

Non convexity issue

Chanwoo Lee

Dec 03, 2019

1 Non convexity in the intercept and the slope

Let us assume $\Theta = \mathcal{C} \times_1 A_1 \times_2 A_2 \times_3 A_3 = g(\mathcal{C}, A_1, A_2, A_3)$. For CP decomposition model, the log-likelihood is

$$\mathcal{L}_Y(\mathcal{C}, A_1, A_2, A_3, \boldsymbol{\omega}) = \sum_{l \in [K]} \sum_{y_{ijk}=l} \log(f_l(g(\mathcal{C}, A_1, A_2, A_3))).$$

where $f_l(\theta) = \phi(w_l + \theta) - \phi(w_{l-1} + \theta)$, and ϕ is logistic function such that $\phi(x) = \frac{1}{1+e^{-x}}$. By the discussion in [1] and [2], this log-likelihood function is concave as long as a function g of parameters is linear. Unfortunately, this is not the case because of the structure of CP-decomposition structure, which makes g non linear function with respect to $(\mathcal{C}, A_1, A_2, A_3)$. However, we can prove that the log-likelihood with respect to $(\boldsymbol{\omega}, \mathcal{C})$ is concave for fixed A_1, A_2, A_3 . It is based on the vectorization of Θ .

$$\text{vec}(\Theta) = (A_3 \otimes A_2 \otimes A_1) \text{vec}(\mathcal{C}) = A \text{vec}(\mathcal{C}).$$

where $A = A_3 \otimes A_2 \otimes A_1$. Let us define a function $\text{Index} : \mathbb{R}^3 \rightarrow \mathbb{R}$ such that $\Theta_{ijk} = \text{vec}(\Theta)_{\text{Index}(i,j,k)}$. From the above formulas, we can write the log-likelihood function for fixed A_1, A_2, A_3 as

$$\mathcal{L}_Y(\mathcal{C}, \boldsymbol{\omega} | A_1, A_2, A_3) = \sum_{l \in [K]} \sum_{y_{ijk}=l} \log(f_l(g(\mathcal{C} | A_1, A_2, A_3))) = \sum_{l \in [K]} \sum_{y_{ijk}=l} \log(f_l(a_{\text{Index}(i,j,k)}^T \text{vec}(\mathcal{C}))).$$

where a_i is i -th row of A . Therefore, we have linear function g with respect to \mathcal{C} .

2 \mathcal{C} and ω together

To update \mathcal{C} and ω together, I changed several part of function.R code.

1. I used “optim” function instead of “nlminb” function. There are the main two reasons for this change. Firstly, calculating Hessian with respect to (ω, \mathcal{C}) is not an easy work. Secondly, ω has a constraint that $\omega_1 < \omega_2 \cdots < \omega_{K-1}$. To perform constrained optimization, “optim” function is better.
2. In the previous codes, ω is only updated once. For this reason, I changed the structure of the codes to make sure that ω is updated for each iteration when ω is unknown.
3. I changed order of updates when the threshold α is given. In the previous codes, order of updates remains the same as our first codes. To get α constrained optimizer, I put (ω, \mathcal{C}) in front.

3 Code

```
1 library(MASS)
2 library(rTensor)
3 library(pracma)
4 library(ggplot2)
5 library(ggthemes)
6 library(gridExtra)
7 library(Matrix)
8
9
10 likelihood = function(ttnsr, thet, alpha){
11   p1 = logistic(c(thet) + alpha[1])
12   p2 = logistic(c(thet) + alpha[2])
13   p = cbind(p1, p2-p1, 1-p2)
14   return(-sum(log(c(p[which(c(ttnsr)==1), 1], p[which(c(ttnsr)==2), 2], p[
15     which(c(ttnsr)==3), 3]))))
16 }
17
18 logistic = function(x){
19   return(1/(1+exp(-x)))
20 }
```

```

19 }
20
21 ##### simulate ordinal tensors based on logistic model
22 # realization = function(tnsr,alpha){
23 #   tnsr=as.tensor(tnsr)
24 #   thet <- k_unfold(tnsr,1)@data
25 #   theta1 <- thet + alpha[1]
26 #   theta2 <- thet + alpha[2]
27 #   result <- k_unfold(tnsr,1)@data
28 #   p1 <- logistic(theta1)
29 #   p2 <- logistic(theta2)-logistic(theta1)
30 #   p3 <- matrix(1,nrow = nrow(thet),ncol = ncol(thet))-logistic(theta2)
31 #   for (i in 1:nrow(thet)) {
32 #     for(j in 1:ncol(thet)){
33 #       result[i,j] <- sample(c(1,2,3),1,prob= c(p1[i,j],p2[i,j],p3[i,j]))
34 #     }
35 #   }
36 #   return(k_fold(result,1,modes = tnsr@modes))
37 # }
38
39
40 # ##### Hessian w.r.t. A_1 #####
41 # Hessi = function(A_1,W1,ttnsr,omega){
42 #   thet =W1%%c(A_1)
43 #   p1 = logistic(thet + omega[1])
44 #   p2 = logistic(thet + omega[2])
45 #   q1 = p1*(1-p1)
46 #   q2 = p2*(1-p2)+p1*(1-p1)
47 #   q3 = p2*(1-p2)
48 #   H = t(rbind(W1[which(c(ttnsr)==1),])*q1[which(c(ttnsr)==1)])%%rbind(
49 #     W1[which(c(ttnsr)==1),]+t(rbind(W1[which(c(ttnsr)==2),])*q2[which(c(
50 #       ttnsr)==2)])%%rbind(W1[which(c(ttnsr)==2),]+t(rbind(W1[which(c(ttnsr)
51 #         ==3),])*q3[which(c(ttnsr)==3)])%%rbind(W1[which(c(ttnsr)==3),])
52 #   return(H)
53 # }
54 #
55 # ##### cost function #####
56 # h1 = function(A_1,W1,ttnsr,omega){
57 #   thet =W1%%c(A_1)

```

```

55 #   p1 = logistic(thet + omega[1])
56 #   p2 = logistic(thet + omega[2])
57 #   p = cbind(p1,p2-p1,1-p2)
58 #   return(-sum(log(c(p[which(c(ttnsr)==1),1],p[which(c(ttnsr)==2),2],p[
        which(c(ttnsr)==3),3]))))
59 # }
60 #
61 # ##### gradient #####
62 # g1 = function(A_1,W1,ttnsr,omega){
63 #   thet =W1%%c(A_1)
64 #   p1 = logistic(thet + omega[1])
65 #   p2 = logistic(thet + omega[2])
66 #   q1 <- p1-1
67 #   q2 <- (p2*(1-p2)-p1*(1-p1))/(p1-p2)
68 #   q3 <- p2
69 #   gd = apply(rbind(W1[which(c(ttnsr)==1),])*q1[which(c(ttnsr)==1)],2,sum
        )+apply(rbind(W1[which(c(ttnsr)==2),])*q2[which(c(ttnsr)==2)],2,sum)+
        apply(rbind(W1[which(c(ttnsr)==3),])*q3[which(c(ttnsr)==3)],2,sum)
70 #
71 #   return(gd)
72 # }
73
74 ##### Hessian w.r.t. A_1 #####
75 Hessi = function(A_1,W1,ttnsr,omega){
76   k = length(omega)
77   thet =W1%%c(A_1)
78   p = matrix(nrow = length(thet),ncol = k)
79   for (i in 1:k) {
80     p[,i] = logistic(thet + omega[i])
81   }
82   q = matrix(nrow = length(thet),ncol = k+1)
83   q[,1] = p[,1]*(1-p[,1])
84   for (i in 2:k) {
85     q[,i] = p[,i]*(1-p[,i])+p[,i-1]*(1-p[,i-1])
86   }
87   q[,k+1] = p[,k]*(1-p[,k])
88   l= lapply(1:(k+1),function(i) t(rbind(W1[which(c(ttnsr)==i),])*q[which(c
        (ttnsr)==i),i])%%rbind(W1[which(c(ttnsr)==i),])))
89   return(Reduce("+", l))

```

```

90 }
91
92
93 ##### cost function #####
94 h1 = function(A_1,W1,ttnsr,omega){
95   k = length(omega)
96   thet =W1%%c(A_1)
97   p = matrix(nrow = length(thet),ncol = k)
98   for (i in 1:k) {
99     p[,i] = logistic(thet + omega[i])
100   }
101   p = cbind(p,rep(1,length(thet)))-cbind(rep(0,length(thet)),p)
102   l = lapply(1:(k+1),function(i) -log(p[which(c(ttnsr)==i),i]))
103   return(sum(unlist(l)))
104 }
105
106 ##### gradient #####
107 g1 = function(A_1,W1,ttnsr,omega){
108   k = length(omega)
109   thet =W1%%c(A_1)
110   p = matrix(nrow = length(thet),ncol = k)
111   for (i in 1:k) {
112     p[,i] = logistic(thet + omega[i])
113   }
114   q = matrix(nrow = length(thet),ncol = k+1)
115   q[,1] <- p[,1]-1
116   for (i in 2:k) {
117     q[,i] <- (p[,i]*(1-p[,i])-p[,i-1]*(1-p[,i-1]))/(p[,i-1]-p[,i])
118   }
119   q[,k+1] <- p[,k]
120   l <- lapply(1:(k+1),function(i) apply(rbind(W1[which(c(ttnsr)==i),])*q[
121     which(c(ttnsr)==i),i],2,sum))
122   return(Reduce("+", l))
123 }
124
125 g2 = function(core,W4,ttnsr,omega){
126   k = length(omega)
127   thet =W4%%c(core)
128   p = matrix(nrow = length(thet),ncol = k)

```

```

128   for (i in 1:k) {
129     p[,i] = logistic(thet + omega[i])
130   }
131   #derivative for C
132   q = matrix(nrow = length(thet), ncol = k+1)
133   q[,1] <- p[,1]-1
134   for (i in 2:k) {
135     q[,i] <- (p[,i]*(1-p[,i])-p[,i-1]*(1-p[,i-1]))/(p[,i-1]-p[,i])
136   }
137   q[,k+1] <- p[,k]
138
139   #deriavtive for omega
140   r = list()
141   r[[1]] <- cbind(p[,1]-1, p[,1]*(1-p[,1])/(p[,2]-p[,1]))
142   for (i in 2:(k-1)) {
143     r[[i]] <- cbind(-p[,i]*(1-p[,i])/(p[,i]-p[,i-1]), p[,i]*(1-p[,i])/(p[,
144       i+1]-p[,i]))
145   }
146   r[[k]] <- cbind(-p[,k]*(1-p[,k])/(p[,k]-p[,k-1]), p[,k])
147
148   l1 <- vector(length = k)
149   for (i in 1:k) {
150     l1[i] <- sum(r[[i]][which(c(ttnsr)==i),1])+sum(r[[i]][which(c(ttnsr)==
151       i+1),2])
152   }
153   l2 <- lapply(1:(k+1), function(i) apply(rbind(W4[which(c(ttnsr)==i),]) * q[
154     which(c(ttnsr)==i),i], 2, sum))
155   return(c(l1, Reduce("+", l2)))
156 }
157 ##### realization #####
158
159 realization = function(theta, omega){
160   k = length(omega)
161   theta=as.tensor(theta)
162   thet <- c(theta@data)
163   p = matrix(nrow = length(thet), ncol = k)

```

```

164   for (i in 1:k) {
165     p[,i] = logistic(thet + omega[i])
166   }
167   p = cbind(p,rep(1,length(thet)))-cbind(rep(0,length(thet)),p)
168   for (j in 1:length(thet)) {
169     thet[j] <- sample(1:(k+1),1,prob = p[j,])
170   }
171   as.tensor(array(thet,dim =theta@modes))
172   return(as.tensor(array(thet,dim =theta@modes)))
173 }
174
175
176
177 ##### update a factor matrix at one time while holding others fixed
178 #####
179 comb = function(A,W,ttnsr,k,omega,alph=TRUE){
180   nA = A
181   tnsr1 <- k_unfold(as.tensor(ttnsr),k)@data
182   if (alph==TRUE) {
183     l <- lapply(1:nrow(A),function(i){optim(A[i,],function(x) h1(x,W,tnsr1
184       [i,],omega),function(x) g1(x,W,tnsr1[i,],omega),method = "BFGS")$par})
185     nA <- matrix(unlist(l),nrow = nrow(A),byrow = T)
186   }else{
187     l <- lapply(1:nrow(A),function(i){constrOptim(A[i,],
188       function(x) h1(x,W,tnsr1[i,],omega),
189       ui = as.matrix(rbind(W,-
190         W)),ci = rep(-alph,2*nrow(W)),method = "BFGS")$par})
191     nA <- matrix(unlist(l),nrow = nrow(A),byrow = T)
192   }
193   return(nA)
194 }
195
196 ##### update core tensor #####
197 corecomb = function(C,W,ttnsr,omega,alph=TRUE){
198   Cvec <- c(C@data)
199   h <- function(x) h1(x,W,ttnsr,omega)
200   g <- function(x) g1(x,W,ttnsr,omega)

```

```

199 H <- function(x) Hessi(x,W,ttnsr,omega)
200 d <- nlmnb(Cvec,h,g,H)
201 ##d <- optim(Cvec,h,g,method="BFGS") ## seems BFGS is faster??
202 C <- new("Tensor",C@num_modes,C@modes,data =d$par)
203 return(C)
204 }
205 ### updating core tensor and omega
206 coreomgcomb = function(C,W,ttnsr,omega,alpha=TRUE){
207   omgc = list()
208   k = length(omega)
209   coeff <- c(omega,C@data)
210   u = cbind(-diag(k-1),matrix(0,nrow = k-1,ncol = length(coeff)-k+1))+
211     cbind(0,diag(k-1),matrix(0,nrow = k-1,ncol = length(coeff)-k))
212   h <- function(x) h1(x[-(1:k)],W,ttnsr,x[1:k])
213   g <- function(x) g2(x[-(1:k)],W,ttnsr,x[1:k])
214   d <- constrOptim(coeff,h,g,method="BFGS",ui = u, ci = rep(0,k-1)) ##
215     seems BFGS is faster??
216   omgc$C <- new("Tensor",C@num_modes,C@modes,data =d$par[-(1:k)])
217   omgc$omega = omega = d$par[1:k]
218   return(omgc)
219 }
220 ##### update core tensor #####
221 prevcorecomb = function(C,W,ttnsr,omega,alpha=TRUE){
222   Cvec <- c(C@data)
223   h <- function(x) h1(x,W,ttnsr,omega)
224   g <- function(x) g1(x,W,ttnsr,omega)
225   H <- function(x) Hessi(x,W,ttnsr,omega)
226
227   if (alpha==TRUE) {
228     d <- nlmnb(Cvec,h,g,H)
229     C <- new("Tensor",C@num_modes,C@modes,data =d$par)
230   }else{
231     d <- constrOptim(Cvec,h,g,ui = rbind(W,-W),ci = rep(-alpha,2*nrow(W)),
232       method = "BFGS")
233     C <- new("Tensor",C@num_modes,C@modes,data =d$par)
234   }
235   return(C)
236 }

```



```

236
237
238 #####ordinal tensor decomposition based on Tucker structure #####
239 fit_ordinal = function(ttnsr,C,A_1,A_2,A_3,omega=TRUE,alpha = TRUE){
240   alphbound <- alph+10^-4
241   result = list()
242   error<- 3
243   iter = 0
244   cost=NULL
245   omg = omega
246   if(sum(omg)==TRUE) omega <- polr(as.factor(c(ttnsr[ttnsr>0]))~offset(-c(
247     prevtheta[ttnsr>0])))$zeta
248   if (alpha == TRUE) {
249     while ((error > 10^-4)&(iter<50) ) {
250       iter = iter +1
251
252       prevtheta <- ttl(C,list(A_1,A_2,A_3),ms=1:3)@data
253       prev <- likelihood(ttnsr[ttnsr>0],prevtheta[ttnsr>0],omega)
254
255       if(sum(omg)==TRUE) {
256         # update C and omega together
257         W4 <- kronecker(kronecker(A_3,A_2),A_1)
258         coreomg <- coreomgcomb(C,W4,c(ttnsr),omega)
259         C <- coreomg$C
260         omega <- coreomg$omega
261       }else{
262         # update C
263         W4 <- kronecker(kronecker(A_3,A_2),A_1)
264         C <- corecomb(C,W4,c(ttnsr),omega)
265       }
266
267       #update A_1
268       W1 = kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data)
269       A_1 <- comb(A_1,W1,ttnsr,1,omega)
270       #orthogonalize A_1
271       qr_res=qr(A_1)
272       A_1=qr.Q(qr_res)
273       C=ttm(C,qr.R(qr_res),1)

```

```

274
275 # update A_2
276 W2 <- kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data)
277 A_2 <- comb(A_2,W2,ttnsr,2,omega)
278 #orthogonalize A_2
279 qr_res=qr(A_2)
280 A_2=qr.Q(qr_res)
281 C=ttm(C,qr.R(qr_res),2)
282
283 # update A_3
284 W3 <- kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data)
285 A_3 <- comb(A_3,W3,ttnsr,3,omega)
286 #orthogonalize A_3
287 qr_res=qr(A_3)
288 A_3=qr.Q(qr_res)
289 C=ttm(C,qr.R(qr_res),3)
290
291 theta <- ttl(C,list(A_1,A_2,A_3),ms=1:3)@data
292 new <- likelihood(ttnsr[ttnsr>0],theta[ttnsr>0],omega)
293 cost = c(cost,new)
294 (error <- abs((new-prev)/prev))
295 }
296 }else{
297 while ((error > 10^-4)&(iter<50) ) {
298   iter = iter +1
299
300   prevtheta <- ttl(C,list(A_1,A_2,A_3),ms=1:3)@data
301   prev <- likelihood(ttnsr[ttnsr>0],prevtheta[ttnsr>0],omega)
302
303   if(sum(omega)==TRUE) {
304     # update C and omega together
305     W4 <- kronecker(kronecker(A_3,A_2),A_1)
306     coreomg <- coreomgcomb(C,W4,c(ttnsr),omega)
307     C <- coreomg$C
308     omega <- coreomg$omega
309   }else{
310     # update C
311     W4 <- kronecker(kronecker(A_3,A_2),A_1)
312     C <- corecomb(C,W4,c(ttnsr),omega)

```

```

313     }
314
315     #update A_1
316     W1 =kronecker(A_3,A_2)%*%t(k_unfold(C,1)@data)
317     A_1 <- comb(A_1,W1,ttnsr,1,omega,alphbound)
318     if(max(abs(ttl(C,list(A_1,A_2,A_3),ms=1:3)@data))>=alph) break
319     #orthogonalize A_1
320     qr_res=qr(A_1)
321     A_1=qr.Q(qr_res)
322     C=ttm(C,qr.R(qr_res),1)
323
324
325     # update A_2
326     W2 <- kronecker(A_3,A_1)%*%t(k_unfold(C,2)@data)
327     A_2 <- comb(A_2,W2,ttnsr,2,omega,alphbound)
328     if(max(abs(ttl(C,list(A_1,A_2,A_3),ms=1:3)@data))>=alph) break
329     #orthogonalize A_2
330     qr_res=qr(A_2)
331     A_2=qr.Q(qr_res)
332     C=ttm(C,qr.R(qr_res),2)
333
334
335     # update A_3
336     W3 <- kronecker(A_2,A_1)%*%t(k_unfold(C,3)@data)
337     A_3 <- comb(A_3,W3,ttnsr,3,omega,alphbound)
338     if(max(abs(ttl(C,list(A_1,A_2,A_3),ms=1:3)@data))>=alph) break
339     #orthogonalize A_3
340     qr_res=qr(A_3)
341     A_3=qr.Q(qr_res)
342     C=ttm(C,qr.R(qr_res),3)
343
344     theta <- ttl(C,list(A_1,A_2,A_3),ms=1:3)@data
345     new <- likelihood(ttnsr[ttnsr>0],theta[ttnsr>0],omega)
346     cost = c(cost,new)
347     error <- abs((new-prev)/prev)
348     if(max(abs(ttl(C,list(A_1,A_2,A_3),ms=1:3)@data))>=alph) break
349 }
350 }
351

```

```

352 result$C <- C; result$A_1 <- A_1; result$A_2 <- A_2; result$A_3 <- A_3
353 result$iteration <- iter
354 result$cost = cost; result$omega=omega
355 return(result)
356 }
357
358
359 ##### ordinal tensor decomposition based on CP structure #####
360 fit_ordinal_cp=function(ttnsr,A_1,A_2,A_3,omega=TRUE,alph = TRUE){
361   alphbound <- alph+10^-4
362   result = list()
363   error<- 3
364   iter = 0
365   cost=NULL
366   if (alph == TRUE) {
367     while ((error > 10^-4)&(iter<50) ) {
368       iter = iter +1
369
370       prevtheta <- tensorize(A_1,A_2,A_3)
371       #update omega
372       if(sum(omega)==TRUE) omega <- polr(as.factor(c(ttnsr[ttnsr>0]))~
offset(-c(prevtheta[ttnsr>0])))$zeta
373       prev <- likelihood(ttnsr[ttnsr>0],prevtheta[ttnsr>0],omega)
374
375       #update A_1
376       W1 = KhatriRao(A_3,A_2)
377       A_1 <- comb(A_1,W1,ttnsr,1,omega)
378       A_1 = apply(A_1,2,function(x){x/norm(x,"2")})
379
380
381       # update A_2
382       W2 <- KhatriRao(A_3,A_1)
383       A_2 <- comb(A_2,W2,ttnsr,2,omega)
384       A_2 = apply(A_2,2,function(x){x/norm(x,"2")})
385
386       # update A_3
387       W3 <- KhatriRao(A_2,A_1)
388       A_3 <- comb(A_3,W3,ttnsr,3,omega)
389

```

```

390
391   theta <- tensorize(A_1,A_2,A_3)
392   new <- likelihood(ttnsr[ttnsr>0],theta[ttnsr>0],omega)
393   cost = c(cost,new)
394   (error <- abs((new-prev)/prev))
395 }
396 }else{
397   while ((error > 10^-4)&(iter<50) ) {
398     iter = iter +1
399
400     prevtheta <- tensorize(A_1,A_2,A_3)
401     #update omega
402     if(sum(omega)==TRUE) omega <- polr(as.factor(c(ttnsr[ttnsr>0]))~
offset(-c(prevtheta[ttnsr>0])))$zeta
403     prev <- likelihood(ttnsr[ttnsr>0],prevtheta[ttnsr>0],omega)
404
405     #update A_1
406     W1 = KhatriRao(A_3,A_2)
407     A_1 <- comb(A_1,W1,ttnsr,1,omega,alphbound)
408     if(max(abs(tensorize(A_1,A_2,A_3)))>=alph) break
409
410
411     # update A_2
412     W2 <- KhatriRao(A_3,A_1)
413     A_2 <- comb(A_2,W2,ttnsr,2,omega,alphbound)
414     if(max(abs(tensorize(A_1,A_2,A_3)))>=alph) break
415
416     # update A_3
417     W3 <- KhatriRao(A_2,A_1)
418     A_3 <- comb(A_3,W3,ttnsr,3,omega,alphbound)
419     if(max(abs(tensorize(A_1,A_2,A_3)))>=alph) break
420
421     pre=rescale(A_1,A_2,A_3)
422     A_1=pre$A_1
423     A_2=pre$A_2
424     A_3=pre$A_3
425
426
427     theta <- tensorize(A_1,A_2,A_3)

```

```

428     new <- likelihood(ttnsr[ttnsr>0],theta[ttnsr>0],omega)
429     cost = c(cost,new)
430     error <- abs((new-prev)/prev)
431     if(max(abs(tensorize(A_1,A_2,A_3)))>=alph) break
432   }
433 }
434
435 result$A_1 <- A_1; result$A_2 <- A_2; result$A_3 <- A_3
436 result$iteration <- iter
437 result$cost = cost; result$omega=omega
438 return(result)
439 }
440
441
442 ##### construct tensor from CP factors
443 tensorize=function(A_1,A_2,A_3){
444   r=ncol(A_1)
445   tensor=0
446   for(i in 1:r){
447     tensor=tensor+A_1[,i]%o%A_2[,i]%o%A_3[,i]
448   }
449   return(tensor)
450 }
451
452
453 ## BIC: Inputs d and r are vectors.
454 bic = function(ttnsr,theta,omega,d,r){
455   return(2*likelihood(ttnsr,theta,omega)+(prod(r)+sum(r*(d-r)))*log(prod(d
456     )))
457 }
458
459
460 rescale=function(A_1,A_2,A_3){
461   r=ncol(A_1)
462   for(i in 1:r){
463     A_3[,i]=A_3[,i]*sqrt(sum(A_1[,i]^2))*sqrt(sum(A_2[,i]^2))
464     A_2[,i]=A_2[,i]/sqrt(sum(A_2[,i]^2))
465     A_1[,i]=A_1[,i]/sqrt(sum(A_1[,i]^2))

```

```
466 }  
467 return(list("A_1"=A_1, "A_2"=A_2, "A_3"=A_3))  
468 }
```

References

- [1] Peter McCullagh (1980). *Regression Models for Ordinal Data* Journal of the Royal Statistical Society. Series B (Methodological) Vol. 42, No. 2 , pp. 109-142 (34 pages)
- [2] BURRIDGE, J. (1980). *A note on maximum likelihood estimation for regression models using grouped data.* J. R. Statist. Soc. B, 42, in press.