**TUTORIAL PARA APLICAÇÃO DO MÉTODO GRADE OF MEMBERSHIP (GoM) NO R**

**Elaboração:** Adriel Felipe de Araújo Bezerra

**Orientação:** Prof. Dr. Moisés Alberto Calle Aguirre

#GoMRcpp.R é um arquivo R-Script desenvolvido por Julimar Pinto e André Caetano no Programa de Pós-Graduação em Ciências Sociais da Pontifícia Universidade Católica de Minas Gerais no ano de 2013.

1. **Ajuste do banco de dados**

Depois das variáveis selecionadas e tratadas, todas devem ter suas categorias recodificadas para cumprir a requisição de estar em formato sequencial a partir do número 1. Por exemplo, se a variável em questão for sexo não deve ter “1” para homens, “3” para mulheres e “9” para dados ausentes (*missings*) ou não resposta. Neste caso, as categorias devem ser “1” para homens, “2” para mulheres e “3” para não resposta. Ou apenas “1” para homens e “2” para mulheres, já que o GoM não requer que os *missings* sejam excluídos das variáveis. Contudo, pode ser criada uma categoria de não resposta em determinados casos.

Para esclarecer melhor, tomemos como exemplo a variável de raça. Supondo que ela esteja codificada como “0” para brancos, “1” para pretos, “2” para pardos, “3” para amarelos, “4” para indígenas, “5” para quem optou por não declarar e contenha dados faltantes (*missings*). O que necessitará ser ajustado é somente o primeiro número das categorias para que ela comece a partir do 1 e as demais devem crescer em ordem sequencial (1, 2, 3, 4, 5). Entretanto, recomenda-se que as categorias com representação muito baixa em comparação com as demais categorias sejam redistribuídas (recodificadas). Claro, respeitando os recortes da pesquisa. Nesse sentido, poderia ser “1” para brancos e “2” para não brancos uma alternativa – sem ser necessário filtrar os *missings*.

A única variável que não precisa estar em ordem sequencial é a de identificação, chamada por Guedes et al. (2016) de “idv” e no GoMRcpp de “SubjID”. Normalmente essa será a variável presente no banco de dados responsável por identificar os casos da unidade de análise.

De acordo com Guedes et al. (2016), o GoM **não** aceita variáveis incluídas no banco de dados que: 1) sejam contínuas; 2) tenham categorias não ordenadas em sequência; 3) sejam textuais.

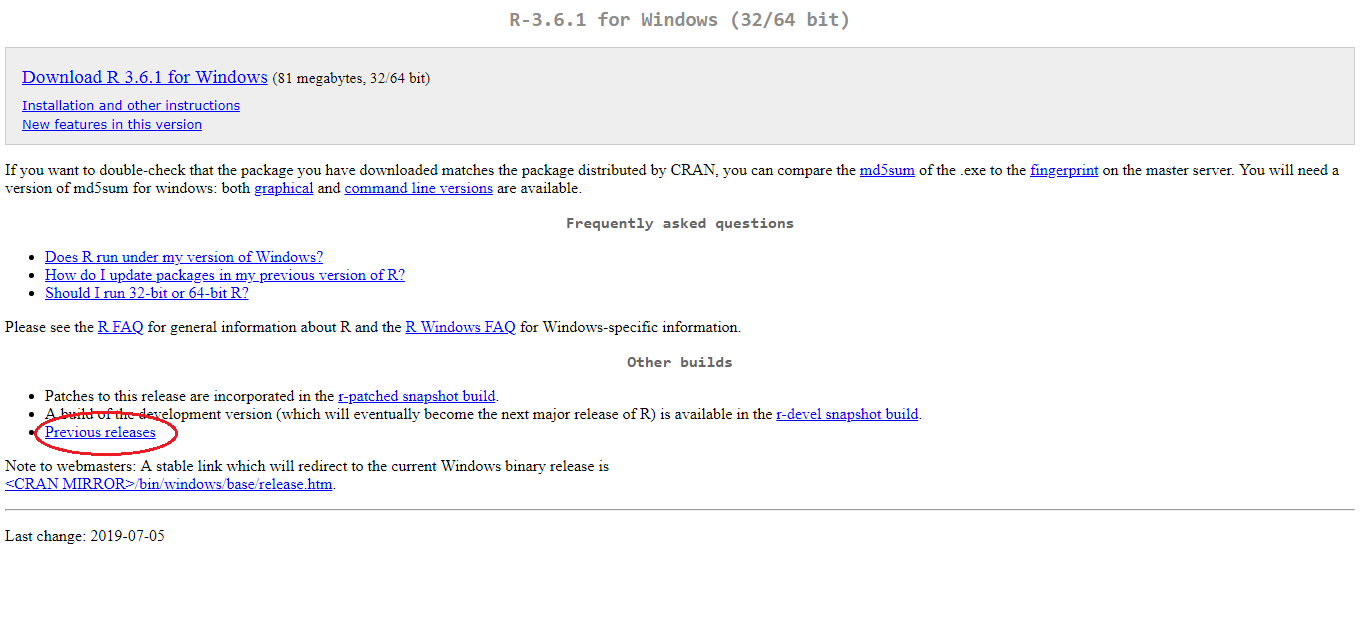
Todas as variáveis devem estar como “Nominal” (no caso do SPSS) ou *as.factor* (no caso do R). A variável de identificação deve ser nomeada como “SubjID”. Para quem não tem afinidade com R ou com programação, recomenda-se utilizar o SPSS para o processo de organização do banco de dados.

Após a preparação do banco de dados, ele deve ser salvo como (“salvar como...”) .csv – o que vai facilitar a leitura no R.

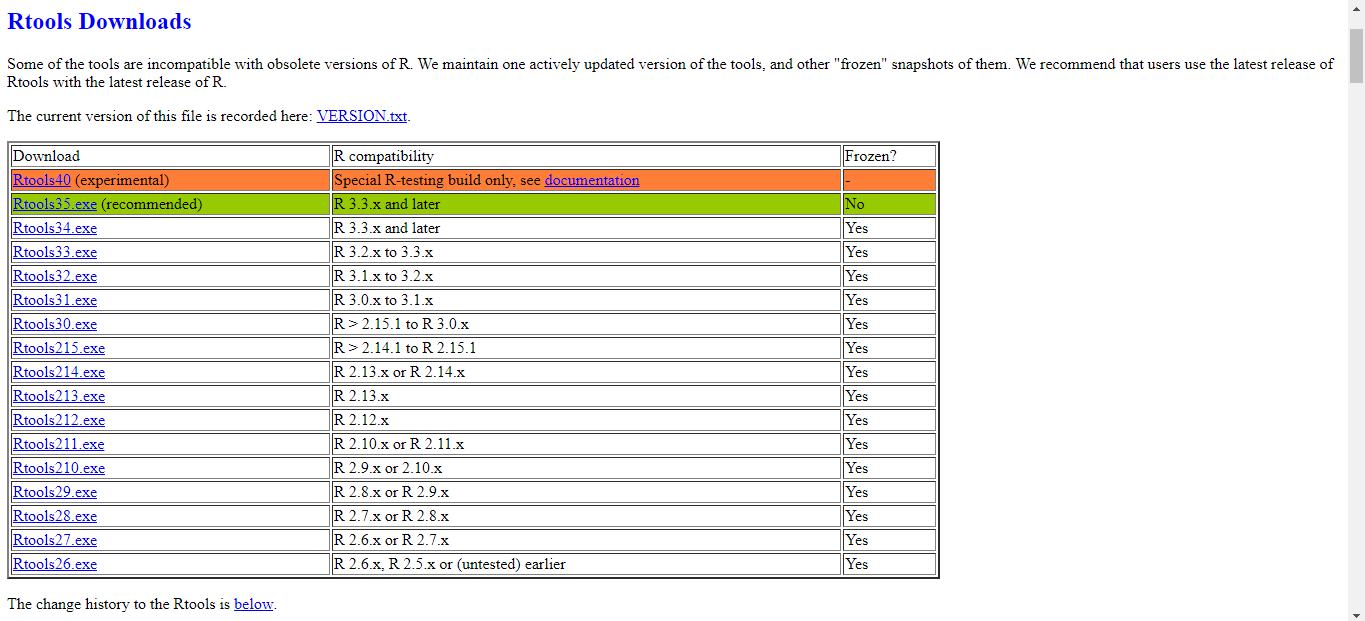
P.S.: No caso do banco ser preparado no SPSS, o tipo das variáveis deve estar como “Numérico” e a coluna Medir na aba de visualização da variável que deve estar como “Nominal”.

1. **Instalação do R, RTools e RStudio**

Para baixar e instalar o R basta acessar a página <<https://cran.r-project.org/bin/windows/base/>>. É importante notar que as versões mais recentes do R podem ainda estar em fase de testes. Caso haja alguma dificuldade com a versão mais recente, é possível recorrer a uma versão anterior como mostrado abaixo:

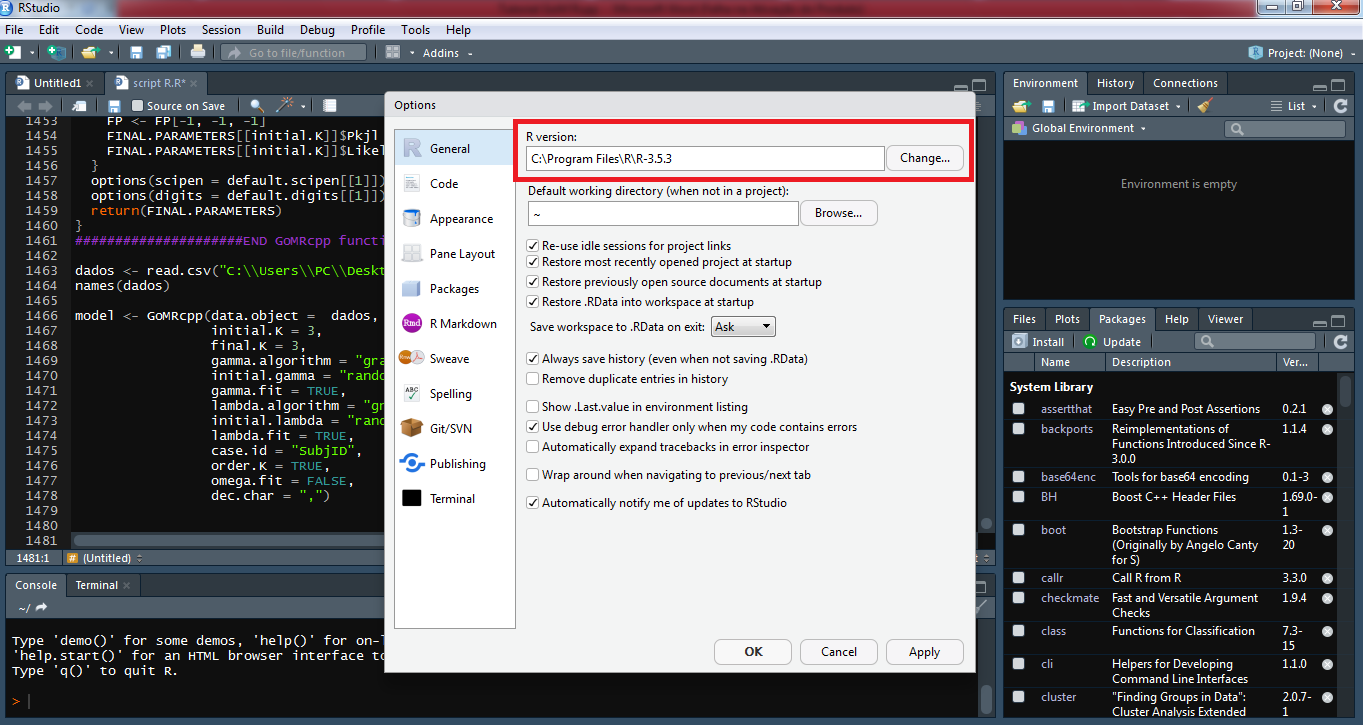


O mais importante é que a versão do R seja compatível com a versão do Rtools, que não deve estar “congelada” (FROZEN? = NO). O Rtools pode ser baixado e instalado na seguinte página: <<https://cran.r-project.org/bin/windows/Rtools/>>. Na página de download do Rtools é mostrado com quais versões do R há compatibilidade e qual versão do Rtools não está congelada.



O RStudio pode ser baixado gratuitamente para desktop na página <<https://www.rstudio.com/products/rstudio/download/>>. Como a interface do RStudio é mais amigável do que o console do R, ajudará em todo o procedimento.

Se já se tem o R instalado, é possível verificar qual a versão que está sendo utilizada no RStudio em “Tools” > “Global Options...” bem como é possível alterá-la no botão “Change...”.



Para a aplicação do método GoM em julho de 2019, foi utilizada a versão do R 3.5.3, o Rtools 35 e o RStudio Desktop 1.2.1335.

1. **Instalando e chamando os *packages* (pacotes)**

Para instalar os pacotes “Rcpp”, “inline” (que faz parte das dependências do pacote “Rcpp”), “dplyr” e “rstan” foi utilizado o seguinte código:

install.packages("Rcpp", dependencies = TRUE)

install.packages("dplyr")

install.packages("rstan")

Para os chamar foi utilizado o código:

require(Rcpp)

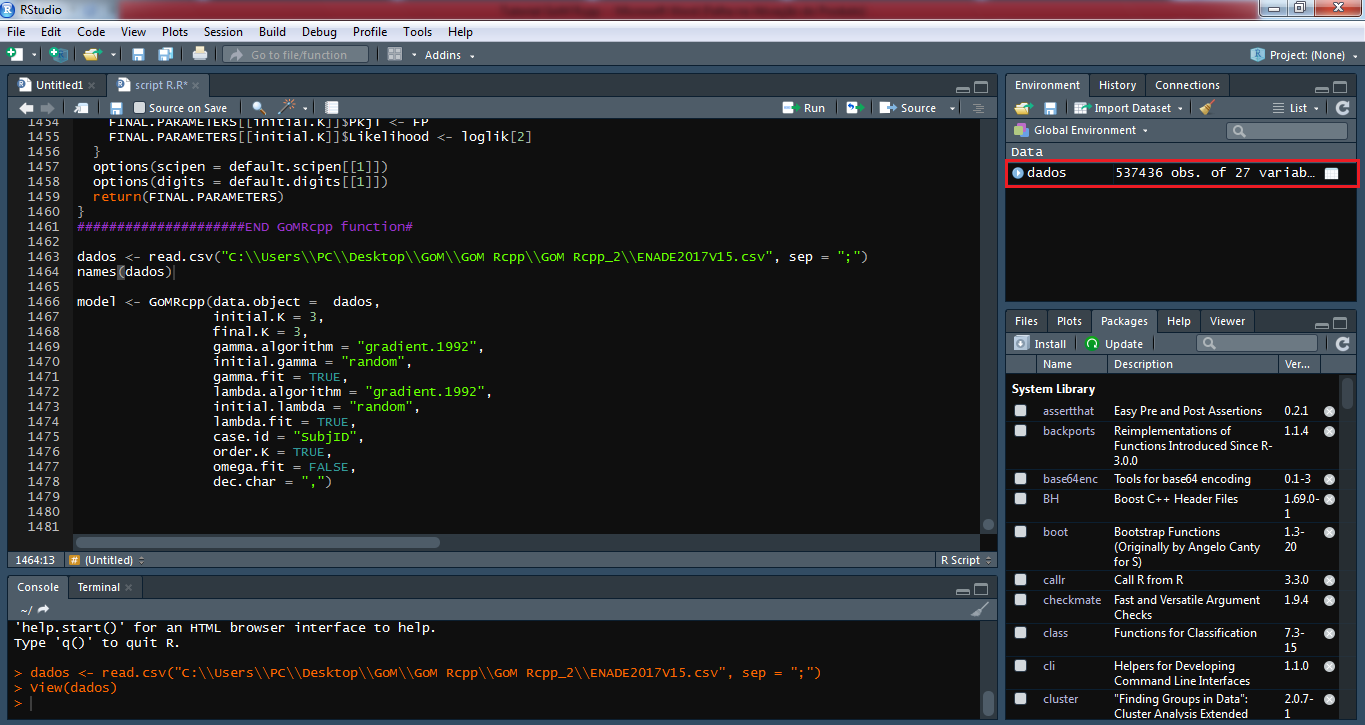
require(inline)

require(dplyr)

require(rstan)

1. **Importação de arquivo .csv**

Para importar um arquivo .csv basta usar o comando “read.csv()”. Dentro do parênteses é necessário estar o caminho totalmente qualificado do arquivo entre aspas simples ou duplas e com o sinal de barra (“/”) que indica cada pasta no sistema operacional Windows trocado por duas barras contrabarras (“\\”). Após informado isso, ainda dentro do parênteses, se separa com uma vírgula (“,”) o próximo comando, que indicará qual o separador do arquivo importado. Neste caso de um arquivo .csv, seu separador é ponto-e-vírgula (“;”). Após isso, pode-se verificar se todas as variáveis foram importadas corretamente através do comando “view()” e “names()” ou no ambiente (“Environment”) do RStudio, clicando em cima do nome do banco de dados importado.



Para carregar o banco de dados, foi utilizado o seguinte código:

dados <- read.csv("C:\\Users\\PC\\Desktop\\GoM\\GoM Rcpp\\GoM Rcpp\_2\\ENADE2017V15.csv", sep = ";")

names(dados)

1. **A função GoMRcpp**

Para ativar a função GoMRcpp.R existem duas formas. A primeira é rodar a função apertando CTRL+Enter (ou CTRL+R, dependendo da versão e configuração do RStudio) com o cursor de texto estando na primeira linha do código da função.

A segunda forma é baixar o arquivo “GoM 3.4 for R” no endereço <<https://gilvanguedes.com/publications/gombook/>>. Dentro do arquivo compactado de extensão .rar haverá instruções, exemplos e o arquivo da função, denominado “GoMRcpp.R”. Esse arquivo pode ser copiado para a pasta “library” dentro da pasta do R, que é onde ficam armazenadas as bibliotecas para aquela versão determinada do R. Se for copiado para a pasta *library*, ele pode ser invocado de forma similar à importação do banco de dados, entretanto o comando será “source()”. Dentro deste comando deve estar a localização totalmente qualificada do arquivo “GoMRcpp.R”. Como exemplo, o código pode ser da seguinte forma:

source("C:\\R-3.5.3\\library\\GoMRcpp.R")

Após o carregamento da função no ambiente do RStudio é possível fazer correr o modelo GoM. Para mais informações sobre a definição dos parâmetros, consultar Guedes et al. (2016) e/ou o repositório do Github de um dos desenvolvedores da função para o R, Julimar Santos <<https://github.com/julimarsp/GoMRcpp.R>>.

A título de exemplo, a modelagem para o GoM no R foi realizada por nós da seguinte forma:

model <- GoMRcpp(data.object = dados,

initial.K = 3,

final.K = 3,

gamma.algorithm = "gradient.1992",

initial.gamma = "random",

gamma.fit = TRUE,

lambda.algorithm = "gradient.1992",

initial.lambda = "random",

lambda.fit = TRUE,

case.id = "SubjID",

order.K = TRUE,

omega.fit = FALSE,

dec.char = ",")

1. **Resultados**

Após a execução de todos esses passos, dois arquivos com extensão .txt serão gerados. O arquivo “LogGoMKx(1)”, onde x é igual ao número de perfis K, mostra o *log* de toda a execução da função, iniciando com a forma como os parâmetros foram escolhidos e as variáveis internas ao modelo, depois uma tabela de frequência de cada variável, as probabilidades de tipo puro e os ajustes até a última tabela, que é utilizada para a análise e descrição dos tipos puros de K (“*Lambda-Marginal Frequency Ratio (LMFR)*”). Essa última tabela pode ser trabalhada no Excel para facilitar a análise.

Já o arquivo “GoMKx(1)”, onde x é igual ao número de perfis K, é um resultado que pode ser importado para o SPSS ou R e, além do banco de dados utilizado, retorna os valores finais (após o ajustamento iterativo) do grau de pertencimento gik de cada indivíduo. A partir deles podem ser construídos os tipos mistos.

1. **Fontes**

GUEDES, G. R.; SIVIERO, P. C. L.; MACHADO, C. J.; PINTO, J.; RODARTE, M. M. S. **Grade of Membership: conceitos básicos e aplicação empírica usando o programa GoM para Windows, Linux, Stata e R**. Belo Horizonte: Editora UFMG, 2016.

PINTO, J. **Passo a passo do** **GoMRcpp.R**. Disponível em: <<https://github.com/julimarsp/GoMRcpp.R/blob/master/01.01.01/Passo-a-Passo_GoMRcpp.R.pdf>>. Acesso em 23 de julho de 2019.

PINTO, Julimar Santos; CAETANO, André Junqueira. A Heterogeneidade da Vulnerabilidade Social das Juventudes: Uma Perspectiva Empírica Através do Método Grade of Membership. Revista Mediações, Londrina, v. 18, n. 1, 2013. ISSN: 1414-0543.

1. **Anexo – Código completo aplicado para executar o GoM no R**

##########################################################################################################  
## Title: GoMRcpp.R                                                                                     ##  
## Version: 01.01.01 - 'Jabuti Jabuticaba'                                                              ##  
## Date: 2014-11-10 (yyyy-mm-dd)                                                                        ##  
## Autor: Julimar Pinto, Andre Caetano                                                                  ##  
## Maintainer: Julimar Pinto <julimarsp dot jsp at gmail dot com>                                       ##  
## License: GPL-2 | GPL-3 [expanded from: GPL (â‰¥ 2)]                                                    ##  
## Description: GoMRcpp.R is a R-Script file for modeling Grade of Membership (GoM) to discrete data    ##  
## Depends: R (â‰¥ 2.15.2); \*Rtools (â‰¥ 2.16.0.1923); \*\*gcc (â‰¥ 4.7.3); Rcpp (â‰¥ 0.9.15); inline (â‰¥ 0.3.10)  ##  
## \*Only for Windows Operating System                                                                   ##  
## \*\*Only for Linux Operating System                                                                    ##  
##########################################################################################################  
  
install.packages("Rcpp", dependencies = TRUE)  
install.packages("dplyr")  
install.packages("rstan")  
  
require(Rcpp)  
require(inline)  
require(dplyr)  
require(rstan)  
  
source("C:\\Program Files\\R\\R-3.5.3\\library\\GoMRcpp.R")  
  
#####################BEGIN GoMRcpp function#  
GoMRcpp <- function (data.object = NULL,   
                     initial.K = 2, final.K = initial.K,   
                     gamma.algorithm = c("gradient.1992", "woodbury.1974"),   
                     initial.gamma = c("equal.values", "random", "pure1", "gamma.object"),   
                     initial.gamma.object = NULL,   
                     gamma.fit = TRUE,   
                     lambda.algorithm = c("gradient.1992", "woodbury.1974"),   
                     initial.lambda = c("random", "pure1", "equal.values", "lambda.object"),   
                     initial.lambda.object = NULL,   
                     lambda.fit = TRUE,   
                     [case.id](http://case.id/) = NA,   
                     case.weight = NA,   
                     internal.var = NULL,   
                     order.K = TRUE,   
                     omega.fit = FALSE,   
                     dec.char = ".") {  
    
  GoM <- '  
  using namespace std;  
  using namespace Rcpp;  
    
  const int MAXITER\_MODEL = 500;  
  const int MAXITER\_PARAMETERS = 25;  
    
  const double ZERO = 1.0E-20;  
  const double REALBIG = 1.0E+30;  
  const double CTOL = 1.0E-07;  
    
  IntegerVector   baselevel       (baselevel\_);  
  IntegerVector   ljlevels        (ljlevels\_);  
  NumericMatrix   cell            (cell\_);  
  CharacterVector gammaalgorithm  (gammaalgorithm\_);  
  NumericMatrix   G               (FG\_);  
  LogicalVector   gammafit        (gammafit\_);  
  CharacterVector lambdaalgorithm (lambdaalgorithm\_);  
  NumericVector   P               (FP\_);  
  LogicalVector   lambdafit       (lambdafit\_);  
  IntegerMatrix   FITP            (FITP\_);  
    
  int I = (cell.nrow() - 1);  
  int J = (cell.ncol() - 2);  
  int K = (G.ncol() - 1);  
    
  int i, k, k\_k, j, l, iter, subiter;  
  double ploglik, difflik;  
  double sumlik, p\_ijl, part, partlik, newpartlik, sumG, sumP, startlik, curlik;  
    
  double old\_G[(K + 1)];  
  double new\_G[(K + 1)];  
    
  NumericVector old\_P(clone(FP\_));  
  NumericVector new\_P(clone(FP\_));  
    
  vector<double> loglik(2);  
  char buffer[255];  
    
  // ## WOODBURY VARIABLES ####################  
    
  double numer, denom;  
    
  // ##########################################  
    
  // ## GRADIENT VARIABLES ####################  
    
  const int HALFSTEPS = MAXITER\_PARAMETERS;  
    
  const double ITOL = 0.0005;  
    
  const double MAXSTEP = 1.0;  
  const double MINSTEP = ZERO;  
    
  int l\_l, converged, halfstep, somefree;  
    
  double f0, f1, norm, g\_ij, bestlik, stepsize;  
    
  int    freeG[(K + 1)];  
  double dL\_dG[(K + 1)];  
    
  NumericVector dL\_dP(clone(FP\_));  
  NumericVector freeP(clone(FP\_));  
    
  // ##########################################  
    
  loglik[0] = 0.0;  
  ploglik = loglik[0];  
  // ## BEGIN loglikelihood FUNCTION ## //  
  sumlik = 0.0;  
  for (i = 1; i <= I; i++) {  
  partlik = 0.0;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel(j);  
  p\_ijl = 0.0;  
  for (k = 1; k <= K; k++) {  
  p\_ijl += (G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)));  
  }  
  if (p\_ijl < ZERO) {  
  p\_ijl = ZERO;  
  }  
  if (p\_ijl < REALBIG) {  
  partlik += log(p\_ijl);  
  }  
  }  
  sumlik += (cell(i, (J + 1)) \* partlik);  
  }  
  // ## END loglikelihood FUNCTION ## //  
  loglik[0] = sumlik;  
  sprintf(buffer, "%-10.4f", loglik[0]);  
  Rcout << "Primal Log-Likelihood is:\t" << buffer << endl << endl;  
  loglik[1] = loglik[0];  
  for (iter = 0; iter < MAXITER\_MODEL; iter++) {  
  difflik = (loglik[1] - ploglik);  
  ploglik = loglik[1];  
  if (iter) {  
  if (fabs(difflik / loglik[1]) < CTOL) {  
  break;  
  }  
  }  
  if (gammafit[0] && gammaalgorithm[0] == "woodbury.1974") {  
  // ## BEGIN Fit\_G\_Woodbury\_1974 FUNCTION ## //  
  subiter = 0;  
  for (i = 1; i <= I; i++) {  
  for (k = 1; k <= K; k++) {  
  old\_G[k] = G(i, k);  
  }  
  // ## BEGIN partlikelihood FUNCTION ## //  
  part = 0.0;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel[j];  
  p\_ijl = 0.0;  
  for (k = 1; k <= K; k++) {  
  p\_ijl += (G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)));  
  }  
  if (p\_ijl < ZERO) {  
  p\_ijl = ZERO;  
  }  
  if (p\_ijl < REALBIG) {  
  part += log(p\_ijl);  
  }  
  }  
  // ## END partlikelihood FUNCTION ## //  
  partlik = part;  
  for (subiter = 0; subiter < MAXITER\_PARAMETERS; subiter++) {  
  for (k = 1; k <= K; k++) {  
  new\_G[k] = 0.0;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel[j];  
  p\_ijl = 0.0;  
  for (k\_k = 1; k\_k <= K; k\_k++) {  
  p\_ijl += (G(i, k\_k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k\_k)));  
  }  
  if (p\_ijl > ZERO) {  
  new\_G[k] += ((G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k))) / p\_ijl);  
  }  
  }  
  }  
  for (k = 1; k <= K; k++) {  
  G(i, k) = (new\_G[k] / J);  
  }  
  // ## BEGIN rescale\_G FUNCTION ## //  
  sumG = 0.0;  
  for (k = 1; k <= K; k++) {  
  if (G(i, k) < ZERO) {  
  G(i, k) = 0.0;  
  }  
  sumG += G(i, k);  
  }  
  if (sumG < ZERO) {  
  sumG = (double)K;  
  for (k = 1; k <= K; k++) {  
  G(i, k) = 1.0;  
  }  
  }  
  for (k = 1; k <= K; k++) {  
  G(i, k) = (G(i, k) / sumG);  
  }  
  // ## END rescale\_G FUNCTION ## //  
  // ## BEGIN partlikelihood FUNCTION ## //  
  part = 0.0;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel[j];  
  p\_ijl = 0.0;  
  for (k = 1; k <= K; k++) {  
  p\_ijl += (G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)));  
  }  
  if (p\_ijl < ZERO) {  
  p\_ijl = ZERO;  
  }  
  if (p\_ijl < REALBIG) {  
  part += log(p\_ijl);  
  }  
  }  
  // ## END partlikelihood FUNCTION ## //  
  newpartlik = part;  
  if (newpartlik < partlik) {  
  for (k = 1; k <= K; k++) {  
  G(i, k) = old\_G[k];  
  }  
  break;  
  } else {  
  if (partlik > ZERO) {  
  if ((fabs(newpartlik - partlik) / partlik) < CTOL) {  
  break;  
  }  
  }  
  }  
  }  
  }  
  sprintf(buffer, "%03d.%03d%s%015.5f%s%015.7f", iter, subiter, "\t", loglik[1], "\t", fabs((loglik[1] - ploglik) / ploglik));  
  Rcout << "Fit G (Woodbury 1974):\t" << buffer << endl << endl;  
  // ## END Fit\_G\_Woodbury\_1974 FUNCTION ## //  
  } else if (gammafit[0] && gammaalgorithm[0] == "gradient.1992") {  
  // ## BEGIN Fit\_G\_Gradient\_1992 FUNCTION ## //  
  subiter = 0;  
  startlik = loglik[1];  
  for (i = 1; i <= I; i++) {  
  converged = 0;  
  for (subiter = 0; converged == 0; subiter++) { //##//  
  // ## BEGIN cellgradient\_G FUNCTION ## //  
  norm = 0.0;  
  for (k = 1; k <= K; k++) {  
  dL\_dG[k] =- (double)J;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel(j);  
  p\_ijl = 0.0;  
  for (k\_k = 1; k\_k <= K; k\_k++) {  
  p\_ijl = p\_ijl + (G(i, k\_k) \* P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k\_k));  
  }  
  if (p\_ijl > ZERO) {  
  dL\_dG[k] += (P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) / p\_ijl);  
  }  
  }  
  if ((G(i, k) <= ZERO && dL\_dG[k] < 0.0) || ((1.0 - G(i, k)) <= ZERO && dL\_dG[k] > 0.0)) {  
  freeG[k] = 0;  
  } else {  
  freeG[k] = 1;  
  norm += (dL\_dG[k] \* dL\_dG[k]);  
  }  
  }  
  // ## END cellgradient\_G FUNCTION ## //  
  if (norm <= ZERO) {  
  converged = 1;  
  break;  
  }  
  somefree = 0;  
  for (k = 1; k <= K; k++) {  
  old\_G[k] = G(i, k);  
  new\_G[k] = G(i, k);  
  if (freeG[k]) {  
  somefree++;  
  }  
  }  
  if (!somefree) {  
  break;  
  }  
  // ## BEGIN partlikelihood FUNCTION ## //  
  part = 0.0;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel[j];  
  p\_ijl = 0.0;  
  for (k = 1; k <= K; k++) {  
  p\_ijl += (G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)));  
  }  
  if (p\_ijl < ZERO) {  
  p\_ijl = ZERO;  
  }  
  if (p\_ijl < REALBIG) {  
  part += log(p\_ijl);  
  }  
  }  
  // ## END partlikelihood FUNCTION ## //  
  f0 = part;  
  stepsize = MAXSTEP;  
  bestlik = f0;  
  for (halfstep = 0; halfstep < HALFSTEPS && stepsize > MINSTEP; halfstep++) {  
  for (k = 1;k <= K; k++) {  
  G(i, k) = old\_G[k] + stepsize \* dL\_dG[k];  
  }  
  // ## BEGIN rescale\_G FUNCTION ## //  
  sumG = 0.0;  
  for (k = 1; k <= K; k++) {  
  if (G(i, k) < ZERO) {  
  G(i, k) = 0.0;  
  }  
  sumG += G(i, k);  
  }  
  if (sumG < ZERO) {  
  sumG = (double)K;  
  for (k = 1; k <= K; k++) {  
  G(i, k) = 1.0;  
  }  
  }  
  for (k = 1; k <= K; k++) {  
  G(i, k) = (G(i, k) / sumG);  
  }  
  // ## END rescale\_G FUNCTION ## //  
  // ## BEGIN partlikelihood FUNCTION ## //  
  part = 0.0;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel[j];  
  p\_ijl = 0.0;  
  for (k = 1; k <= K; k++) {  
  p\_ijl += (G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)));  
  }  
  if (p\_ijl < ZERO) {  
  p\_ijl = ZERO;  
  }  
  if (p\_ijl < REALBIG) {  
  part += log(p\_ijl);  
  }  
  }  
  // ## END partlikelihood FUNCTION ## //  
  f1 = part;  
  if (f1 > bestlik) {  
  bestlik = f1;  
  for (k = 1; k <= K; k++) {  
  new\_G[k] = G(i, k);  
  }  
  } else {  
  if (bestlik > (f0 + ZERO) && bestlik > (f1 + ZERO)) {  
  break;  
  }  
  }  
  stepsize = (stepsize / 2.0);  
  }  
  f1 = bestlik;  
  for (k = 1;k <= K; k++) {  
  G(i, k) = new\_G[k];  
  }  
  if (fabs(f0) > ZERO && fabs((f1 - f0) / f0) < CTOL) {  
  converged = 1;  
  }  
  if (fabs(f0) > ZERO && fabs((f1-f0)/f0) < ITOL && subiter > MAXITER\_PARAMETERS) {  
  break;  
  }  
  }  
  }  
  sprintf(buffer, "%03d.%03d%s%015.5f%s%015.7f", iter, subiter, "\t", loglik[1], "\t", fabs((loglik[1] - ploglik) / ploglik));  
  Rcout << "Fit G (Gradient 1992):\t" << buffer << endl << endl;  
  // ## END Fit\_G\_Gradient\_1992 FUNCTION ## //  
  }  
  // ## BEGIN loglikelihood FUNCTION ## //  
  sumlik = 0.0;  
  for (i = 1; i <= I; i++) {  
  partlik = 0.0;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel(j);  
  p\_ijl = 0.0;  
  for (k = 1; k <= K; k++) {  
  p\_ijl += (G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)));  
  }  
  if (p\_ijl < ZERO) {  
  p\_ijl = ZERO;  
  }  
  if (p\_ijl < REALBIG) {  
  partlik += log(p\_ijl);  
  }  
  }  
  sumlik += (cell(i, (J + 1)) \* partlik);  
  }  
  // ## END loglikelihood FUNCTION ## //  
  loglik[1] = sumlik;  
  if (lambdafit[0] && lambdaalgorithm[0] == "woodbury.1974") {  
  // ## BEGIN Fit\_P\_Woodbury\_1974 FUNCTION ## //  
  startlik = loglik[1];  
  for (subiter = 0; subiter < MAXITER\_PARAMETERS; subiter++) {  
  for (k = 1; k <= K; k++) {  
  for (j = 1; j <= J; j++) {  
  for (l = 1; l <= ljlevels(j); l++) {  
  old\_P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) = P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k));  
  }  
  }  
  }  
  for (k = 1; k <= K; k++) {  
  for (j = 1; j <= J; j++) {  
  if (FITP(k, j)) {  
  for (l = 1; l <= ljlevels(j); l++) {  
  numer = 0.0;  
  denom = 0.0;  
  for (i = 1; i <= I; i++) {  
  p\_ijl = 0.0;  
  for (k\_k = 1; k\_k <= K; k\_k++) {  
  p\_ijl += (G(i, k\_k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k\_k)));  
  }  
  if (p\_ijl > ZERO) {  
  g\_ij = ((G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k))) / p\_ijl);  
  } else {  
  g\_ij = 0.0;  
  }  
  denom += g\_ij;  
  if (l == (cell(i, j) + 1 - baselevel(j))) {  
  numer += g\_ij;  
  }  
  }  
  if (denom > ZERO) {  
  new\_P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) = (numer / denom);  
  } else {  
  new\_P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) = 0.0;  
  }  
  }  
  }  
  }  
  }  
  for (k = 1; k <= K; k++) {  
  for (j = 1; j <= J; j++) {  
  if (FITP(k, j)) {  
  for (l = 1; l <= ljlevels(j); l++) {  
  P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) = new\_P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k));  
  }  
  // ## BEGIN rescale\_P FUNCTION ## //  
  sumP = 0.0;  
  for (l = 1; l <= ljlevels(j); l++) {  
  if (P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) < ZERO) {  
  P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) = 0.0;  
  }  
  sumP += P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k));  
  }  
  if (sumP < ZERO) {  
  sumP = (double)ljlevels(j);  
  for (l = 1; l <= ljlevels(j); l++) {  
  P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) = 1.0;  
  }  
  }  
  for (l = 1; l <= ljlevels(j); l++) {  
  P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) = (P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) / sumP);  
  }  
  // ## END rescale\_P FUNCTION ## //  
  }  
  }  
  }  
  // ## BEGIN loglikelihood FUNCTION ## //  
  sumlik = 0.0;  
  for (i = 1; i <= I; i++) {  
  partlik = 0.0;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel(j);  
  p\_ijl = 0.0;  
  for (k = 1; k <= K; k++) {  
  p\_ijl += (G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)));  
  }  
  if (p\_ijl < ZERO) {  
  p\_ijl = ZERO;  
  }  
  if (p\_ijl < REALBIG) {  
  partlik += log(p\_ijl);  
  }  
  }  
  sumlik += (cell(i, (J + 1)) \* partlik);  
  }  
  // ## END loglikelihood FUNCTION ## //  
  curlik = sumlik;  
  sprintf(buffer, "%03d.%03d%s%015.5f%s%015.7f", iter, subiter, "\t", startlik, "\t", fabs((startlik - ploglik) / ploglik));  
  Rcout << "Fit P (Woodbury 1974):\t" << buffer << endl;  
  if (curlik < startlik) {  
  for (k = 1; k <= K; k++) {  
  for (j = 1; j <= J; j++) {  
  for (l = 1; l <= ljlevels(j); l++) {  
  P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) = old\_P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k));  
  }  
  }  
  }  
  loglik[1] = startlik;  
  break;  
  } else {  
  if ((startlik > ZERO) && (fabs(curlik - startlik) / startlik) < CTOL) {  
  break;  
  }  
  }  
  startlik = curlik;  
  }  
  Rcout << endl;  
  // ## END Fit\_P\_Woodbury\_1974 FUNCTION ## //  
  } else if (lambdafit[0] && lambdaalgorithm[0] == "gradient.1992") {  
  // ## BEGIN Fit\_P\_Gradient\_1992 FUNCTION ## //  
  converged = 0;  
  for (subiter = 0; converged == 0; subiter++) { //##//  
  // ## BEGIN gradient\_P FUNCTION ## //  
  norm = 0.0;  
  for (k = 1; k <= K; k++) {  
  for (j = 1; j <= J; j++) {  
  for (l = 1; l <= ljlevels(j); l++) {  
  dL\_dP(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) = 0.0;  
  for (i = 1; i <= I; i++) {  
  l\_l = cell(i, j) + 1 - baselevel(j);  
  p\_ijl = 0.0;  
  for (k\_k = 1; k\_k <= K; k\_k++) {  
  p\_ijl += (G(i, k\_k) \* P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k\_k));  
  }  
  if (l == l\_l) {  
  if (p\_ijl > ZERO) {  
  dL\_dP(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) += (cell(i, (J + 1)) \* G(i, k) \* ((1.0 / p\_ijl) - 1.0));  
  }  
  } else {  
  dL\_dP(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) -= (cell(i, (J + 1)) \* G(i, k));  
  }  
  }  
  if (((P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) <= ZERO) && (dL\_dP(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) < 0.0)) || (((1.0 - P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) <= ZERO) && (dL\_dP(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) > 0.0))) {  
  freeP(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) = 0;  
  } else {  
  freeP(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) = 1;  
  norm += dL\_dP(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) \* dL\_dP(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k);  
  }  
  }  
  }  
  }  
  // ## END gradient\_P FUNCTION ## //  
  if (norm <= ZERO) {  
  converged = 1;  
  break;  
  }  
  somefree = 0;  
  for (k = 1; k <= K; k++) {  
  for (j = 1; j <= J; j++) {  
  for (l = 1; l <= ljlevels(j); l++) {  
  old\_P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) = P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k);  
  new\_P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) = P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k);  
  if (freeP(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) {  
  somefree++;  
  }  
  }  
  }  
  }  
  if (!somefree) {  
  break;  
  }  
  stepsize = MAXSTEP;  
  // ## BEGIN loglikelihood FUNCTION ## //  
  sumlik = 0.0;  
  for (i = 1; i <= I; i++) {  
  partlik = 0.0;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel(j);  
  p\_ijl = 0.0;  
  for (k = 1; k <= K; k++) {  
  p\_ijl += (G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)));  
  }  
  if (p\_ijl < ZERO) {  
  p\_ijl = ZERO;  
  }  
  if (p\_ijl < REALBIG) {  
  partlik += log(p\_ijl);  
  }  
  }  
  sumlik += (cell(i, (J + 1)) \* partlik);  
  }  
  // ## END loglikelihood FUNCTION ## //  
  f0 = sumlik;  
  bestlik = f0;  
  for (halfstep = 0; halfstep < HALFSTEPS && converged == 0; halfstep++) {  
  for (k = 1; k <= K; k++) {  
  for (j = 1; j <= J; j++) {  
  if (FITP(k, j)) {  
  for (l = 1; l <= ljlevels(j); l++) {  
  P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) = old\_P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) + stepsize \* dL\_dP(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k);  
  }  
  // ## BEGIN rescale\_P FUNCTION ## //  
  sumP = 0.0;  
  for (l = 1; l <= ljlevels(j); l++) {  
  if (P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) < ZERO) {  
  P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) = 0.0;  
  }  
  sumP += P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k));  
  }  
  if (sumP < ZERO) {  
  sumP = (double)ljlevels(j);  
  for (l = 1; l <= ljlevels(j); l++) {  
  P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) = 1.0;  
  }  
  }  
  for (l = 1; l <= ljlevels(j); l++) {  
  P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) = (P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)) / sumP);  
  }  
  // ## END rescale\_P FUNCTION ## //  
  }  
  }  
  }  
  // ## BEGIN loglikelihood FUNCTION ## //  
  sumlik = 0.0;  
  for (i = 1; i <= I; i++) {  
  partlik = 0.0;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel(j);  
  p\_ijl = 0.0;  
  for (k = 1; k <= K; k++) {  
  p\_ijl += (G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)));  
  }  
  if (p\_ijl < ZERO) {  
  p\_ijl = ZERO;  
  }  
  if (p\_ijl < REALBIG) {  
  partlik += log(p\_ijl);  
  }  
  }  
  sumlik += (cell(i, (J + 1)) \* partlik);  
  }  
  // ## END loglikelihood FUNCTION ## //  
  f1 = sumlik;  
  if (f1 > bestlik) {  
  bestlik = f1;  
  for (k = 1; k <= K; k++) {  
  for (j = 1; j <= J; j++) {  
  for (l = 1; l <= ljlevels(j); l++) {  
  new\_P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) = P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k);  
  }  
  }  
  }  
  } else {  
  if ((bestlik > (f0 + ZERO)) && (bestlik > (f1 + ZERO))) {  
  break;  
  }  
  }  
  stepsize = (stepsize / 2.0);  
  }  
  f1 = bestlik;  
  loglik[1] = bestlik;  
  for (k = 1; k <= K; k++) {  
  for (j = 1; j <= J; j++) {  
  for (l = 1; l <= ljlevels(j); l++) {  
  P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k) = new\_P(((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k);  
  }  
  }  
  }  
  sprintf(buffer, "%03d.%03d%s%015.5f%s%015.7f", iter, subiter, "\t", loglik[1], "\t", fabs((loglik[1] - ploglik) / ploglik));  
  Rcout << "Fit P (Gradient 1992):\t" << buffer << endl;  
  if ((fabs(f0) > ZERO) && (fabs((f1 - f0) / f0) < CTOL)) {  
  converged = 1;  
  }  
  if ((fabs(f0) > ZERO) && (fabs((f1 - f0) / f0) < ITOL) && (subiter >= MAXITER\_PARAMETERS)) {  
  break;  
  }  
  }  
  Rcout << endl;  
  // ## END Fit\_P\_Gradient\_1992 FUNCTION ## //  
  }  
  }  
  // ## BEGIN loglikelihood FUNCTION ## //  
  sumlik = 0.0;  
  for (i = 1; i <= I; i++) {  
  partlik = 0.0;  
  for (j = 1; j <= J; j++) {  
  l = cell(i, j) + 1 - baselevel(j);  
  p\_ijl = 0.0;  
  for (k = 1; k <= K; k++) {  
  p\_ijl += (G(i, k) \* P((((l \* ((J + 1) \* (K + 1))) + ((K + 1) \* j)) + k)));  
  }  
  if (p\_ijl < ZERO) {  
  p\_ijl = ZERO;  
  }  
  if (p\_ijl < REALBIG) {  
  partlik += log(p\_ijl);  
  }  
  }  
  sumlik += (cell(i, (J + 1)) \* partlik);  
  }  
  // ## END loglikelihood FUNCTION ## //  
  loglik[1] = sumlik;  
  sprintf(buffer, "%-10.4f", loglik[1]);  
  Rcout << "Latter Log-Likelihood is:\t" << buffer << endl;  
  return(wrap(loglik));  
  '  
  if (require(Rcpp) && require(inline)) {  
    GoM\_Model <- cxxfunction (signature(  
      baselevel\_ = "numeric",   
      ljlevels\_ = "numeric",   
      cell\_ = "matrix",   
      gammaalgorithm\_ = "character",   
      FG\_ = "matrix",   
      gammafit\_ = "bool",   
      lambdaalgorithm\_ = "character",   
      FP\_ = "numeric",   
      lambdafit\_ = "bool",   
      FITP\_ = "matrix"),   
      body = GoM,   
      includes = "#include <cstdio>",   
      plugin = "Rcpp")  
  } else {  
    stop ("The Rcpp and inline packages are not installed ...")  
  }  
    
  #####################BEGIN verify.parameters function#  
  verify.parameters <- function ([case.id](http://case.id/),   
                                 case.weight,  
                                 data.object,  
                                 internal.var,  
                                 initial.gamma, initial.lambda,  
                                 gamma.algorithm, lambda.algorithm,   
                                 initial.gamma.object, initial.lambda.object) {  
    if (!gamma.algorithm %in% c("woodbury.1974", "gradient.1992")) {  
      stop("The gamma.algorithm information is wrong ...")  
    }  
    if (!(initial.gamma %in% c("random", "pure1", "equal.values", "gamma.object"))) {  
      stop("The initial.gamma information is wrong ...")  
    }  
    if (initial.gamma == c("gamma.object") & missing(initial.gamma.object)) {  
      stop("The initial GoM scores object is missing ...")  
    }  
    if (!(lambda.algorithm %in% c("woodbury.1974", "gradient.1992"))) {  
      stop("The lambda.algorithm information is wrong ...")  
    }  
    if (!(initial.lambda %in% c("random", "pure1", "equal.values", "lambda.object"))) {  
      stop("The initial.lambda information is wrong ...")  
    }  
    if (initial.lambda == c("lambda.object") & missing(initial.lambda.object)) {  
      stop("The initial pure type probabilities object is missing ...")  
    }  
    if ([is.na](http://is.na/)([case.id](http://case.id/)) | !([case.id](http://case.id/) %in% names(data.object))) {  
      stop("The [case.id](http://case.id/) is missing ...")  
    }  
    if (![is.na](http://is.na/)(case.weight) && !(case.weight %in% names(data.object))) {  
      stop("The case.weight is missing ...")  
    }  
    if (missing(internal.var)) {  
      stop("The internal.var is missing ...")  
    } else if (length(internal.var[(c(internal.var) %in% names(data.object) == FALSE)]) > 0) {  
      stop("The internal.var information is wrong ...")  
    }  
  }  
  #####################END verify.parameters function#  
    
  #####################BEGIN summary.parameters function#  
  summary.parameters <- function ([case.id](http://case.id/),  
                                  case.weight,  
                                  data.object,  
                                  internal.var,  
                                  initial.K, final.K,  
                                  initial.gamma, initial.lambda,  
                                  gamma.algorithm, lambda.algorithm,  
                                  gamma.fit, lambda.fit, order.K, omega.fit) {  
    cat(paste("\n\*GoMRcpp Summary\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n", sep = "", collapse = NULL))  
    cat(paste("Input data object---------------------: ", "From input data object", "\n", sep = "", collapse = NULL))  
    cat(paste("Number of profiles in initial model---: ", initial.K, "\n", sep = "", collapse = NULL))  
    cat(paste("Number of profiles in final model-----: ", final.K, "\n", sep = "", collapse = NULL))  
    if(gamma.fit == TRUE) {  
      cat(paste("GoM scores algorithm------------------: ", gamma.algorithm, "\n", sep = "", collapse = NULL))  
    } else {  
      cat(paste("GoM scores algorithm------------------: None\n", sep = "", collapse = NULL))  
    }  
    if (initial.gamma == "gamma.object") {  
      cat(paste("Initial GoM scores--------------------: From input gamma object\n", sep = "", collapse = NULL))  
    } else {  
      cat(paste("Initial GoM scores--------------------: ", initial.gamma, "\n", sep = "", collapse = NULL))  
    }  
    cat(paste("Fit GoM scores------------------------: ", gamma.fit, "\n", sep = "", collapse = NULL))  
    if (lambda.fit == TRUE) {  
      cat(paste("Pure type probabilities algorithm-----: ", lambda.algorithm, "\n", sep = "", collapse = NULL))  
    } else {  
      cat(paste("Pure type probabilities algorithm-----: None\n", sep = "", collapse = NULL))  
    }  
    if (initial.lambda == "lambda.object") {  
      cat(paste("Initial pure type probabilities-------: From input lambda object\n", sep = "", collapse = NULL))  
    } else {  
      cat(paste("Initial pure type probabilities-------: ", initial.lambda, "\n", sep = "", collapse = NULL))  
    }  
    cat(paste("Fit pure type probabilities-----------: ", lambda.fit, "\n", sep = "", collapse = NULL))  
    cat(paste("Sort profiles-------------------------: ", order.K, "\n", sep = "", collapse = NULL))  
    cat(paste("Records in data object----------------: ", nrow(data.object), "\n", sep = "", collapse = NULL))  
    if (omega.fit == TRUE) {  
      cat(paste("Unique data patterns------------------: All patterns", "\n", sep = "", collapse = NULL))  
    } else {  
      cat(paste("Unique data patterns------------------: From input data object", "\n", sep = "", collapse = NULL))  
    }  
    cat(paste("Case label----------------------------: ", [case.id](http://case.id/), "\n", sep = "", collapse = NULL))  
    if (![is.na](http://is.na/)(case.weight)) {  
      cat(paste("Case weight---------------------------: ", case.weight, "\n", sep = "", collapse = NULL))  
    } else {  
      cat(paste("Case weight---------------------------: None\n", sep = "", collapse = NULL))  
    }  
    cat("Internal variables--------------------:", c(internal.var), "\n")  
    if (length(names(data.object)[(names(data.object) %in% c(c([case.id](http://case.id/)), c(case.weight), c(internal.var)) == FALSE)] > 0)) {  
      external.var <- names(data.object)[(names(data.object) %in% c(c([case.id](http://case.id/)), c(case.weight), c(internal.var)) == FALSE)]  
    } else {  
      external.var <- c("------")  
    }  
    cat("External variables--------------------:", external.var, "\n")  
  }  
  #####################END summary.parameters function#  
    
  #####################BEGIN cell.data function#  
  cell.data <- function (data.object, case.weight, internal.var, omega.fit) {  
    cell <- data.frame(sapply(data.object[(names(data.object) %in% c(internal.var) == TRUE)], as.factor))  
    if (![is.na](http://is.na/)(case.weight)) {  
      cell[c(case.weight)] <- data.frame(sapply(data.object[c(case.weight)], as.numeric))  
    }  
    if (max(mapply(nlevels, cell[, c(internal.var)])) == 2) {  
      cat(paste("\n\*Note: All internal variables are dichotomous.\n", sep = "", collapse = NULL))  
    }  
    cell <- cell[do.call(order, cell[c(internal.var)]), ]  
    cell$Patterns <- do.call(paste, c(as.list(cell[c(internal.var)]), sep=""))  
    if (omega.fit == TRUE) {  
      cellomega <- as.data.frame(ftable(cell[c(internal.var)]))  
      cellomega$Freq <- as.numeric(cellomega$Freq + 1)  
      cellomega <- cellomega[do.call(order, cellomega[c(internal.var)]), ]  
      cellomega$Patterns <- do.call(paste, c(as.list(cellomega[c(internal.var)]), sep=""))  
      if ([is.na](http://is.na/)(case.weight)) {  
        cell <- cellomega[, c(c("Patterns"), c(internal.var), c("Freq"))]  
      } else if (![is.na](http://is.na/)(case.weight)) {  
        cellStringFreq <- aggregate(cell[c(case.weight)], list(cell$Patterns), FUN = sum, simplify = TRUE)  
        names(cellStringFreq) <- c("Patterns", "Freqomega")  
        cellomega <- merge(cellomega, cellStringFreq, by = c("Patterns"), all.x = TRUE)  
        cellomega$Freqomega <- as.numeric(cellomega$Freqomega + 1)  
        cellomega$Freqomega[[is.na](http://is.na/)(cellomega$Freqomega)] <- 1  
        cell <- cellomega[, c(c("Patterns"), c(internal.var), c("Freqomega"))]  
        names(cell) <- c("Patterns", c(internal.var), "Freq")  
      }  
      cat(paste("\n\*Note: ", nrow(cell), " unique data patterns has found for combinations of the all patterns.\n\n", sep = "", collapse = NULL))  
    }  
    if (omega.fit == FALSE) {  
      if ([is.na](http://is.na/)(case.weight)) {  
        cell$FreqCell <- sequence(rle(cell$Patterns)$lengths)  
        cellStringFreq <- aggregate(cell$FreqCell, list(cell$Patterns), FUN = max, simplify = TRUE)  
        names(cellStringFreq) <- c("Patterns", "Freq")  
      } else if (![is.na](http://is.na/)(case.weight)) {  
        cellStringFreq <- aggregate(cell[c(case.weight)], list(cell$Patterns), FUN = sum, simplify = TRUE)  
        names(cellStringFreq) <- c("Patterns", "Freq")  
      }  
      cell <- merge(cell, cellStringFreq, by = c("Patterns"))  
      cell <- cell[, c(c("Patterns"), c(internal.var), c("Freq"))]  
      cell <- unique(cell)  
      cat(paste("\n\*Note: ", nrow(cell), " unique data patterns has found in data object.\n\n", sep = "", collapse = NULL))  
    }  
    cell <- rbind(rep(NA, (length(c(internal.var)) + 2)), cell)  
    row.names(cell) <- NULL  
    return (cell)  
  }  
  #####################END cell.data function#  
    
  #####################BEGIN ljlevels.function function#  
  ljlevels.function <- function (cell, internal.var) {  
    ljlevels <- sapply(c(c(NA), c(cell[c(internal.var)])), nlevels)  
    return (ljlevels)  
  }  
  #####################END ljlevels.function function#  
    
  #####################BEGIN l.levels.j.function function#  
  l.levels.j.function <- function (cell, internal.var) {  
    l.levels.j <- (sapply(sapply(c(c(NA), c(cell[c(internal.var)])), levels), as.numeric))  
    for (i in 1:length(internal.var)) {  
      if (min(l.levels.j[[i+1]]) < 1) {  
        stop(paste0("\n\nThe data.object can only encompass codes values from 1 (one):\n\n",   
                    "\tTherefore, you must to avoid to use the 0 (zero) code ...\n\n\n"))  
      }  
    }  
    return (l.levels.j)  
  }  
  #####################END l.levels.j.function function#  
    
  #####################BEGIN parameter.FG function#  
  parameter.FG <- function (initial.K, initial.gamma, cell, initial.gamma.object) {  
    if (initial.gamma == c("random")) {  
      FG <- as.data.frame(matrix(NA, nrow(cell), initial.K, byrow = T))  
      for (i in 1:nrow(cell)) {  
        ki <- sample(c(1:initial.K), initial.K, replace = FALSE, prob = NULL)  
        random.gamma <- c(rep(as.double(0), initial.K))  
        for (k in 1:initial.K) {  
          if (k == 1) {  
            random.gamma[k] <- as.double(runif(1, min = as.double(0), max = as.double(1)))  
          } else {  
            random.gamma[k] <- as.double(runif(1, min = as.double(0), max = (as.double(1) - sum(random.gamma))))  
          }  
        }  
        for (k in 1:initial.K) {  
          if (k == 1) {  
            FG[i, ki[k]] <- random.gamma[k]  
          } else if (k != 1) {  
            if (sum(FG[i, 1:initial.K], random.gamma[k], na.rm = TRUE) <= as.double(1)) {  
              if (k != length(ki)) {  
                FG[i, ki[k]] <- random.gamma[k]  
              } else if (k == length(ki)) {  
                FG[i, ki[k]] <- (as.double(1) - (sum(FG[i, 1:initial.K], na.rm = TRUE)))  
              }  
            } else if (sum(FG[i, 1:initial.K], random.gamma[k], na.rm = TRUE) > as.double(1)) {  
              FG[i, ki[k]] <- (as.double(1) - (sum(FG[i, 1:initial.K], na.rm = TRUE)))  
            }  
          }  
        }  
      }  
    } else if (initial.gamma == c("pure1")) {  
      FG <- as.data.frame(matrix(c(as.double(1), rep(as.double(0), (initial.K - 1))), nrow(cell), initial.K, byrow = T))  
    } else if (initial.gamma == c("equal.values")) {  
      FG <- as.data.frame(matrix(c(as.double(1) / initial.K), nrow(cell), initial.K, byrow = T))  
    } else if (initial.gamma == c("gamma.object")) {  
      FG <- initial.gamma.object  
    }  
    if (initial.gamma != c("gamma.object")) {  
      FG <- cbind(rep(NA, nrow(cell)), FG)  
      FG[1, ] <- NA  
    } else {  
      FG <- cbind(rep(NA, (nrow(cell) - 1)), FG)  
      FG <- rbind(rep(NA, (initial.K + 1)), FG)  
    }  
    names(FG) <- c("Patterns", paste("k", 1:initial.K, sep = ""))  
    FG[, "Patterns"] <- NA  
    return (FG)  
  }  
  #####################END parameter.FG function#  
    
  #####################BEGIN fit.P function#  
  fit.P <- function (initial.K, initial.lambda, ljlevels, initial.lambda.object) {  
    FITP <- matrix(as.integer(1), (initial.K + 1), length(ljlevels), byrow = T)  
    FITP[1, ] <- NA  
    FITP[, 1] <- NA  
    if (initial.lambda == c("lambda.object")) {  
      for (k in 2:(initial.K + 1)) {  
        for (j in 2:length(ljlevels)) {  
          for (l in 2:(ljlevels[[j]] + 1)) {  
            if (initial.lambda.object[(k - 1), (j - 1), (l - 1)] < as.double(0)) {  
              FITP[k, j] <- as.integer(0)  
            }  
          }  
        }  
      }  
    }  
    return (FITP)  
  }  
  #####################BEGIN fit.P function#  
    
  #####################BEGIN parameter.FP function#  
  parameter.FP <- function (initial.K, initial.lambda, ljlevels, initial.lambda.object) {  
    FP <- array(c((as.double(1) / initial.K)), c((initial.K + 1), length(ljlevels), (max(ljlevels) + 1)), dimnames = list(c(paste("k", 0:initial.K, sep = "")), c(paste("j", 0:(length(ljlevels) - 1), sep = "")), c(paste("l", 0:max(ljlevels), sep = ""))))  
    FP[1, , ] <- NA  
    FP[, 1, ] <- NA  
    FP[, , 1] <- NA  
    if (initial.lambda == c("pure1")) {  
      for (j in 2:length(ljlevels)) {  
        FP[2, j, 2] <- as.double(1)  
        for (l in 3:(ljlevels[[j]] + 1)) {  
          FP[2, j, l] <- as.double(0)  
        }  
      }  
    }  
    if (initial.lambda == c("random")) {  
      for (k in 2:(initial.K + 1)) {  
        for (j in 2:length(ljlevels)) {  
          li <- sample(c(2:(ljlevels[[j]] + 1)), ljlevels[[j]], replace = FALSE, prob = NULL)  
          random.lambda <- c(rep(as.double(0), (ljlevels[[j]] + 1)))  
          for (l in 2:(ljlevels[[j]] + 1)) {  
            if (l == 2) {  
              random.lambda[l] <- as.double(runif(1, min = as.double(0), max = as.double(1)))  
            } else {  
              random.lambda[l] <- as.double(runif(1, min = as.double(0), max = (as.double(1) - sum(random.lambda))))  
            }  
          }  
          for (l in 2:(ljlevels[[j]] + 1)) {  
            if (l == 2) {  
              FP[k, j, 2:(ljlevels[[j]] + 1)] <- NA  
              FP[k, j, li[l - 1]] <- random.lambda[l]  
            } else if (l != 2) {  
              if (sum(FP[k, j, 2:(ljlevels[[j]] + 1)], random.lambda[l], na.rm = TRUE) <= as.double(1)) {  
                if ((l - 1) != length(li)) {  
                  FP[k, j, li[l - 1]] <- random.lambda[l]  
                } else if ((l - 1) == length(li)) {  
                  FP[k, j, li[l - 1]] <- as.double(1) - sum(FP[k, j, 2:(ljlevels[[j]] + 1)], na.rm = TRUE)  
                }  
              } else if (sum(FP[k, j, 2:(ljlevels[[j]] + 1)], random.lambda[l], na.rm = TRUE) > as.double(1)) {  
                FP[k, j, li[l - 1]] <- as.double(1) - sum(FP[k, j, 2:(ljlevels[[j]] + 1)], na.rm = TRUE)  
              }  
            }  
          }  
        }  
      }  
    }  
    if (initial.lambda == c("lambda.object")) {  
      for (k in 2:(initial.K + 1)) {  
        for (j in 2:length(ljlevels)) {  
          for (l in 2:(ljlevels[[j]] + 1)) {  
            if (initial.lambda.object[(k - 1), (j - 1), (l - 1)] < as.double(0)) {  
              FP[k, j, l] <- -(initial.lambda.object[(k - 1), (j - 1), (l - 1)])  
            } else {  
              FP[k, j, l] <- initial.lambda.object[(k - 1), (j - 1), (l - 1)]  
            }  
          }  
        }  
      }  
    }  
    for (k in 2:(initial.K + 1)) {  
      for (j in 2:length(ljlevels)) {  
        for (l in 2:(max(ljlevels) + 1)) {  
          if (l > (max(ljlevels[[j]]) + 1)) {  
            FP[k, j, l] <- NA  
          }  
        }  
      }  
    }  
    return (FP)  
  }  
  #####################END parameter.FP function#  
    
  #####################BEGIN v.order.K function#  
  v.order.K <- function (initial.K, cell, ljlevels, FP) {  
    N <- sum(cell$Freq, na.rm = TRUE)  
    Zc <- 2.58  
    for (l in 2:((max(ljlevels) + 1) - 1)) {  
      v.order <- c(rep(as.double(0), initial.K))  
      for (k in 2:(initial.K + 1)) {  
        Pc1 <- 0  
        Pc2 <- 0  
        for (j in 2:length(ljlevels)) {  
          if (l < (max(ljlevels[[j]]) + 1)) {  
            if ((as.double(sum(FP[k, j, 2:l])) != as.double(0)) == TRUE) {  
              if (v.order[k - 1] == as.double(0)) {  
                v.order[k - 1] <- as.double(sum(FP[k, j, 2:l]) ^ (1 / N))  
              } else {  
                v.order[k - 1] <- prod(v.order[k - 1], as.double(sum(FP[k, j, 2:l]) ^ (1 / N)))  
              }  
            } else {  
              Pc1 <- Pc1 + 1  
              Pc2 <- Pc2 + ((length(ljlevels) - 1) + 2) - j  
            }  
          } else {  
            l\_l <- ((max(ljlevels[[j]]) + 1) - 1)  
            if ((as.double(sum(FP[k, j, 2:l\_l])) != as.double(0)) == TRUE) {  
              if (v.order[k - 1] == as.double(0)) {  
                v.order[k - 1] <- as.double(sum(FP[k, j, 2:l\_l]) ^ (1 / N))  
              } else {  
                v.order[k - 1] <- prod(v.order[k - 1], as.double(sum(FP[k, j, 2:l\_l]) ^ (1 / N)))  
              }  
            } else {  
              Pc1 <- Pc1 + 1  
              Pc2 <- Pc2 + ((length(ljlevels) - 1) + 2) - j  
            }  
          }  
        }  
        v.order[k - 1] <- (v.order[k - 1] / (1 + ((Pc2 / sum((length(ljlevels) - 1):1)) \* Pc1)))  
      }  
      p.v.order <- matrix(NA, initial.K, initial.K)  
      for (k in 1:initial.K) {  
        for (k\_k in 1:initial.K) {  
          if (k != k\_k) {  
            p.v.order[k, k\_k] <- ((v.order[k] - v.order[k\_k]) / sqrt((((N \* v.order[k]) + (N \* v.order[k\_k])) / (N + N)) \* (1 - (((N \* v.order[k]) + (N \* v.order[k\_k])) / (N + N))) \* ((N + N) / (N \* N))))  
          }  
        }  
      }  
      if (min(abs(p.v.order), na.rm = TRUE) > Zc) {  
        break  
      }  
    }  
    if (min(abs(p.v.order), na.rm = TRUE) > Zc) {  
      v.order <- order(v.order, decreasing = TRUE)  
    } else {  
      v.order <- NULL  
    }  
    return (v.order)  
  }  
  #####################END v.order.K function#  
    
  #####################BEGIN v.order.P function#  
  v.order.P <- function (initial.K, ljlevels, v.order, beforeP) {  
    afterP <- array(NA, c((initial.K + 1), length(ljlevels), (max(ljlevels) + 1)), dimnames = list(c(paste("k", 0:initial.K, sep = "")), c(paste("j", 0:(length(ljlevels) - 1), sep = "")), c(paste("l", 0:max(ljlevels), sep = ""))))  
    k <- 1  
    for (k\_k in v.order) {  
      k <- k + 1  
      for (j in 2:length(ljlevels)) {  
        for (l in 2:(max(ljlevels[[j]]) + 1)) {  
          afterP[k, j, l] <- beforeP[(k\_k + 1), j, l]  
        }  
      }  
    }  
    return(afterP)  
  }  
  #####################END v.order.P function#  
    
  #####################BEGIN data.gamma function#  
  data.gamma <- function ([case.id](http://case.id/), data.object, internal.var, case.weight,   
                          initial.K, cell, ljlevels, l.levels.j,   
                          order.K, v.order, IG, FG,   
                          omega.fit, dec.char) {  
    IG[, "Patterns"] <- cell$Patterns  
    IG <- IG[-1, ]  
    row.names(IG) <- NULL  
    FG[, "Patterns"] <- cell$Patterns  
    FG$FreqPatterns <- cell$Freq  
    FG <- FG[-1,]  
    row.names(FG) <- NULL  
    if ((order.K == TRUE) && (!is.null(v.order))){  
      v.order <- (v.order + 1)  
      IG <- IG[, c(1, v.order)]  
      FG <- FG[, c(1, v.order, (max(v.order) + 1))]  
    }  
    names(IG) <- c("Patterns", paste("initial\_gik", 1:initial.K, sep = ""))  
    names(FG) <- c("Patterns", paste("final\_gik", 1:initial.K, sep = ""), "FreqPatterns")  
    names.object <- names(data.object)  
    data.object <- data.object[do.call(order, data.object[c(internal.var)]), ]  
    data.object$Patterns <- do.call(paste, c(as.list(data.object[c(internal.var)]), sep=""))  
    if (omega.fit == TRUE) {  
      data.object <- merge(data.object, IG, by = c("Patterns"), all.y = TRUE)  
      data.object <- merge(data.object, FG, by = c("Patterns"), all.y = TRUE)  
    } else {  
      data.object <- merge(data.object, IG, by = c("Patterns"))  
      data.object <- merge(data.object, FG, by = c("Patterns"))  
    }  
    data.object <- data.object[c(c(names.object), c("Patterns"), c("FreqPatterns"), c(paste("initial\_gik", 1:initial.K, sep = "")), c(paste("final\_gik", 1:initial.K, sep = "")))]  
    data.object <- data.object[do.call(order, data.object[[case.id](http://case.id/)]), ]  
    if (omega.fit == TRUE) {  
      if (![is.na](http://is.na/)(case.weight)) {  
        data.object[case.weight][[is.na](http://is.na/)(data.object[case.weight])] <- 1  
      }  
      st <- 0  
      for (j in 1:(length(ljlevels) - 1)) {  
        st <- (st + as.numeric((nchar(max(l.levels.j[[j + 1]])))))  
        data.object[, c(internal.var[j])] <- as.data.frame(substr(data.object[, "Patterns"], st, ((st + as.numeric(nchar(max(l.levels.j[[j + 1]])))) - 1)))  
      }  
    }  
    row.names(data.object) <- NULL  
    nf <- 0  
    repeat {  
      nf <- (nf + 1)  
      dataoutput <- paste (getwd(), "/GoMK", initial.K, "(", nf, ")", ".TXT", sep = "", collapse = NULL)  
      logname <- paste (getwd(), "/LogGoMK", initial.K, "(", nf, ")", ".TXT", sep = "", collapse = NULL)  
      if ((file.exists(dataoutput) == FALSE) && (file.exists(logname) == FALSE)) {  
        break  
      }  
    }  
    write.table(data.object, file = dataoutput, quote = FALSE, sep = " ", eol = "\n", na = ".", dec = dec.char, row.names = FALSE, col.names = TRUE, qmethod = c("escape", "double"), fileEncoding = "")  
    return (nf)  
  }  
  #####################END data.gamma function#  
    
  #####################BEGIN loggom function#  
  loggom <- function ([case.id](http://case.id/),  
                      case.weight,  
                      data.object,  
                      internal.var,  
                      initial.K, final.K,  
                      initial.gamma, initial.lambda,  
                      gamma.algorithm, lambda.algorithm,  
                      gamma.fit, lambda.fit, order.K, omega.fit,   
                      cell, ljlevels, l.levels.j, IP, FP, loglik, nf, dec.char, v.order) {  
    output <- paste (getwd(), "/LogGoMK", initial.K, "(", nf, ")", ".TXT", sep = "", collapse = NULL)  
    file.create(output)  
    sink(output)  
    cat(paste(date(), "\n\n", sep = "", collapse = NULL))  
    summary.parameters([case.id](http://case.id/),  
                       case.weight,  
                       data.object,  
                       internal.var,  
                       initial.K, final.K,  
                       initial.gamma, initial.lambda,  
                       gamma.algorithm, lambda.algorithm,  
                       gamma.fit, lambda.fit, order.K, omega.fit)  
    if (max(ljlevels, na.rm = TRUE) == 2) {  
      cat(paste("\n\n\*Note: All internal variables are dichotomous.\n", sep = "", collapse = NULL))  
    }  
    if (omega.fit == TRUE) {  
      cat(paste("\n\n\*Note ", (nrow(cell) - 1), " unique data patterns (I) has found for combinations of the all patterns.\n", sep = "", collapse = NULL))  
    } else if (omega.fit == FALSE) {  
      cat(paste("\n\n\*Note ", (nrow(cell) - 1), " unique data patterns (I) has found in data object.\n", sep = "", collapse = NULL))  
    }  
    charnamevar <- max(sapply(internal.var, nchar))  
    cat(paste("\n\nFrequency Table Original Data:\n", sep = "", collapse = NULL))  
    for (j in 1:(length(ljlevels) - 1)) {  
      if (![is.na](http://is.na/)(case.weight)) {  
        n <- xtabs(data.object[, case.weight] ~ data.object[, internal.var[j]], data.object)  
      } else {  
        n <- table(data.object[, internal.var[j]])  
      }  
      p <- prop.table(n)  
      for (l in l.levels.j[[(j + 1)]]) {  
        if (l == (min(l.levels.j[[(j + 1)]]))) {  
          if (j == 1) {  
            t <- do.call(paste, as.list(rep("", charnamevar)))  
            cat(paste(t, " \t", "   ", "\tn\t%\n", sep = "", collapse = NULL))  
          }  
          t <- do.call(paste, as.list(rep("", (charnamevar - (nchar(internal.var[j]))))))  
          cat(paste(internal.var[j], t, sep = "", collapse = NULL))  
          cat(paste("\t", "l", l, "\t", sep = "", collapse = NULL))  
        } else {  
          t <- do.call(paste, as.list(rep("", charnamevar)))  
          cat(paste(t, " \t", "l", l, "\t", sep = "", collapse = NULL))  
        }  
        if ((p[[l]] \* 100) < 10) {  
          cat(paste(format(n[[l]], nsmall = 0, decimal.mark = dec.char), "\t", "0", format(round((p[[l]] \* 100), 3), nsmall = 3, decimal.mark = dec.char), "\n", sep = "", collapse = NULL))  
        } else {  
          cat(paste(format(n[[l]], nsmall = 0, decimal.mark = dec.char), "\t", format(round((p[[l]] \* 100), 3), nsmall = 3, decimal.mark = dec.char), "\n", sep = "", collapse = NULL))  
        }  
      }  
      cat(paste("\n", sep = "", collapse = NULL))  
    }  
    LJ <- 0  
    for (i in 1:2) {  
      if (i == 1) {  
        cat(paste("\nPrimal Pure Type Probabilities:\n", sep = "", collapse = NULL))  
      } else {  
        cat(paste("\nLatter Pure Type Probabilities:\n", sep = "", collapse = NULL))  
      }  
      for (j in 2:length(ljlevels)) {  
        if (i == 1) {  
          LJ <- sum(LJ, max(l.levels.j[[j]]))  
        }  
        for (l in 2:(ljlevels[[j]] + 1)) {  
          if ((j == 2) && (l == (min(l.levels.j[[j]]) + 1))) {  
            t <- do.call(paste, as.list(rep("", charnamevar)))  
            cat(paste(t, "\t", "   ", sep = "", collapse = NULL))  
            for (k in 2:(initial.K + 1)) {  
              cat(paste("\tk", (k - 1), "    ", sep = "", collapse = NULL))  
            }  
            cat(paste("\n", sep = "", collapse = NULL))  
            t <- do.call(paste, as.list(rep("", (charnamevar - (nchar(internal.var[j - 1]))))))  
            cat(paste(internal.var[j - 1], t, sep = "", collapse = NULL))  
            cat(paste("\t", "l", l - 1, sep = "", collapse = NULL))  
          } else if (l == (min(l.levels.j[[j]]) + 1)) {  
            t <- do.call(paste, as.list(rep("", (charnamevar - (nchar(internal.var[j - 1]))))))  
            cat(paste(internal.var[j - 1], t, sep = "", collapse = NULL))  
            cat(paste("\t", "l", l - 1, sep = "", collapse = NULL))  
          }  
          for (k in 2:(initial.K + 1)) {  
            if((k == 2) && (l != (min(l.levels.j[[j]]) + 1))) {  
              t <- do.call(paste, as.list(rep("", charnamevar)))  
              cat(paste(t, " \t", "l", l - 1, sep = "", collapse = NULL))  
            }  
            if (i == 1) {  
              cat(paste("\t", format(round(IP[k, j, l], 4), nsmall = 4, decimal.mark = dec.char), sep = "", collapse = NULL))  
            } else {  
              cat(paste("\t", format(round(FP[k, j, l], 4), nsmall = 4, decimal.mark = dec.char), sep = "", collapse = NULL))  
            }  
          }  
          cat(paste("\n", sep = "", collapse = NULL))  
        }  
        cat(paste("\n", sep = "", collapse = NULL))  
      }  
    }  
    if (is.null(v.order)) {  
      cat(paste("\n\*Note: Could not organize pure type probabilities with a confidence interval of 99.0%.\n\n", sep = "", collapse = NULL))  
    }  
    AIC <- ((2 \* ((initial.K \* LJ) + (initial.K \* (sum(cell$Freq, na.rm = TRUE))))) - (2 \* loglik[2]))  
    cat(paste("\nPrimal Log-Likelihood is:       \t", format(round(loglik[1], 4), nsmall = 4, decimal.mark = dec.char) , "\n", sep = "", collapse = NULL))  
    cat(paste("\n\nLatter Log-Likelihood is:       \t", format(round(loglik[2], 4), nsmall = 4, decimal.mark = dec.char) , "\n", sep = "", collapse = NULL))  
    cat(paste("\n\nAkaike Information Criterion:   \t", format(round(AIC, 4), nsmall = 4, decimal.mark = dec.char) , "\n", sep = "", collapse = NULL))  
    cat(paste("\n\nLambda-Marginal Frequency Ratio (LMFR):\n", sep = "", collapse = NULL))  
    for (j in 2:length(ljlevels)) {  
      if (omega.fit == FALSE) {  
        if (![is.na](http://is.na/)(case.weight)) {  
          n <- xtabs(data.object[, case.weight] ~ data.object[, internal.var[(j - 1)]], data.object)  
        } else {  
          n <- table(data.object[, internal.var[(j - 1)]])  
        }  
      } else {  
        n <- xtabs(cell[-1, "Freq"] ~ cell[-1, internal.var[(j - 1)]], cell)  
      }  
      p <- prop.table(n)  
      for (l in l.levels.j[[j]]) {  
        if (l == (min(l.levels.j[[j]]))) {  
          if (j == 2) {  
            t <- do.call(paste, as.list(rep("", charnamevar)))  
            cat(paste(t, "\t", "   ", "\tn\t%", sep = "", collapse = NULL))  
            for (k in 2:(initial.K + 1)) {  
              cat(paste("\tk", (k - 1), "    ", sep = "", collapse = NULL))  
            }  
            for (k in 2:(initial.K + 1)) {  
              cat(paste("\tk", (k - 1), "/%lj", sep = "", collapse = NULL))  
            }  
            cat(paste("\n", sep = "", collapse = NULL))  
          }  
          t <- do.call(paste, as.list(rep("", (charnamevar - (nchar(internal.var[(j - 1)]))))))  
          cat(paste(internal.var[(j - 1)], t, sep = "", collapse = NULL))  
          cat(paste("\t", "l", l, "\t", sep = "", collapse = NULL))  
        } else {  
          t <- do.call(paste, as.list(rep("", charnamevar)))  
          cat(paste(t, " \t", "l", l, "\t", sep = "", collapse = NULL))  
        }  
        if ((p[[l]] \* 100) < 10) {  
          cat(paste(format(n[[l]], nsmall = 0, decimal.mark = dec.char), "\t", "0", format(round((p[[l]] \* 100), 3), nsmall = 3, decimal.mark = dec.char), sep = "", collapse = NULL))  
        } else {  
          cat(paste(format(n[[l]], nsmall = 0, decimal.mark = dec.char), "\t", format(round((p[[l]] \* 100), 3), nsmall = 3, decimal.mark = dec.char), sep = "", collapse = NULL))  
        }  
        for (k in 2:(initial.K + 1)) {  
          cat(paste("\t", format(round(FP[k, j, (l + 1)], 4), nsmall = 4, decimal.mark = dec.char), sep = "", collapse = NULL))  
        }  
        for (k in 2:(initial.K + 1)) {  
          cat(paste("\t", format(round((FP[k, j, (l + 1)] / p[[l]]), 4), nsmall = 4, decimal.mark = dec.char), sep = "", collapse = NULL))  
        }  
        cat(paste("\n", sep = "", collapse = NULL))  
      }  
      if (j != length(ljlevels)) {  
        cat(paste("\n", sep = "", collapse = NULL))  
      }  
    }  
    sink()  
    if (is.null(v.order)) {  
      cat(paste("\n\*Note: Could not organize pure type probabilities with a confidence interval of 99.0%.\n", sep = "", collapse = NULL))  
    }  
    cat(paste("\n\*Note: Saved frequency table, pure type probabilities and loglikelihood values to ", output, ".\n", sep = "", collapse = NULL))  
    cat(paste("\n\*Note: Saved original data with GoM scores to " , getwd(), "/GoMK", initial.K, "(", nf, ")", ".TXT", ".\n", sep = "", collapse = NULL))  
  }  
  #####################END loggom function#  
    
  if (!(is.data.frame (data.object))) {  
    stop("The data.object is not a data frame ...")  
  }  
  if (initial.K < 2) {  
    stop("The initial.K information is wrong ...")  
  }  
  if (final.K < 2) {  
    stop("The final.K information is wrong ...")  
  }  
  gamma.algorithm <- gamma.algorithm[1]  
  initial.gamma <- initial.gamma[1]  
  if ((gamma.fit != TRUE) & (gamma.fit != FALSE)) {  
    stop("The gamma.fit information is wrong ...")  
  }  
  lambda.algorithm <- lambda.algorithm[1]  
  initial.lambda <- initial.lambda[1]  
  if ((lambda.fit != TRUE) & (lambda.fit != FALSE)) {  
    stop("The lambda.fit information is wrong ...")  
  }  
  if (is.null(internal.var)) {  
    internal.var <- names(data.object)[!(names(data.object) %in% c(c([case.id](http://case.id/)), c(case.weight)))]  
  }  
  if ((order.K != TRUE) & (order.K != FALSE)) {  
    stop("The order.K information is wrong ...")  
  }  
  if ((omega.fit != TRUE) & (omega.fit != FALSE)) {  
    stop("The omega.fit information is wrong ...")  
  }  
  dec.char <- substr(dec.char, 1, 1)  
  if ((dec.char != ",") && (dec.char != ".")) {  
    dec.char <- "."  
  }  
  pathfolder <- getwd()  
  default.scipen <- options("scipen")  
  default.digits <- options("digits")  
  options(digits = 15)  
  FINAL.PARAMETERS <- vector("list", max(initial.K, final.K))  
  for (initial.K in initial.K:final.K) {  
    verify.parameters([case.id](http://case.id/),   
                      case.weight,  
                      data.object,  
                      internal.var,  
                      initial.gamma, initial.lambda,  
                      gamma.algorithm, lambda.algorithm,  
                      initial.gamma.object, initial.lambda.object)  
    summary.parameters([case.id](http://case.id/),  
                       case.weight,  
                       data.object,  
                       internal.var,  
                       initial.K, final.K,  
                       initial.gamma, initial.lambda,  
                       gamma.algorithm, lambda.algorithm,  
                       gamma.fit, lambda.fit, order.K, omega.fit)  
    cell <- cell.data(data.object, case.weight, internal.var, omega.fit)  
    ljlevels <- ljlevels.function(cell, internal.var)  
    l.levels.j <- l.levels.j.function(cell, internal.var)  
    baselevel <- c(rep(0, length(ljlevels)))  
    for (j in 2:length(ljlevels)) {  
      baselevel[j] <- l.levels.j[[j]][1]  
    }  
    names(baselevel) <- names(ljlevels)  
    if (final.K > (nrow(cell) - 1)) {  
      stop("The final.K information can not be greater than value of (I) ...")  
    }  
    cell <- as.matrix(cell)  
    row.names(cell) <- NULL  
    cell[, 1] <- NA  
    cell <- apply(cell, 2, as.numeric)  
    if (initial.gamma == c("gamma.object")) {  
      if (!is.data.frame(initial.gamma.object)) {  
        stop("The initial.gamma.object is not a data frame ...")  
      }  
    }  
    FG <- parameter.FG(initial.K, initial.gamma, cell, initial.gamma.object)  
    FG <- as.matrix(FG)  
    IG <- as.data.frame(round(FG, 4))  
    if (nrow(IG) != nrow(cell)) {  
      stop("The number of lines of initial.gamma.object is wrong ...")  
    }  
    if (initial.lambda == c("lambda.object")) {  
      if (!is.array(initial.lambda.object)) {  
        stop("The initial.lambda.object is not a array ...")  
      }  
    }  
    FITP <- fit.P(initial.K, initial.lambda, ljlevels, initial.lambda.object)  
    FP <- parameter.FP(initial.K, initial.lambda, ljlevels, initial.lambda.object)  
    FP <- as.array(FP)  
    IP <- array(NA, c((initial.K + 1), length(ljlevels), (max(ljlevels) + 1)), dimnames = list(c(paste("k", 0:initial.K, sep = "")), c(paste("j", 0:(length(ljlevels) - 1), sep = "")), c(paste("l", 0:max(ljlevels), sep = ""))))  
    for (k in 2:(initial.K + 1)) {  
      for (j in 2:length(ljlevels)) {  
        for (l in 2:(ljlevels[[j]] + 1)) {  
          IP[k, j, l] <- round(FP[k, j, l], 4)  
        }  
      }  
    }  
    ###############################C++###############################  
    if (require(Rcpp) && require(inline)) {  
      loglik <- GoM\_Model(baselevel, ljlevels, cell,   
                          gamma.algorithm, FG, gamma.fit,   
                          lambda.algorithm, FP, lambda.fit, FITP)  
    }  
    #################################################################  
    newfolder <- paste("K", initial.K, sep = "", collapse = NULL)  
    if (file.exists(newfolder) == FALSE) {  
      dir.create(newfolder, showWarnings = TRUE, recursive = FALSE, mode = "0777")  
    }  
    setwd(paste(pathfolder, "/K", initial.K, sep = "", collapse = NULL))  
    cell <- as.data.frame(cell)  
    cell$Patterns <- do.call(paste, c(as.list(cell[c(internal.var)]), sep=""))  
    FG <- as.data.frame(round(FG, 4))  
    FP <- round(FP, 4)  
    if (order.K == TRUE) {  
      v.order <- v.order.K(initial.K, cell, ljlevels, FP)  
      if (!is.null(v.order)) {  
        IP <- v.order.P(initial.K, ljlevels, v.order, IP)  
        FP <- v.order.P(initial.K, ljlevels, v.order, FP)  
      }  
    } else {  
      v.order <- as.integer(0)  
    }  
    options(scipen = 9999999)  
    options(digits = 4)  
    nf <- data.gamma([case.id](http://case.id/), data.object, internal.var, case.weight,   
                     initial.K, cell, ljlevels, l.levels.j,   
                     order.K, v.order, IG, FG,   
                     omega.fit, dec.char)  
    loggom([case.id](http://case.id/),   
           case.weight,   
           data.object,   
           internal.var,   
           initial.K, final.K,   
           initial.gamma, initial.lambda,   
           gamma.algorithm, lambda.algorithm,   
           gamma.fit, lambda.fit, order.K, omega.fit,  
           cell, ljlevels, l.levels.j, IP, FP, loglik, nf, dec.char, v.order)  
    setwd(pathfolder)  
    FG <- FG[-1, -1]  
    row.names(FG) <- NULL  
    if ((order.K == TRUE) && (!is.null(v.order))){  
      FG <- FG[, v.order]  
    }  
    names(FG) <- c(paste("gik", 1:initial.K, sep = ""))  
    FINAL.PARAMETERS[[initial.K]]$Gik <- FG  
    FP <- FP[-1, -1, -1]  
    FINAL.PARAMETERS[[initial.K]]$Pkjl <- FP  
    FINAL.PARAMETERS[[initial.K]]$Likelihood <- loglik[2]  
  }  
  options(scipen = default.scipen[[1]])  
  options(digits = default.digits[[1]])  
  return(FINAL.PARAMETERS)  
}  
#####################END GoMRcpp function#  
  
dados <- read.csv("C:\\Users\\Aluno\\Downloads\\ENADE2017V13\_Walter2.csv", sep = ";")  
names(dados)  
  
dados <- dados %>%  
  rename(SubjID='ï..SubjID')  
  
model <- GoMRcpp(data.object =  dados,  
                 initial.K = 3,  
                 final.K = 3,  
                 gamma.algorithm = "gradient.1992",  
                 initial.gamma = "random",  
                 gamma.fit = TRUE,  
                 lambda.algorithm = "gradient.1992",  
                 initial.lambda = "random",  
                 lambda.fit = TRUE,  
                 [case.id](http://case.id/) = "SubjID",  
                 order.K = TRUE,  
                 omega.fit = FALSE,  
                 dec.char = ",")