       24       15          3    4
## 19  mars-05     Environnement  57       57       14          4    0
## 20  mars-05      Faits divers  32       28       18          3    9
## 21  mars-05 Histoire_hommages   7        7        1          1    3
```

When we apply the function on the data we will have

```
table2<-TrDtBbB(Data=table1,lengthBlock=7,cls=c(3:7),rs=c(1:7),time_col=1)
print(table2)
```

```
##            time       1     2    3    4    5    6    7
## TF1        "janv.-05" "214" "27" "35" "14" "31" "24" "36"
## France_2   "janv.-05" "191" "42" "18" "12" "25" "19" "38"
## France_3   "janv.-05" "88"  "35" "10" "4"  "15" "9"  "24"
## Canal_plus "janv.-05" "18"  "4"  "1"  "3"  "1"  "1"  "10"
## Arte       "janv.-05" "40"  "0"  "8"  "3"  "3"  "1"  "23"
## TF1        "févr.-05" "42"  "41" "46" "27" "46" "17" "14"
## France_2   "févr.-05" "30"  "66" "43" "31" "25" "16" "12"
## France_3   "févr.-05" "13"  "23" "24" "22" "12" "13" "12"
## Canal_plus "févr.-05" "4"   "12" "2"  "7"  "0"  "8"  "1"
## Arte       "févr.-05" "14"  "5"  "7"  "7"  "6"  "0"  "9"
## TF1        "mars-05"  "47"  "48" "92" "33" "57" "32" "7"
## France_2   "mars-05"  "44"  "81" "50" "24" "57" "28" "7"
## France_3   "mars-05"  "27"  "38" "27" "15" "14" "18" "1"
## Canal_plus "mars-05"  "1"   "14" "2"  "3"  "4"  "3"  "1"
## Arte       "mars-05"  "10"  "2"  "13" "4"  "0"  "9"  "3"
```

And if for our block we want only some rows or columns we can also select the rows and the columns in the `rs` and `cls` function parameters:

Example 1.a:

```
table3<-TrDtBbB(Data=table1,lengthBlock=7,cls=c(3:5,7),rs=c(1:2,4:5,7))
print(table3)
```

```
##            1  2  4  5  7
## TF1      214 27 14 31 36
## France_2 191 42 12 25 38
## France_3  88 35  4 15 24
## Arte      40  0  3  3 23
## TF1       42 41 27 46 14
## France_2  30 66 31 25 12
## France_3  13 23 22 12 12
## Arte      14  5  7  6  9
## TF1       47 48 33 57  7
## France_2  44 81 24 57  7
## France_3  27 38 15 14  1
## Arte      10  2  4  0  3
```

InvDtBbyB

Inversing data frame block by block

Description

This function takes any NxN data table and inverse its data block by block based on the parameters that we'll give to the function

Usage

InvDtBbyB<-function(Data=data(),lengthBlock=integer(),cls=c(),rs=c())

Arguments

Data	The data table or matrix that we're going to inverse block by block
lengthBlock	The number of the rows where the function will be referred to create the blocks for the invers porecess
cls	The columns that we want to include in the inversion process.
rs	The rows of the first block. Based on this rows the function will select identically the rows of every other block.

Details

Notice that in the `InvDtBbyB` function the column lenght must be equal to the row length of the block otherwise the fucntion will fail to give results. so the length of ***cls*** and ***rs*** parameters must be equal.

Example

For the example we are going to create a matrix with 9 rows and 3 columns that we will separate with 3 blocks of 3x3 and give the invers for the 3 blocs.

Example2:

```
set.seed(922)
UniDt<-matrix(runif(27),nrow = 9,ncol = 3)
UniDt
```

```
##             [,1]      [,2]       [,3]
##  [1,] 0.60869859 0.4690046 0.37645949
##  [2,] 0.38352539 0.6905934 0.67036358
##  [3,] 0.50077254 0.2410970 0.92302824
##  [4,] 0.97238599 0.4947444 0.06798979
##  [5,] 0.70045459 0.2981380 0.25770726
##  [6,] 0.04224092 0.2547806 0.15350140
##  [7,] 0.12777067 0.7615183 0.45259126
##  [8,] 0.96530657 0.2808974 0.08196870
##  [9,] 0.56762568 0.6317889 0.27986053
```

The result for `InvDtBbyB` function will be.

```
InvDt=NULL
InvDt<-InvDtBbyB(Data=UniDt,lengthBlock=3,cls=c(1:3),rs=c(1:3))
head(InvDt,6)
```

```
##             [,1]       [,2]       [,3]
## [1,]  2.56280781 -1.8428237  0.2931314
## [2,] -0.09859393  2.0107862 -1.4201529
## [3,] -1.36465281  0.4745695  1.2953043
## [4,]  0.35603518  1.0491148 -1.9190136
## [5,]  1.72942022 -2.6198691  3.6323864
## [6,] -2.96845465  4.0597428  1.0136676
```

Let's check the first two blocks of the Inverted data.
Example 2.a:

```
#First Block
UniDt[c(1:3),]%*%InvDt[c(1:3),]
```

```
##               [,1]          [,2] [,3]
## [1,]  1.000000e+00 0.000000e+00    0
## [2,]  0.000000e+00 1.000000e+00    0
## [3,] -4.440892e-16 1.110223e-16    1
```

```
#Second block
UniDt[c(4:6),]%*%InvDt[c(4:6),]
```

```
##              [,1]          [,2] [,3]
## [1,] 1.000000e+00 1.110223e-16    0
## [2,] 1.110223e-16 1.000000e+00    0
## [3,] 0.000000e+00 0.000000e+00    1
```

As we see, the product between the inverted blocks and the original blocks give to us identical matrices. so we can conclude that the function had inverted the data block by block.

ProdDtBbB

Produit of datas block by block

Description

This function is used in the case when we need to execute the product block by block of two different datas or matrices. we can also use it to obtain the squeare of each block of a data table or matrix.

Usage

ProdDtBbB<-function(Data1=data(),length_block1=integer(),cl_Dt1=c(),rows1=c(),Data2=matrix(),length_block2=integer(), cl_Dt2=c(),rows2=c())

Arguments

Data1	The first data table or matrix that we want to apply the product block by block with another data table (if we want a squeare product block by block, we can use the same Data2).
length_block1	The length of the blocks of Data1 that will multiply the blocks of the other matrix
cl_Dt1	The columns of Data1 that we want to include in the first data product process.The lenght of the columns of Data1 must be equal to the length of the rows of Data2.
rows1	The rows of the first block of Data1. Based on this rows the function will select identically the rows of every other block.
Data2	The second data table or matrix that we will apply the product function block by block of the first data table (or the same if we need the squeare value block by block of the data table.
length_block2	The length of the blocks that will be multiplied by the blocks of the first data table.
cl_Dt2	The columns of the second data frame that we want to include in the data product process.
rows2	The rows of the first block of the data frame. Based on this rows the function will select identically the rows of every other block. The lenght of the rows of Data2 must be equal to the lenght of the columns of Data1.

Details

As we've explained, the function can be used to produce the squeare values block by block of any data frame. For that we have to use the same data in both data parameters: `Data1` *=X1 and* `Data2` *=X1*
Also when we chose the parameters of the blocks for the `Data1` and `Data2` we must be careful that their value must be equal. As in any matrix production [M1(a,b)] x [(c,d)M2] we can't operate this product if b is different to a. The same thing is for the **cl_Dt1** and **rows2**.

Example

For the example we are going to create two data matrices with different number of rows and columns, so we can see the combination between the rows and the columns between the matrices.
Example 3:

```
set.seed(922)
d1<-matrix(rnorm(30),nrow =6,ncol =5)
d2<-matrix(runif(20),nrow = 10,ncol =2)
print(d1);print(d2)
```

```
##              [,1]        [,2]        [,3]       [,4]       [,5]
## [1,]  0.275928724 -0.01317422 -0.11911719 -2.1200212  1.0283787
## [2,]  0.001936479 -0.65952118 -0.58325586 -1.4283524 -0.3893446
## [3,]  0.525708397 -0.58017765  1.57153031 -0.2814728 -0.3159189
## [4,] -1.136992696 -0.31479279 -1.04327442  0.4413982  1.0750308
## [5,]  0.170332505  1.42573954  1.68974002  0.6125098  0.5811308
## [6,]  0.497532951 -0.65042997 -0.01923804  0.1316755 -0.2437851
```

```
##             [,1]       [,2]
##  [1,] 0.36398239 0.04240139
##  [2,] 0.02832807 0.18764200
##  [3,] 0.94562767 0.11955683
##  [4,] 0.82159395 0.79151725
##  [5,] 0.03517565 0.54844884
##  [6,] 0.30693517 0.56585236
##  [7,] 0.95607149 0.14737321
##  [8,] 0.35431054 0.70151794
##  [9,] 0.68557252 0.04711552
## [10,] 0.46021067 0.06829147
```

The function will give us the product of the matrices d1 and d2 block by block. Notice that *cl_Dt1* is equal to *rows2*

```
ProdM<-ProdDtBbB(Data1=d1,length_block1=3,cl_Dt1=c(1:5),rows1=c(1:3),Data2=d2
               ,length_block2=5,cl_Dt2=c(1:2),rows2=c(1:5))
print(ProdM)
```

```
##            [,1]       [,2]
## [1,] -1.7182033 -1.1190339
## [2,] -1.7567421 -1.5375051
## [3,]  1.4186268 -0.2947437
## [4,] -0.2222394 -1.3274256
## [5,]  2.7014451  1.5604268
## [6,] -0.4978829  0.1617339
```

And if we want the proof that the function worked in our example, we might execute the following manual calculus.

```
proof1<-d1[c(1:3),]%*%d2[c(1:5),]; proof2=d1[c(4:6),]%*%d2[c(6:10),]
ProdM-rbind(proof1,proof2)
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
## [3,]    0    0
## [4,]    0    0
## [5,]    0    0
## [6,]    0    0
```

In the next case let's try to execute the squeare value of d1, block by block.
Example 3.a:

```
ProdDtBbB(Data1=d1,length_block1=3,cl_Dt1=c(1:2),rows1=c(1,3),Data2=d1,length_block2=3,cl_Dt2
=c(1:2),rows2=c(1,3))
```

```
##             [,1]        [,2]
## [1,]  0.06921086 0.004008241
## [2,] -0.15994621 0.329680304
## [3,]  1.13613260 0.562667775
## [4,] -0.88930167 0.266439359
```

BlockTrMean

Mean of transposed Data block by block

Description

This function gives the rows mean or the columns mean of the blocks transposed data. It also gives the total mean of each transposed block.

Usage

BlockTrMean(Data=data(),lengthBlock=numeric(),cls=c(),rs=c(),Mean=character())

Arguments

Data	The data table or matrix that we're going to transpose block by block.
lengthBlock	The number of the rows where the function will be referred to create the transposed blocks.
cls	The columns that we want to include in the transposition.
rs	The rows of the first block. Based on this rows the function will select identically the rows of every other block.
Mean	By typing "Row", "Col" or "Block" it will give us the Mean of the rows or columns for each block or the total mean of every block.
time_col	The time parameter is optional. We can ignore it if our data has no time values.

Details

Notice that the `BlockTrMean` function can be used even in the cases where we want the same block mean processes without transposing the blocks. The only difference is that we have to apply "Row" when we want the Columns mean and vice versa.

Example

For the example we're going to use the same datas as in the example of trBlock and we will apply the three cases: Row, Col and Block mean.

The original table before the transposation by block is:

```
head(table1,9)
```

```
##       MOIS       THEMATIQUES TF1 France_2 France_3 Canal_plus Arte
## 1 janv.-05      Catastrophes 214      191        88          18   40
## 2 janv.-05   Culture_loisirs  27       42        35           4    0
## 3 janv.-05          Economie  35       18        10           1    8
## 4 janv.-05         Education  14       12         4           3    3
## 5 janv.-05     Environnement  31       25        15           1    3
## 6 janv.-05      Faits divers  24       19         9           1    1
## 7 janv.-05 Histoire_hommages  36       38        24          10   23
## 8 févr.-05      Catastrophes  42       30        13           4   14
## 9 févr.-05   Culture_loisirs  41       66        23          12    5
```

The result for `Mean=Row` will be
Example 4.1.a:

```
table2.1<-BlockTrMean(Data=table1,lengthBlock=7,cls=c(3:7),rs=c(1:7),Mean="Row")
print(table2.1)
```

```
##         TF1 France_2 France_3 Canal_plus      Arte
## rw 54.42857 49.28571 26.42857   5.428571 11.142857
## rw 33.28571 31.85714 17.00000   4.857143  6.857143
## rw 45.14286 41.57143 20.00000   4.000000  5.857143
```

The result for `Mean=Col` will be
Example 4.1.b:

```
table2.2<-BlockTrMean(Data=table1,lengthBlock=7,cls=c(3:7),rs=c(1:7),Mean="Col")
print(table2.2)
```

```
##        1    2    3    4    5    6    7
## cl 110.2 21.6 14.4  7.2 15.0 10.8 26.2
## cl  20.6 29.4 24.4 18.8 17.8 10.8  9.6
## cl  25.8 36.6 36.8 15.8 26.4 18.0  3.8
```

The result for `Mean=Block` will be
Example 4.1.c:

```
table2.3<-BlockTrMean(Data=table1,lengthBlock=7,cls=c(3:7),rs=c(1:7),time_col = 1,Mean="Bloc
k")
print(table2.3)
```

```
##    time
## bl "janv.-05" "29.3428571428571"
## bl "févr.-05" "18.7714285714286"
## bl "mars-05"  "23.3142857142857"
```

BlockTrSD

SD of transposed Data block by block

Description

This function gives the rows Standart Deviation (SD) or the columns SD of the blocks transposed data. It also give the total SD of each transposed block.

Usage

BlockTrSD(Data=data(),lengthBlock=numeric(),cls=c(),rs=c(),SD=character())

Arguments

Data	The data table or matrix that we're going to transpose block by block.
lengthBlock	The number of the rows where the function will be referred to create the transposed blocks.
cls	The columns that we want to include in the transposition.
rs	The rows of the first block. Based on this rows the function will select identically the rows of every other block.
SD	By typing "Row", "Col" or "Block" it will give us the SD of the rows or columns for each block or the total SD of every block.
time_col	The time parameter is optional. We can ignore it if our data has no time values.

Details

Notice that the `BlockTrSD` function can be used even in the cases where we want the same block SD processes without transposing the blocks. The only difference is that we have to apply "Row" when we want the Columns SD and vice versa.

Example

For the example we're going to use the same datas as in the example of trBlock and we will apply the three cases: Row, Col and Block SD.

The result for `SD=Row` will be

Example 4.2.a:

```
table3.1<-BlockTrSD(Data=table1,lengthBlock=7,cls=c(3:7),rs=c(1:7),SD="Row")
print(table3.1)
```

```
##         TF1 France_2  France_3 Canal_plus      Arte
## rw 70.75982 63.42900 29.090990   6.399405 14.960265
## rw 13.75638 18.19733  5.656854   4.336995  4.220133
## rw 26.21341 24.35061 11.888370   4.546061  4.810702
```

The result for `SD=Col` will be
Example 4.2.b:

```
table3.2<-BlockTrSD(Data=table1,lengthBlock=7,cls=c(3:7),rs=c(1:7),SD="Col")
print(table3.2)
```

```
##           1        2        3         4        5         6         7
## cl 88.35270 18.71630 13.01153  5.357238 13.19091 10.449880 11.322544
## cl 15.19210 24.56217 20.10721 11.233877 18.28114  6.978539  5.128353
## cl 20.29039 30.86746 35.68893 12.911235 28.39542 12.267844  3.033150
```

The result for `SD=Block` will be
Example 4.2.c:

```
table3.3<-BlockTrSD(Data=table1,lengthBlock=7,cls=c(3:7),rs=c(1:7),time_col = 1,SD="Block")
print(table3.3)
```

```
##    time
## bl "janv.-05" "46.7841997317237"
## bl "févr.-05" "15.8839277152682"
## bl "mars-05"  "23.8139497286921"
```

LargeTrMean

Mean by block of large transposed Data

Description

This function gives the rows mean or the columns mean of transposed blocks of a large data. It also gives the total mean of each transposed block. For this package there is no need to do a transposed large data by block to long data because it can easly executed by apply the transposed function. The interest of this function is that it can transpose and give the mean block by block of a large data frame or matrix.

Usage

LargeTrMean(Data=data(),lengthBlock=numeric(),cls=c(),rs=c(),Mean=character())

Arguments

Data	The data table or matrix that we're going to transpose by blocks.
lengthBlock	The number of the rows of the first block that will be transposed. The function will transpose the rest of the blocks based on the value of the first block.
cls	The columns of the first block. Based on this columns the function will select identically the rows of every other block.
rs	The rows that we want to include in the transposition.
Mean	By typing "Row", "Col" or "Block" it will give us the Mean of the rows or columns for each block or the total mean of every block.

Details

Notice that the `largeTrMean` function can be used even in the cases where we want the same block mean processes without transposing the blocks. The only difference is that we have to apply "Row" when we want the Columns mean and vice versa.

Example

For the example we're going to use some large datas with four potential blocks and we'll apply the three cases: Row, Col and Block mean.

The large data table before our operations is:

```
head(largeDt)
```

```
##   THEMATIQUES   a  b  c  d a.1 b.1 c.1 d.1 a.2 b.2 c.2 d.2 a.3 b.3 c.3 d.3
## 1          C1 214 27 35 14  31  24  42  41  46  27  46  17  47  48  92  33
## 2          C2 191 42 18 12  25  19  30  66  43  31  25  16  44  81  50  24
## 3          C3  88 35 10  4  15   9  13  23  24  22  12  13  27  38  27  15
## 4          C4  18  4  1  3   1   1   4  12   2   7   0   8   1  14   2   3
## 5          C5  40  0  8  3   3   1  14   5   7   7   6   0  10   2  13   4
```

The result for `Mean=Row` will be
Example 4.3.a:

```
largeDt1.1<-LargeTrMean(Data=largeDt,lengthBlock=6,cls=c(2,4:5),rs=c(1:4),Mean = "Row")
print(largeDt1.1)
```

```
##       rw    rw
## a 127.75 22.25
## c  16.00 28.75
## d   8.25 21.75
```

The result for `Mean=Col` will be
Example 4.3.b:

```
largeDt1.2<-LargeTrMean(Data=largeDt,lengthBlock=6,cls=c(2,4:5),rs=c(1:4),Mean ="Col")
print(largeDt1.2)
```

```
##          cl        cl
## 1 87.666667 38.333333
## 2 73.666667 34.666667
## 3 34.000000 19.666667
## 4  7.333333  4.333333
```

The result for `Mean=Block` will be
Example 4.3.c:

```
largeDt1.3<-LargeTrMean(Data=largeDt,lengthBlock=6,cls=c(2,4:5),rs=c(1:4),Mean ="Block")
print(largeDt1.3)
```

```
##            bl    bl
## [1,] 50.66667 24.25
```

LargeTrSD

SD by block of large transposed Data

Description

This function gives the rows SD or the columns SD of transposed blocks of a large data. It also gives the total SD of each transposed block.

Usage

LargeTrSD(Data=data(),lengthBlock=numeric(),cls=c(),rs=c(),SD=character())

Arguments

Data	The data table or matrix that we're going to transpose by blocks.
lengthBlock	The number of the rows of the first block that will be transposed. The function will transpose the rest of the blocks based on the value of the first block.
cls	The columns of the first block. Based on this columns the function will select identically the rows of every other block.
rs	The rows that we want to include in the transposition.
SD	By typing "Row", "Col" or "Block" it will give us the SD of the rows or columns for each block or the total mean of every block.

Details

Notice that the `largeTrSD` function can be used even in the cases where we want the same block SD processes without transposing the blocks. The only difference is that we have to apply "Row" when we want the Columns mean and vice versa.

Example

For the example we're going to use the same large datas as in the examples that we used to demostrate the largeTrMean. By applying the three cases: Row, Col and Block SD we are going to have the next results.

```
head(largeDt,3)
```

```
##   THEMATIQUES   a  b  c  d a.1 b.1 c.1 d.1 a.2 b.2 c.2 d.2 a.3 b.3 c.3 d.3
## 1          C1 214 27 35 14  31  24  42  41  46  27  46  17  47  48  92  33
## 2          C2 191 42 18 12  25  19  30  66  43  31  25  16  44  81  50  24
## 3          C3  88 35 10  4  15   9  13  23  24  22  12  13  27  38  27  15
```

The result for `SD=Row` will be
Example 4.4.a:

```
largeDt2.1<-LargeTrSD(Data=largeDt,lengthBlock=6,cls=c(2,4:5),rs=c(1:4),SD = "Row")
print(largeDt2.1)
```

```
##          rw       rw
## a 91.405233 17.01715
## c 14.445299 20.32035
## d  5.560276 10.50000
```

The result for `SD=Col` will be
Example 4.4.b:

```
largeDt2.2<-LargeTrSD(Data=largeDt,lengthBlock=6,cls=c(2,4:5),rs=c(1:4),SD ="Col")
print(largeDt2.2)
```

```
##           cl        cl
## 1 109.910570 10.016653
## 2 101.657923  7.234178
## 3  46.861498  5.859465
## 4   9.291573  2.516611
```

The result for `SD=Block` will be
Example 4.4.c:

```
largeDt2.3<-LargeTrSD(Data=largeDt,lengthBlock=6,cls=c(2,4:5),rs=c(1:4),SD ="Block")
print(largeDt2.3)
```

```
##           bl       bl
## [1,] 74.80561 15.25615
```