Functions Reference v.1.0.0

[Introduction 2](#_Toc481595403)

[Global variables 2](#_Toc481595404)

[Preferred variable names 2](#_Toc481595405)

[File: import.R 4](#_Toc481595406)

[import 4](#_Toc481595407)

[File: payoff.R 4](#_Toc481595408)

[Cx 4](#_Toc481595409)

[payoffDM 5](#_Toc481595410)

[CpayoffMatrix 5](#_Toc481595411)

[File: matching.R 6](#_Toc481595412)

[generateAssignmentMatrix 6](#_Toc481595413)

[CmatchMatrix 7](#_Toc481595414)

[Cmates 9](#_Toc481595415)

[Cmate 10](#_Toc481595416)

[File: inequalities.R 12](#_Toc481595417)

[Cineqmembers 12](#_Toc481595418)

[Cinequalities 16](#_Toc481595419)

[File: dataArray.R 16](#_Toc481595420)

[CdataArray 17](#_Toc481595421)

[File: objective.R 18](#_Toc481595422)

[Coefficient1 (global variable) 18](#_Toc481595423)

[objective 18](#_Toc481595424)

[File: maximize.R 19](#_Toc481595425)

[maximize 19](#_Toc481595426)

[File: confidence.R 20](#_Toc481595427)

[generateRandomSubsample 20](#_Toc481595428)

[pointIdentifiedCR 20](#_Toc481595429)

## Introduction

The full code contains more functions than those exposed in this document. A more detailed reference will be added in the next document revision. The curious reader can explore these inside the. R files.

The library contains several files that group together related functionality

FIles: {"import.R", "export.R", "payoff.R", "matching.R", "inequalities.R", "dataArray.R", "objective.R", "maximize.R", "confidence.R"}

## Global variables

For efficiency reasons, some global variables have been defined. Those variable names are used inside the code (and should not be modified).

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Name** | **Default Value** | **Description** |
| objective | coefficient1 | 1 | To normalize the payoff function, we set the first coefficient to either 1 or -1 coefficient1=1 (default) or coefficient=-1 |
| import | u | Null | upstream agents attributes |
| import | d | Null | downstream agent attributes |
| import | noAttr | Null | Number of attributes |

## Preferred variable names

We tried to use meaningful variable names inside the code and in the example files. These variables are listed here. It is not obligatory to use the ones below but we recommend them for consistency.

|  |  |  |
| --- | --- | --- |
| **Name** | **Description** | **Usage Examples** |
| header | The header list | {header, noM, noU, noD, noAttr, distanceMatrices, matchMatrix, mate} =  import(filename) |
| noM | Number of markets |
| noU | List of number of upstream agents per market |
| noD | List of number of downstream agents per market |
| noAttr | Number of attributes |
| distanceMatrices | The matrix containing the pair-level attributes ( distances, multiplications, etc) which characterize an upstream-downstream pair |
| matchMatrix | The matrix that shows who is matched with whom per market. Each element is ….? | matchMatrix=CmatchMatrix(payoffMatrix,quotaU,quotaD) |
| mate | A more “human readable” matching structure. Each element is |  |
| payoffMatrix | The matrix containing the payoffs of upstream-downstream pairings, according to the matching production function and the characteristics of agents in the match. | payoffMatrix = CpayoffMatrix(noM,noU,noD,Cx,distanceMatrices,noAttr) |
| ineqmembers | The list of pairings (matched and unmatched upstreams and donwstreams) that describe how the inequalities of the objective function are formed. Each element of this list shows the members of an inequality. | ineqmembers = Cineqmembers(mate) |
| inequalities | Applies the payoff function f o ineqmembers to create the data values that correspond to the inequalities of the objective function. | inequalities=Cinequalities(f,ineqmembers) |
| dataArray | A list of list, each element consisting of all data values which are necessary to evaluate an inequality. dataArray is the same as in the original code by Fox. | dataArray = CdataArray(distanceMatrices,ineqmembers) |

## File: import.R

import

Using functions: -

**Description**

{header,noM,noU,noD,noAttr,distanceMatrices,matchMatrix,mate}=import(filename)

imports a tab delimited file .dat that includes the precomputed data where the “distance attributes” between each pair of (u,d) agents have been already calculated.

noM- number of markets

noU- number of upstream agents

noD- number of downstream agents

noAttr- number of distance attributes characterizing a pair (u,d)

distanceMatrices is a matrix of noU x noD rows and noAttr columns which contains the “distance attributes” (i.e. multiplications of individual attributes or pair-specific attributes) between any pairs of upstream-downstream agents, matched or non-matched.

Datafiles consists of rows in the form {m,u,d,distance1, distance2, …, distance(noAttr), match (0 or 1)}.

File: payoff.R

*{Cx, payoffDM, CpayoffMatrix}*

Cx

Needs global variables: -

Creates global variables: x1,x2,x3,... (just creates, it does not set any value at this point)

Using functions: -

**Description**

Cx(n) creates a list of n variables named x1,x2,...,xn.

**Examples**

*Input:*

*Cx(3)*

*Output:*

*x1,x2,x3*

payoffDM

Needs global variables: Cx,distanceMatrices,noAttr

Creates global variables: -

Using functions: -

**Description**

payoffDM(Cx,distanceMatrices,noAttr,m,i,j) returns the payoff of i-upstream and j-upstream in the m-market.

It is used in the case of precomputed data. It is assumed that noAttr, Cx and distanceMatrices have been already assigned.

**Examples**

*Input:*

*distanceMatrix = ….*

*Output:*

*payoff[1,2,3] =...*

*payoff[2,1,1] = ...*

CpayoffMatrix

Needs global variables: noM, noU, noD, noAttr,Cx,distanceMatrices

Creates global variables: payoffMatrix

Using functions: -

**Description**

CpayoffMatrix(noM, noU, noD, noAttr,Cx,distanceMatrices) calculates and assigns the payoffMatrix.

payoffDM is used for precomputed data

File: matching.R

*{generateAssignmentMatrix, CmatchMatrix, Cmates, Cmate ,* *assignpayoffMatrix}*

generateAssignmentMatrix

Using functions: -

**Description**

generateAssignmentMatrix(payoffs,quotaU,quotaD)

Generates the optimal assignment of matches from the given matrix of payoffs for each match. The optimal assignment is the one that maximizes the total payoff (i.e. the sum of all payoffs) in a market. In an assignment matrix, each entry (i,j) is 1 if i and j are matched and 0 otherwise. The quota can be a number (the same for all streams) .

Notice that the quota is max number of matches per agent ( so in the data the real number of matches could be lower than the max).

**Examples**

*Input:*

*For 1-1 relationships (when quotaU=quotaD=1)*

*generateAssignmentMatrix(payoffMatrix,1,1)*

*For many to many relationships (when quotaU=n, quotaD=m)*

*generateAssignmentMatrix(payoffMatrix,n,m)*

*Output:*

*1-1*

$`1`

[1] *0, 0, 0, 0, 0, 0, 0, 0, 1*

$`2`

[1] *0, 0, 0, 0, 0, 0, 1, 0, 0*

$`3`

[1] *1, 0, 0, 0, 0, 0, 0, 0, 0*

$`4`

[1] *0, 0, 0, 1, 0, 0, 0, 0, 0*

$`5`

[1] *0, 0, 1, 0, 0, 0, 0, 0, 0*

*3-2*

$`1`

[1] *0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1*

$`2`

[1] *0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0*

$`3`

[1] *1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0*

$`4`

[1] *0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1*

$`4`

[1] *1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0*

CmatchMatrix

Creates global variables: matchMatrix

Using functions: generateAssignmentMatrix, assignpayoffMatrix

**Description**

CmatchMatrix(payoffMatrix,quotaU,quotaD)

Calculates and creates/updates the global variable 'matchMatrix'.

**Examples**

*Input:*

*CmatchMatrix[payoffMatrix, 2, 2]*

*Output:*

[[1]]

[[1]]$`1`

[1] *0, 0, 0, 0, 0, 1, 0, 0, 1, 0*

[[1]]$`2`

[1] *1, 0, 0, 1, 0, 0, 0, 0, 0, 0*

[[1]]$`3`

[1] *0, 0, 1, 0, 0, 1, 0, 0, 0, 0*

[[1]]$`4`

[1] *0, 0, 0, 1, 1, 0, 0, 0, 0, 0*

[[1]]$`5`

[1] *0, 1, 0, 0, 0, 0, 1, 0, 0, 0*

[[1]]$`6`

[1] *1, 0, 0, 0, 0, 0, 0, 1, 0, 0*

[[1]]$`7`

[1] *0, 0, 0, 0, 1, 0, 1, 0, 0, 0*

[[1]]$`8`

[1] *0, 1, 0, 0, 0, 0, 0, 0, 0, 1*

[[1]]$`9`

[1] *0, 0, 1, 0, 0, 0, 0, 0, 0, 1*

[[1]]$`10`

[1] *0, 0, 0, 0, 0, 0, 0, 1, 1, 0*

[[2]]

[[2]]$`1`

[1] *0, 0, 0, 0, 0, 0, 0, 0, 1, 1*

[[2]]$`2`

[1] *1, 0, 0, 0, 0, 0, 1, 0, 0, 0*

[[2]]$`3`

[1] *0, 0, 1, 1, 0, 0, 0, 0, 0, 0*

[[2]]$`4`

[1] *1, 1, 0, 0, 0, 0, 0, 0, 0, 0*

[[2]]$`5`

[1] *0, 0, 0, 0, 0, 0, 0, 1, 0, 1*

[[2]]$`6`

[1] *0, 0, 0, 1, 1, 0, 0, 0, 0, 0*

[[2]]$`7`

[1] *0, 0, 0, 0, 0, 1, 0, 0, 1, 0*

[[2]]$`8`

[1] *0, 0, 1, 0, 0, 0, 1, 0, 0, 0*

[[2]]$`9`

[1] *0, 1, 0, 0, 1, 0, 0, 0, 0, 0*

[[2]]$`10`

[1] *0, 0, 0, 0, 0, 1, 0, 1, 0, 0*

[[3]]

[[3]]$`1`

[1] *0, 0, 0, 0, 0, 0, 1, 1, 0, 0*

[[3]]$`2`

[1] *0, 0, 1, 0, 0, 0, 1, 0, 0, 0*

[[3]]$`3`

[1] *0, 0, 0, 0, 0, 0, 0, 0, 1, 1*

[[3]]$`4`

[1] *1, 1, 0, 0, 0, 0, 0, 0, 0, 0*

[[3]]$`5`

[1] *0, 0, 0, 0, 1, 0, 0, 0, 1, 0*

[[3]]$`6`

[1] *0, 0, 0, 0, 1, 1, 0, 0, 0, 0*

[[3]]$`7`

[1] *0, 0, 1, 0, 0, 1, 0, 0, 0, 0*

[[3]]$`8`

[1] *0, 0, 0, 1, 0, 0, 0, 1, 0, 0*

[[3]]$`9`

[1] *1, 1, 0, 0, 0, 0, 0, 0, 0, 0*

[[3]]$`10`

[1] *0, 0, 0, 1, 0, 0, 0, 0, 0, 1*

Cmates

Needs global variables: -

Creates global variables: mates

Using functions: -

**Description**

Cmates(matchMatrix) simplifies the matchMatrix to a list of triples that define matches across all markets. It provides another way to express all the matching information that is, which upstream is matched with which downstream and in which market. The output consists of a list of lists of triples each of which has the following structure: {market\_index,upstream\_index,downstream\_index}.

Output example:{{{1,1,3},{1,2},{1,3,1,2}},{{2,1,1},{2,2,1,3},{2,3,2}}}. In this example we have 2 lists, one per market and each inner list contains the triples. Note that in market 1, upstream agent 2 is not contributing which is fine.

This function is mainly used for the calculation of the total payoff - see Ctotalpayoff routine.

**Examples**

*Input:*

*matchMatrix<- list(list(c(0,0,1),c(0,0,0),c(1,1,0)),list(c(1,0,0,0),c(1,0,1,0),c(0,1,0,0)))*

*Cmates(matchMatrix)*

*Output:*

*[[1]]*

*[[1]][[1]]*

*[1] 1 1 3*

*[[1]][[2]]*

*[1] 1 2*

*[[1]][[3]]*

*[1] 1 3 1 2*

*[[2]]*

*[[2]][[1]]*

*[1] 2 1 1*

*[[2]][[2]]*

*[1] 2 2 1 3*

*[[2]][[3]]*

*[1] 2 3 2*

Cmate

Needs global variables: -

Creates global variables: mate

Using functions: -

**Description**

Cmate(matchMatrix) simplifies the matchMatrix from the original code by Santiago & Fox, to a matrix format which consists of lists of pairs, one pair per market. Here, each pair has the following structure: {{{1},{2},...,{noU within this market}},{{downstreams that are matched with upstream1},{downstreams that are matched with upstream2},...,{downstreams that are matched with upstream noU}}.

Example: {{{{1},{2},{3}},{{3},{},{1,2}}},{{{1},{2},{3}},{{1},{1,3},{2}}}}. In this example, there are three upstream and three downstream agents in each market, indexed 1, 2, 3. In the first market, upstream 1 is matched with downstream 3, upstream 2 is not matched, and upstream 3 is matched with downstream 1 and 2. In the second market, upstream 1 is matched with downstream 1, upstream 2 with downstream 1 and 3, and upstream 3 with downstream 2.

The mate=Cmate(matchMatrix) is later fed into the Cineqmembers routine.

**Examples**

matchMatrix<- list(list(c(0,0),c(0,1),c(1,1)), list(c(0,1,0),c(0,1,0)), list(c(1,1,0,0),c(1,0,1,1), c(1,0,1,0),c(1,1,1,1)))

Cmate(matchMatrix)

[[1]]

[[1]][[1]]

[[1]][[1]][[1]]

[1] 1 2 3

[[1]][[2]]

[[1]][[2]][[1]]

integer(0)

[[1]][[2]][[2]]

[1] 2

[[1]][[2]][[3]]

[1] 1 2

[[2]]

[[2]][[1]]

[[2]][[1]][[1]]

[1] 1 2

[[2]][[2]]

[[2]][[2]][[1]]

[1] 2

[[2]][[2]][[2]]

[1] 2

[[3]]

[[3]][[1]]

[[3]][[1]][[1]]

[1] 1 2 3 4

[[3]][[2]]

[[3]][[2]][[1]]

[1] 1 2

[[3]][[2]][[2]]

[1] 1 3 4

[[3]][[2]][[3]]

[1] 1 3

[[3]][[2]][[4]]

[1] 1 2 3 4

Another example:

Cmate(list(list(c(0,0,1),c(0,0,0),c(1,1,0)),list(c(1,0,0,0),c(1,0,1,0),c(0,1,0,0))))

Returns:

[[1]]

[[1]][[1]]

[[1]][[1]][[1]]

[1] 1 2 3

[[1]][[2]]

[[1]][[2]][[1]]

[1] 3

[[1]][[2]][[2]]

integer(0)

[[1]][[2]][[3]]

[1] 1 2

[[2]]

[[2]][[1]]

[[2]][[1]][[1]]

[1] 1 2 3

[[2]][[2]]

[[2]][[2]][[1]]

[1] 1

[[2]][[2]][[2]]

[1] 1 3

[[2]][[2]][[3]]

[1] 2

File: inequalities.R

*{Cineqmembers, Cinequalities}*

Cineqmembers

Creates global variables: ineqmembers

Using functions: -

**Description**

Cineqmembers(mate) generates all the members required to form the inequalities for many to many relationships defined by the mate. The produced list of lists of triples defines also the way inequalities are formed. At this time, inequalities are created to follow the theoretical proofs done by J Fox. CAUTION: ineqmembers is the largest object so it consumes a lot of memory.

**Examples**

*Input:*

*mate*

[[1]]

[[1]][[1]]

[[1]][[1]][[1]]

[1] 1 2 3

[[1]][[2]]

[[1]][[2]][[1]]

[1] 3

[[1]][[2]][[2]]

integer(0)

[[1]][[2]][[3]]

[1] 1 2

[[2]]

[[2]][[1]]

[[2]][[1]][[1]]

[1] 1 2 3

[[2]][[2]]

[[2]][[2]][[1]]

[1] 1

[[2]][[2]][[2]]

[1] 1 3

[[2]][[2]][[3]]

[1] 2

Cineqmembers(mate)

*Output:*

[[1]]

[[1]][[1]]

[[1]][[1]][[1]]

[[1]][[1]][[1]][[1]]

[1] 1 1 3

[[1]][[1]][[2]]

[[1]][[1]][[2]][[1]]

[1] 1 1 3

[[1]][[1]][[2]][[2]]

[1] 1 3 1

[[1]][[1]][[2]][[3]]

[1] 1 3 2

[[1]][[1]][[3]]

[[1]][[1]][[3]][[1]]

[1] 1 3 1

[[1]][[1]][[3]][[2]]

[1] 1 3 2

[[1]][[2]]

[[1]][[2]][[1]]

[[1]][[2]][[1]][[1]]

[1] 1 2 3

[[1]][[2]][[2]]

[[1]][[2]][[2]][[1]]

[1] 1 1 1

[[1]][[2]][[2]][[2]]

[1] 1 3 3

[[1]][[2]][[2]][[3]]

[1] 1 1 2

[[1]][[2]][[3]]

[[1]][[2]][[3]][[1]]

[1] 1 2 1

[[1]][[2]][[3]][[2]]

[1] 1 2 2

[[2]]

[[2]][[1]]

[[2]][[1]][[1]]

[[2]][[1]][[1]][[1]]

[1] 2 1 1

[[2]][[1]][[1]][[2]]

[1] 2 2 1

[[2]][[1]][[1]][[3]]

[1] 2 2 3

[[2]][[1]][[2]]

[[2]][[1]][[2]][[1]]

[1] 2 1 1

[[2]][[1]][[2]][[2]]

[1] 2 3 2

[[2]][[1]][[3]]

[[2]][[1]][[3]][[1]]

[1] 2 2 1

[[2]][[1]][[3]][[2]]

[1] 2 2 3

[[2]][[1]][[3]][[3]]

[1] 2 3 2

[[2]][[2]]

[[2]][[2]][[1]]

[[2]][[2]][[1]][[1]]

[1] 2 1 1

[[2]][[2]][[1]][[2]]

[1] 2 2 1

[[2]][[2]][[1]][[3]]

[1] 2 1 3

[[2]][[2]][[2]]

[[2]][[2]][[2]][[1]]

[1] 2 1 2

[[2]][[2]][[2]][[2]]

[1] 2 3 1

[[2]][[2]][[3]]

[[2]][[2]][[3]][[1]]

[1] 2 2 2

[[2]][[2]][[3]][[2]]

[1] 2 3 1

[[2]][[2]][[3]][[3]]

[1] 2 3 3

Cinequalities

Needs global variables: -

Creates global variables: inequalities

Using functions: -

**Description**

Cinequalities(f,ineqmembers) applies the f function to ineqmembers to create inequalities.

File: dataArray.R

*{CdataArray}*

CdataArray

Needs global variables: ineqmembers

Creates global variables: dataArray

Using functions: Cinequalities

**Description**

CdataArray(distanceMatrices ,ineqmembers) creates the dataArray.

**Examples**

Check the examples/precomputed\_numeric.Rmd

File: objective.R

*{coefficient1, objective}*

Coefficient1 (global variable)

**Description**

To normalize the payoff function, we set the first coefficient to either 1 or -1

coefficient1=1 (default) or coefficient=-1

objective

Needs global variables: dataArray, coefficient1

Updates global variables: -

Using functions: -

**Description**

Objective(x1,x2,...,xn) defines the objective function to minimize, as the number of satisfied inequalities. For a specific x-vector value we get a list of numbers. The negative number of positives is the outcome.

**Examples**

*Input:*

*coefficient1 = -1;*

*b<-(2,3)*

*dataArray=list(c(a11, a12, a13),c(a21, a22, a23),c(a31, a32, a33),c(a41, a42, a43),c(a51, a52, a53),c(a61, a62, a63))*

objective(dataArray,b)

*Output:*

*-UnitStep[-1. a11 + 2 a12 + 3 a13] - UnitStep[-1. a21 + 2 a22 + 3 a23] -*

*UnitStep[-1. a31 + 2 a32 + 3 a33] - UnitStep[-1. a41 + 2 a42 + 3 a43] -*

*UnitStep[-1. a51 + 2 a52 + 3 a53] - UnitStep[-1. a61 + 2 a62 + 3 a63]*

File: maximize.R

*{maximize}*

maximize

Need global variables: dataArray

Creates global variables: -

Using functions: objective

**Description**

Maximize(par) is MSE specific and uses the optimize function. It uses the objective function (that counts the negative number of satisfied inequalities). It returns a list (max,(x1->value1, x2->value2, ...)) where max is the maximum number of satisfied inequalities found and the solution of the maximization method (value1,value2,...). Implemented methods for now include only Differential Evolution.

For differential evolution method, the following parameters (par variable) should be specified:

* lower, upper: two vectors specifying scalar real lower and upper bounds on each parameter to be optimized, so that the i-th element of lower and upper applies to the i-th parameter. The implementation searches between lower and upper for the global optimum (minimum) of fn.
* CR: crossover probability from interval [0,1]. Default to 0.5.
* trace: Positive integer or logical value indicating whether printing of progress occurs at each iteration. The default value is TRUE. If a positive integer is specified, printing occurs every trace iterations.
* itermax: the maximum iteration (population generation) allowed. Default is 200.
* F: differential weighting factor from interval [0,2]. Default to 0.8.
* NP: number of population members. Defaults to NA; if the user does not change the value of NP from NA or specifies a value less than 4 it is reset when DEoptim is called as 10\*length(lower). For many problems it is best to set NP to be at least 10 times the length of the parameter vector.
* reltol: relative convergence tolerance. The algorithm stops if it is unable to reduce the value by a factor of reltol \* (abs(val) + reltol) after steptol steps. Defaults to sqrt(.Machine$double.eps), typically about 1e-8.
* RandomSeed: Random Seed to be used for result reproducibility

**Example**

lower <- c(-10, -10,-10,-10)

upper <- -lower

par<-list(lower=lower,upper=upper,NP=50,itermax=100,trace=FALSE,reltol=0.001,CR=0.5,F=0.6,RandomSeed=0)

x<-maximize(par)

g(bestmem,bestval)%=%x

Output:

bestmem

par1 par2 par3 par4

0.89823636 -4.91207964 2.45977145 0.07624073

bestval

[1] 95

File: confidence.R

*{generateRandomSubsample, pointIdentifiedCR}*

generateRandomSubsample

**Description**

generateRandomSubsample[ssSize,groupIDs,dataArray] generates a subsample of a given size from a data array.

Parameters:

ssSize - Size of the subsample generated, in terms of the number of distinct entities that will be represented in the subsample (ie,nests or coalitions).

groupIDs - A data map that the routine will use to examine the rows of the data array for possible inclusion into the subsample.

dataArray - A data array structure suitable for passing into the objective function.

**examples:**

input:

RandomKSubset[{1, 2, 3, 4, 5, 6, 7, 8}, 4]

output:

{2,5,6,7}

pointIdentifiedCR

Using functions: generateRandomSubsample, maximize

**Description**

pointIdentifiedCR(ssSize,numSubsamples,pointEstimate,args,groupIDs,dataArray,options,par)

generates a confidence region estimate using subsampling.

Parameters:

ssSize - The size of each subsample to be estimated.

numSubsamples -The number of subsamples to use in estimating the confidence region.

pointEstimate - The point estimate to build the confidence region around (typically the output

of pairwiseMSE).

objFunc - The objective function used in pairwiseMSE.

args - A list of unique symbols used in pairwiseMSE.

groupIDs - A data map used to generate the subsamples.

dataArray - The dataArray parameter used in pairwiseMSE.

options - An optional parameter specifying options. Available options are:

progressUpdate - How often to print progress (0 to disable).Default=0.

confidenceLevel - The confidence level of the region.Default=.95.

asymptotics - Type of asymptotics to use (nests or coalitions).Default=nests.

symmetric - True or False.If True,the confidence region will be

symmetric.Default=False.";

**Example**

Input: (This utilizes a datafile where attributes follow specific distributions)

ssSize<-3

options<-list()

options["progressUpdate"]<-1

options["confidenceLevel"]<-0.95

options["asymptotics"]<-"nests"

options["symmetric"]<-FALSE

numSubsamples<-50

pointEstimate<-as.numeric(bestmem)

b<-Cx[2:3]

lower <- c(-10, -10)

upper <- -lower

par<-list(lower=lower,upper=upper,NP=50,itermax=100,trace=FALSE,reltol=0.001,CR=0.5,F=0.6,RandomSeed=0)

pointIdentifiedCR(ssSize, numSubsamples,pointEstimate,Cx,groupIDs,dataArray,options,par)

Output:

Check the examples/precomputed\_numeric.Rmd