Recoding Matlab code of MR-GRAS to R - Alexandre Bourgeois 01/08/2021

Recoding Matlab code of MR-GRAS to R

REFERENCES : Temursho , U., Oosterhaven , J. and M.A. Cardenete (2019) , A multi - regional generalized RAS updating technique , IOpedia Research Paper No. 2, September 2019 , www. IOpedia .eu

```
#REFERENCES article : Temursho , U., Oosterhaven , J. and M.A. Cardenete (2019) , A multi - regional ge

#M 1 function [X,r,s,T] = mrgras (X0 ,u,v,G,Q,W, eps )
mrgras <- function(X0 ,u,v,G,Q,W, eps=0.00001,theshld=5000) {
  #M 2 % PURPOSE : estimate a new multi - regional (or any partitioned ) matrix X as
  #M 3 % close as possible to a given matrix X0 subject to the row sums , column
  #M 4 % sums and non - overlapping aggregation constraints , using MR - GRAS approach
  #M 5 % ----------------------------------------------------------------------
  #M 6 % USAGE :
  #M 7 % Write X = mrgras (X0 ,u,v,G,Q,W) OR [X,r,s,T] = mrgras (X0 ,u,v,G,Q,W) with
  #M 8 % or without eps as the seventh optional input argument , where
  #M 9 %
  #M 10 % INPUT :
  #M 11 % -> X0 = benchmark ( original ) matrix , not necessarily square
  #M 12 % -> u = column vector of row totals (new row sums )
  #M 13 % -> v = column vector of column totals ( new column sums )
  #M 14 % -> W = non - overlapping aggregation constraints matrix
  #M 15 % -> G & Q = the row and column aggregator matrices such that G*X0*Q = W;
  #M 16 % non - overlapping aggregation necessarily implies that the
  #M 17 % column sums of G and the row sums of Q must be all unity
  #M 18 % -> eps = convergence tolerance level ; if empty , the default threshold
  #M 19 % level is 0.1e -5 (=0.000001)
  #M 20 % -> In case of missing w_IJ , set the corresponding missing number to
  #M 21 % w_IJ = 1010101 ( assuming that 1010101 is not among the w_IJ 's)
  #M 22 %
  #M 23 % OUTPUT (input - output analysis - related interpretation ):
  #M 24 % -> X = updated / balanced / adjusted / projected matrix
  #M 25 % -> r = substitution effects ( row multipliers )
  #M 26 % -> s = fabrication effects ( column multipliers )
  #M 27 % -> T = technology or regional effects ( aggregation multipliers )
  #M 28 % ----------------------------------------------------------------------
  #M 29 % REFERENCES :
  #M 30 % Temursho , U., Oosterhaven , J. and M.A. Cardenete (2019) , A multi - regional
  #M 31 % generalized RAS updating technique , IOpedia Research Paper No. 2,
  #M 32 % September 2019 , www. IOpedia .eu
  #M 33 % ----------------------------------------------------------------------
  #M 34 % NOTE FROM THE AUTHOR : Using this program and publishing the results in
  #M 35 % the form of a report , working / discussion paper , journal article , etc .
  #M 36 % requires citation of the above reference paper .
  #M 37 % ----------------------------------------------------------------------
```

```
#M 38 % Written by: Umed Temursho , May 2019
#M 39 % E- mail : utemursho@gmail . com
#M 40
#M 41 [m,n] = size (X0);
m<-nrow(X0)
n<-ncol(X0)
#M 42 [h,k] = size (W);
h<-nrow(W)
k<-ncol(W)
#M 43 N = zeros (m,n);
N<-matrix(0,m,n)
#M 44 N(X0 <0) = -X0(X0 <0) ;
N[X0 <0]<--X0[X0<0]
#M 45 N = sparse (N); % could save memory with large - scale matrices
#M 46 P = X0+N;
P<-X0+N
#M 47 P = sparse (P);
#M 48 %
#M 49 r0 = ones (m ,1) ; % initial guess for r in step 0
r0<-matrix(1,m,1)
r0<-as.vector(r0)
#M 50 T0 = ones (h,k); % initial guess for T in step 0
T0<-matrix(1,h,k)
#M 51 Te = G '* T0*Q '; % T expanded to fit the dimention of X0
Te<-t(G)%*%T0%*%t(Q)
#M 52 p_rt = (P.* Te) '*r0;
p_rt<-t(P*Te)%*%r0
p_rt<-as.vector(p_rt)

#M 53 n_rt = (N.* invM (Te)) '* invd (r0)* ones (m ,1);
OneM<-matrix(1,m,1)
OneM<-as.vector(OneM)
n_rt <-t(N*invM(Te))%*%invd(r0)%*%OneM
n_rt<-as.vector(n_rt)

#M 54 s1 = invd (2* p_rt )*(v+ sqrt (v .^2+4* p_rt .* n_rt )); % first step s
s1<-invd(2*p_rt)%*%(v+sqrt(v^2+4*p_rt*n_rt))
s1<-as.vector(s1)

#M 55 ss = -invd (v)* n_rt ;
ss<--invd(v)%*%n_rt
ss<-as.vector(ss)

#M 56 s1( p_rt ==0) = ss( p_rt ==0) ;
s1[p_rt==0]<-ss[p_rt==0]

#M 57 %
#M 58 p_st = (P.* Te)*s1;
p_st<-(P*Te)%*%s1
p_st<-as.vector(p_st)
#M 59 n_st = (N.* invM (Te))* invd (s1)* ones (n ,1) ;
OneN<-matrix(1,n,1)
OneN<-as.vector(OneN)
```

```
# n_st<-(N*invM(Te))*invd(s1)*vect_1_n    Version origine
n_st<-(N*invM(Te))%*%invd(s1)%*%OneN
n_st<-as.vector(n_st)
#M 60 r1 = invd (2* p_st )*(u+ sqrt (u .^2+4* p_st .* n_st )); % first step r
r1<-invd(2*p_st)%*%(u+sqrt(u^2+4*p_st*n_st ))
r1<-as.vector(r1)
#M 61 rr = -invd (u)* n_st ;
rr<--invd(u)%*%n_st
rr<-as.vector(rr)
#M 62 r1( p_st ==0) = rr( p_st ==0) ;
r1[p_st==0]<-rr[p_st==0]

#M 63 %
#M 64 P_rs = G*( diag (r1)*P* diag (s1))*Q;
P_rs<-G%*%(diag(r1)%*%P%*%diag(s1))%*%Q
#M 65 N_rs = G*( invd (r1)*N* invd (s1))*Q;
N_rs<-G%*%(invd(r1)%*%N%*%invd(s1))%*%Q
#M 66 T1 = invM (2* P_rs ) .*( W+ sqrt (W .^2+4* P_rs .* N_rs )); % first step T
T1<-invM(2*P_rs)*(W+sqrt(W^2+4*P_rs*N_rs))
#M 67 TT = -invM (W).* N_rs ;
TT<--invM(W)*N_rs
#M 68 T1( P_rs ==0) = TT( P_rs ==0) ;
T1[P_rs==0]<-TT[P_rs==0]
#M 69 T1(W ==1010101) = 1; % set t_IJ =1 for missing aggregation total w_IJ
T1[W ==1010101]<-1
#M 70 Te = G '* T1*Q ';
T1<-as.matrix(T1)
Te<-t(G)%*%T1%*%t(Q)
#M 71 %
#M 72 p_rt = (P.* Te) '*r1;
p_rt<-t(P*Te)%*%r1
p_rt<-as.vector(p_rt)
#M 73 n_rt = (N.* invM (Te)) '* invd (r1)* ones (m ,1);
n_rt<-t(N*invM(Te))%*%invd(r1)%*%OneM
n_rt<-as.vector(n_rt)
#M 74 s2 = invd (2* p_rt )*(v+ sqrt (v .^2+4* p_rt .* n_rt )); % second step s
s2<-invd(2*p_rt)%*%(v+sqrt(v^2+4*p_rt*n_rt ))
s2<-as.vector(s2)
#M 75 ss = -invd (v)* n_rt ;
ss<--invd(v)%*%n_rt
ss<-as.vector(ss)
#M 76 s2( p_rt ==0) = ss( p_rt ==0) ;
s2[p_rt==0]<-ss[p_rt==0]

#M 77 %
#M 78 p_st = (P.* Te)*s2;
p_st<-(P*Te)%*%s2
p_st<-as.vector(p_st)
#M 79 n_st = (N.* invM (Te))* invd (s2)* ones (n ,1) ;
n_st<-(N*invM(Te))%*%invd(s2)%*%OneN
n_st<-as.vector(n_st)
#M 80 r2 = invd (2* p_st )*(u+ sqrt (u .^2+4* p_st .* n_st )); % second step r
r2<-invd(2*p_st)%*%(u+sqrt(u^2+4*p_st*n_st ))
```

```
r2<-as.vector(r2)
#M 81 rr = -invd (u)* n_st ;
rr<--invd(u)%*%n_st
rr<-as.vector(rr)
#M 82 r2( p_st ==0) = rr( p_st ==0) ;
r2[p_st==0]<-rr[p_st==0]

#M 83 %
#M 84 P_rs = G*( diag (r2)*P* diag (s2))*Q;
P_rs<-G%*%(diag(r2)%*%P%*%diag(s2))%*%Q
#M 85 N_rs = G*( invd (r2)*N* invd (s2))*Q;
N_rs<-G%*%(invd(r2)%*%N%*%invd(s2))%*%Q
#M 86 T2 = invM (2* P_rs ) .*( W+ sqrt (W .^2+4* P_rs .* N_rs )); % second step T
T2<-invM(2*P_rs)*(W+sqrt(W^2+4*P_rs*N_rs ))
#M 87 TT = -invM (W).* N_rs ;
TT<--invM(W)*N_rs
#M 88 T2( P_rs ==0) = TT( P_rs ==0) ;
T2[P_rs==0]<-TT[P_rs==0]
#M 89 T2(W ==1010101) = 1; % set t_IJ =1 for missing w_IJ
T2[W==1010101]<-1

#M 90 %
#M 91 tmax = max ( max ( abs (T2 -T1)));
tmax<-max(abs(T2-T1))
#M 92 dif = [s2 -s1;r2 -r1; tmax ];
dif<-rbind(c(s2-s1),c(r2-r1),tmax)
#M 93 iter = 1 % first iteration
iter<-1
#M 94 if nargin < 7 || isempty ( eps)
#M 95 eps = 0.1e -5; % default tolerance level
#M 96 end
if(nargs()<7 || is.null(eps)){eps<-0.000001}
#M 97 M = max ( abs ( dif ));
M<-max(abs(dif))

#M 98 while (M > eps )
iter=0
while(M>eps & iter < theshld){  # Default conditions : M>eps or 5000 iterations
  #M 99 s1 = s2;
  s1<-s2
  #M 100 r1 = r2;
  r1<-r2
  #M 101 T1 = T2;
  T1<-T2
  #M 102 Te = G '* T1*Q ';
  T1<-as.matrix(T1)
  Te<-t(G)%*%T1%*%t(Q)
  #M 103 %
  #M 104 p_rt = (P.* Te) '*r1;
  p_rt<-t(P*Te)%*%r1
  p_rt<-as.vector(p_rt)
  #M 105 n_rt = (N.* invM (Te)) '* invd (r1)* ones (m ,1);
  n_rt<-t(N*invM(Te))%*%invd(r1)%*%OneM
```

```
  n_rt<-as.vector(n_rt)
  #M 106 s2 = invd (2* p_rt )*(v+ sqrt (v .^2+4* p_rt .* n_rt )); % next step s
  s2<-invd(2*p_rt)%*%(v+sqrt(v^2+4*p_rt*n_rt))
  s2<-as.vector(s2)
  #M 107 ss = -invd (v)* n_rt ;
  ss<--invd(v)%*%n_rt
  ss<-as.vector(ss)
  #M 108 s2( p_rt ==0) = ss( p_rt ==0) ;
  s2[p_rt==0]<-ss[p_rt==0]

  #M 109 %
  #M 110 p_st = (P.* Te)*s2;
  p_st<-(P*Te)%*%s2
  p_st<-as.vector(p_st)
  #M 111 n_st = (N.* invM (Te))* invd (s2)* ones (n ,1) ;
  n_st<-(N*invM(Te))%*%invd(s2)%*%OneN
  n_st<-as.vector(n_st)
  #M 112 r2 = invd (2* p_st )*(u+ sqrt (u .^2+4* p_st .* n_st )); % next step r
  r2<-invd(2*p_st)%*%(u+sqrt(u^2+4*p_st* n_st))
  r2<-as.vector(r2)
  #M 113 rr = -invd (u)* n_st ;
  rr<--invd(u)%*%n_st
  rr<-as.vector(rr)
  #M 114 r2( p_st ==0) = rr( p_st ==0) ;
  r2[p_st==0]<-rr[p_st==0]

  #M 115 %
  #M 116 P_rs = G*( diag (r2)*P* diag (s2))*Q;
  P_rs<-G%*%(diag(r2)%*%P%*%diag(s2))%*%Q
  #M 117 N_rs = G*( invd (r2)*N* invd (s2))*Q;
  N_rs<-G%*%(invd(r2)%*%N%*%invd(s2))%*%Q
  #M 118 T2 = invM (2* P_rs ) .*( W+ sqrt (W .^2+4* P_rs .* N_rs )); % next step T
  T2<-invM(2*P_rs)*(W+sqrt(W^2+4*P_rs*N_rs))
  #M 119 TT = -invM (W).* N_rs ;
  TT<--invM(W)*N_rs
  #M 120 T2( P_rs ==0) = TT( P_rs ==0) ;
  T2[P_rs==0]<-TT[P_rs==0]
  #M 121 T2(W ==1010101) = 1; % set t_IJ =1 for missing w_IJ
  T2[W==1010101]<-1

  #M 122 %
  #M 123 tmax = max ( max ( abs (T2 -T1)));
  tmax<-max(abs(T2-T1))
  #M 124 dif = [s2 -s1;r2 -r1; tmax ];
  dif<-rbind(c(s2-s1),c(r2-r1),tmax)
  #M 125 iter = iter +1
  iter <- iter +1
  #M 126 M = max ( abs ( dif ));
  M<-max(abs(dif))
  #M 127 end
}
print(paste0(iter," iterations required to ensure convergence (if equal to threshold : check converge
#M 128 s = s2; % final step s
```

```
  s <- s2
  #M 129 r = r2; % final step r
  r <- r2
  #M 130 T = T2; % final step T
  Tfin <- T2
  #M 131 Te = G '*T*Q ';
  Tfin<-as.matrix(Tfin)
  Te <- t(G)%*%Tfin%*%t(Q)
  #M 132 %
  #M 133 X = Te .*( diag (r)*P* diag (s))-invM (Te) .*( invd (r)*N* invd (s)); % updated matrix
  X <- Te*(diag(r)%*%P%*%diag(s))-invM(Te)*(invd(r)%*%N%*%invd(s))
  #M 134 end
  return(list(X,r,s,Tfin))
}
#M 135
#M 136 function invd = invd (x) % auxiliary function used above
invd <- function(x) {
  #M 137 invd = 1./ x;
  sortie<-1/x
  #M 138 invd (x ==0) = 1;
  sortie[x==0]<-1
  #M 139 invd = diag ( invd );
  #M 140 end
  return(diag(sortie))
}
#M 141
#M 142 function invM = invM (X) % auxiliary function used above
invM <- function(X) {
  #M 143 invM = 1./ X;
  sortie<-1/X
  #M 144 invM (X ==0) = 1;
  sortie[X==0]<-1
  #M 145 end
  return(sortie)
}
#M 146 % --------------------------------------------------------------------------
#M 147 % END OF THE CODE
#M 148 % --------------------------------------------------------------------------
```

Test of the code

We use the example given in the article (Updating with exhaustive constraints p.23)

```
datamatTEI<-c(63,9,14,9,-18,75,-14,53,-10,66,69,66,16,56,-21,9,93,-25,53,16,74,72,-1,80,4,-48,14,64,51,
datamatUcible<-c(160,194,145,320,134,151)
datamatVcible<-c(197,71,151,242,178,265)
Wcible<-c(230,0,250,123,75,130,86,174,36)
datamatTEIupdatedTarget<-c(74.2,8.2,16.4,10.6,-21.5,72.1,-13.4,44.4,-10.4,68.5,52.8,52.2,18.8,64.8,-19.
Gagg<-cbind(diag(3),diag(3))
Qagg<-rbind(diag(3),diag(3))

Mcible<-matrix(Wcible,nrow=3,ncol=3,byrow=TRUE)
Xtarget<-matrix(datamatTEIupdatedTarget,nrow=6,ncol=6,byrow=TRUE)
print(Xtarget)
```

```
##       [,1]  [,2]  [,3] [,4]  [,5]  [,6]
## [1,]  74.2   8.2  16.4 10.6 -21.5  72.1
## [2,] -13.4  44.4 -10.4 68.5  52.8  52.2
## [3,]  18.8  64.8 -19.3 10.5  98.3 -28.0
## [4,]  61.7  14.5  85.5 83.5  -1.2  76.0
## [5,]   4.0 -59.6  12.9 63.9  37.5  75.3
## [6,]  51.7  -1.2  65.9  5.1  12.2  17.4
```

```
# With paper's names
X0<-matrix(datamatTEI,nrow=6,ncol=6,byrow=TRUE)
U<-as.vector(datamatUcible)
V<-as.vector(datamatVcible)
W<-matrix(Wcible,nrow=3,ncol=3,byrow=TRUE)
G<-Gagg
Q<-Qagg
W0<-G %*% X0 %*% Q

resultat<-mrgras(X0 ,U,V,G,Q,W, eps=0.0000001,theshld=1000)
```

```
## [1] "12 iterations required to ensure convergence (if equal to threshold : check convergence)"
```

```
print(resultat[[1]])
```

```
##            [,1]       [,2]      [,3]      [,4]       [,5]      [,6]
## [1,]  74.245480   8.241533  16.36994 10.556140 -21.513217  72.10012
## [2,] -13.422031  44.359339 -10.39911 68.515197  52.766704  52.17990
## [3,]  18.777638  64.750609 -19.31884 10.512274  98.251786 -27.97346
## [4,]  61.732934  14.480949  85.51896 83.465446  -1.209264  76.01098
## [5,]   4.012449 -59.633774  12.94707 63.894384  37.507731  75.27214
## [6,]  51.653535  -1.198657  65.88199  5.056554  12.196261  17.41032
```

The algorithm converge before the threshold, and we will check consistency and precision.

If the algorithm had'nt converge, you should investigate the convergence and check source data (is there any NA ou NaN or NULL or something like that ?).

Deviations at fine levels :

```
print(resultat[[1]]-Xtarget)
```

```
##             [,1]         [,2]          [,3]         [,4]         [,5]
## [1,]  0.04547997  0.041532526 -0.0300577161 -0.043859806 -0.013217309
## [2,] -0.02203075 -0.040660515  0.0008870869  0.015197177 -0.033296244
## [3,] -0.02236207 -0.049390616 -0.0188439845  0.012273654 -0.048213688
## [4,]  0.03293418 -0.019050787  0.0189603394 -0.034554344 -0.009264431
## [5,]  0.01244911 -0.033774104  0.0470695859 -0.005615539  0.007730863
## [6,] -0.04646537  0.001342976 -0.0180146936 -0.043446215 -0.003738672
##               [,6]
## [1,]  0.0001221746
## [2,] -0.0200971144
## [3,]  0.0265375885
## [4,]  0.0109752022
## [5,] -0.0278595585
## [6,]  0.0103210897
```

The convergence is good at 1 decimal, wich is consistent with the target.

The algorithm reaches convergence at 10^-6 in 11 iterations, and in R it is near to be the same.

Deviations at agregated level :

```
Wtarget<-G %*% resultat[[1]] %*% Q
print(Wtarget)
```

```
##      [,1]          [,2] [,3]
## [1,]  230 -7.105427e-15  250
## [2,]  123  7.500000e+01  130
## [3,]   86  1.740000e+02   36
```

```
print(Wtarget-Mcible)
```

```
##              [,1]          [,2]         [,3]
## [1,] 0.000000e+00 -7.105427e-15 0.000000e+00
## [2,] 2.842171e-14 -1.421085e-14 2.842171e-14
## [3,] 0.000000e+00  2.842171e-14 7.105427e-15
```

We find a very good precision on the 3rd dimension.

Consistency of the algorithm :

```
Mresult<-resultat[[1]]
print(rowSums(Mresult)-U)
```

```
## [1] -1.556154e-07 -3.618795e-07  8.867970e-07  1.556154e-07  3.618795e-07
## [6] -8.867970e-07
```

```
print(colSums(Mresult)-V)
```

```
## [1]  5.073102e-06 -5.193373e-07  6.179617e-07 -5.073102e-06  5.193373e-07
## [6] -6.179616e-07
```

```
print(sum(rowSums(Mresult)))
```

```
## [1] 1104
```

```
print(sum(colSums(Mresult)))
```

```
## [1] 1104
```

```
print(Mresult[1,1]+Mresult[4,1]+Mresult[1,4]+Mresult[4,4]) # we should find 230
```

```
## [1] 230
```

Discrepancies un row sums and col sums are under the threshold, and even better on the 3rd dimension.