RLP Reference Manual

Rui Carlos Gonçalves

August 20, 2015

Abstract

This manual provides a brief description about how to use *RLP* software, which allows users to solve optimization problems using linear programming (*Simplex* algorithm). It also provides API documentation of the modules that comprise the application.

1 Application Options

When running *RLP*, there are four options that may be provided:

`input` this option requires a value, and it specifies the file from which the input data is read (if this option is not provided, the input data is read from the *stdin*);

`tables` specifies whether the intermediate tables generated during the iterations of the algorithm should be printed (by default, the tables are sent to the *stdout*);

`output` this option requires a value, and it specifies the file to which the output tables are printed (thus, it is only useful if the `tables` option is set);

`help` shows the application help;

`version` shows info about the program.

2 Syntax of Input Data

The input data can be read from an input file, or directly from the standard input. In any case, the following syntax should be used:

- the first line must start by `MIN` or `MAX`, according to the kind of optimization problem;
- next, it should be provided the name of the variable[1] that identifies the function to optimize (the *objective function*);
- next, it must be used the character =, followed by the expression defining the objective function (this expression must be a polynomial, where each term has degree 1, and each of them is defined by a real number followed by a variable—the multiplication symbol is optional);
- next, expression `ST` (subject to) defines the end of the objective function, and the beginning of the restrictions;
- finally, the restrictions are specified, one per line (they are also defined by a polynomial of degree one, followed by a relational operator—<=, >=, or =—and a real number).

---

[1] All variable names must start with a lowercase letter, which may be followed by any number or letter.

2.1 Example

```
MAX z = 2x + 3y1 - 2y2
ST x + y = 2
   -y + 2y1 <= 20
   y1 - y2 <= 0
```

specifies the problem:

$$
\begin{array}{rcccccccc}
Max\, z = & 2x & + & 3y_1 & + & 2y_2 & & & \\
s.t. & x & & & & & + & y & = & 2 \\
 & & & 2y_1 & & & - & y & \leq & 20 \\
 & & & y_1 & - & y_2 & & & \leq & 0 \\
 & \multicolumn{5}{l}{x, y_1, y_2, y \geq 0} & & & &
\end{array}
$$

3 Application Implementation

To solve the optimization problems, the Simplex algorithm is used. If necessary, the Simplex Dual algorithm is used in a first phase. The restrictions using the relation operator = are converted to two restrictions using operators <= and >=.

4 Format of Output Tables

The first row is always used to define the objective function. The remaining rows define the restrictions. Regarding the columns, the first ones are associated with the variables of the objective function, and the next are associated with the *free* variables. The last two columns have a special meaning, which depends on the row. In the first row we have the current value of the objective function, and the value `0`. For the remaining rows, we have the value associated to each variable at the moment, and an index of the variable.

A Source Code

A.1 rlpl.l (lexical analyser)

```
%{
#include <string.h>
#include "rlpy.h"
%}

real    [0-9]+(\.)?[0-9]*
var     [a-z][a-zA-Z0-9]*

%option yylineno
%option nounput
%option noinput

%%

"ST"            {return ST;}
"MIN"           {return MIN;}
"MAX"           {return MAX;}
{real}          {yylval.real=atof(yytext);return REAL;}
{var}           {yylval.str=strdup(yytext);return VAR;}
"+"             {return OPADD;}
"-"             {return OPSUB;}
"*"             {return OPMULT;}
```

```
"="             {return OPEQ;}
">="            {return OPGE;}
"<="            {return OPLE;}
"("             {return OP;}
")"             {return CP;}
[ \n\t]         {}

%%

int yywrap()
{return 1;}
```

A.2 rlpy.y (parser)

```
%{
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include "hashmap.h"
#include "array.h"
#include "read.h"
#include "main.h"
#include "rlp.h"

extern FILE* yyin;
extern int yylineno;
extern char* yytext;

int type=0;
int varc=0;
int condc=0;
int eqc=0;
char* vobj=NULL;
Prob prob=NULL;

int yylex();
void yyerror(const char* msg)
{fprintf(stderr,"ERROR! line %d: %s - %s\n",yylineno,yytext,msg);exit(255);}
%}

%union{
        double real;
        int integer;
        char* str;
      }

// Terminals
%token OPADD    "+"
%token OPSUB    "-"
%token OPMULT   "*"
%token OPEQ     "="
%token OPLE     "<="
%token OPGE     ">="
%token OP       "("
%token CP       ")"
%token MIN      "MIN"
%token MAX      "MAX"
%token ST       "ST"

// Pseudo-terminals
```

```
%token <real> REAL
%token <str> VAR

// Non-terminals
%type <real> Real
%type <integer> OpRel Sig

%%

Prob    : Type VAR "=" Exp "ST" LCond   {vobj=$2;}
        ;
Type    : "MAX"                         {type=-1;}
        | "MIN"                         {type=1;}
        ;
Exp     : Head Tail                     {condc++;}
        ;
Head    : VAR                           {varc+=addCoefHead(prob,$1,varc,1);}
        | Sig VAR                       {varc+=addCoefHead(prob,$2,varc,$1);}
        | REAL OptS VAR                 {varc+=addCoefHead(prob,$3,varc,$1);}
        | Sig REAL OptS VAR             {varc+=addCoefHead(prob,$4,varc,$1*$2);}
        ;
Tail    :                               {}
        | Tail Prod                     {}
        ;
Prod    : Sig VAR                       {varc+=addCoefTail(prob,$2,varc,$1);}
        | Sig Real OptS VAR             {varc+=addCoefTail(prob,$4,varc,$2*$1);}
        ;
Real    : "(" Sig REAL ")"              {$$=$2*$3;}
        | REAL                          {$$=$1;}
        ;
Sig     : "+"                           {$$=1;}
        | "-"                           {$$=-1;}
        ;
OptS    :                               {}
        | "*"                           {}
        ;
LCond   : Exp OpRel Real                {setOpRHS(prob,$2,$3);}
        | LCond Exp OpRel Real          {setOpRHS(prob,$3,$4);}
        ;
OpRel   : ">="                          {$$=-1;}
        | "<="                          {$$=1;}
        | "="                           {$$=0;eqc++;}
        ;

%%

int main(int argc,char** argv)
{
  double* matrix;
  int err,ftab;
  char* in,*out;
  FILE* fin,*fout;

  in=NULL;
  out=NULL;
  fin=NULL;
  fout=NULL;

  if((ftab=opt(argc,argv,&in,&out))<0) return 0;

  if(in)
  {
    fin=fopen(in,"r");

    if(!fin)
    {
      fprintf(stderr,"ERROR! Can not open file \'%s\'.\n",in);
      exit(254);
    }
    yyin=fin;
  }

  if(ftab)
  {
    if(out)
    {
      fout=fopen(out,"w");
      if(!fout) fprintf(stderr,"ERROR! Can not open file \'%s\'.\n",out);
    }
    else fout=stdout;
  }

  prob=newProb();

  yyparse();

  matrix=loadMatrix(prob,varc,condc-1,eqc,type);

  err=simplex(matrix,varc,condc-1+eqc,fout);

  if(err)
  {
    printf("Can not solve this problem.\n");
    return 1;
  }

  puts("");
  printRes(prob->invpos,matrix,vobj,varc,condc+eqc-1,type);
  puts("");

  listDelete(prob->exps);
  arrayMap(prob->invpos,free);
  arrayDelete(prob->invpos);
  hashDelete(prob->pos);
  return 0;
}
```

A.3 read.* (auxiliary read functions)

```
/**
 * Definitions of data types and functions required to read and store the
 * data about the problem.
 *
 * @author Rui Carlos Gonçalves
 * @file read.h
 * @version 1.5
 * @date 08/2015
 */
#ifndef _READ_
#define _READ_
```

```
#include "array.h"
#include "hashmap.h"
#include "list.h"

/**
 * Expression structure.
 */
typedef struct sExp
{
  /// Coefficients of the variables.
  Array coefs;
  /// Operator of the expression.
  int op;
  /// Right-hand side of the expression.
  double rhs;
}SExp;

/**
 * Expression definition.
 */
typedef SExp* Exp;

/**
 * Problem structure.
 */
typedef struct sProb
{
  /// Variables and their indexes.
  HashMap pos;
  /// Variables on each index.
  Array invpos;
  /// List of expressions (objective function, and conditions).
  List exps;
}SProb;

/**
 * Problem definition.
 */
typedef SProb* Prob;

//################################################################################

/**
 * Creates an expression.
 *
 * @param arraysize size of the coefficients array.
 *
 * @return <tt>NULL</tt> if an error occurred; the new expression otherwise.
 */
Exp newExp(int arraysize);

/**
 * Creates a problem.
 *
 * @return <tt>NULL</tt> if an error occurred; the new problem otherwise.
 */
Prob newProb();

/**
 * Hash function for variables names.
 *
 * @param varid the variable identifier.
 *
 * @return the hash code of the variable.
 */
int varhash(const char* varid);

/**
 * Adds a new condition to the problem, and adds the coefficients of a variable.
 * If the variable did not exist in the variables set, it is added.
 *
 * @param prob  the problem.
 * @param varid the variable identifier.
 * @param varc  the index of the variable if it does not exist.
 * @param coef  the coefficient of the variable.
 *
 * @return 0 if the variable already existed; 0 otherwise.
 */
int addCoefHead(Prob prob,const char* varid,int varc,double coef);

/**
 * Adds the coefficient of a variable.
 * If the variable did not exist in the variables set, it is added.
 *
 * @param prob  the problem.
 * @param varid the variable identifier.
 * @param varc  the index of the variable if it does not exist.
 * @param coef  the coefficient of the variable.
 *
 * @return 0 if the variable already existed; 0 otherwise.
 */
int addCoefTail(Prob prob,const char* varid,int varc,double coef);

/**
 * Adds the relational operator and the right-hand side to a condition.
 *
 * @param prob the problem.
 * @param op   the identifier of the relational operator.
 * @param rhs  the right-hand side of the condition.
 *
 * @return 0.
 */
int setOpRHS(Prob prob,int op,double rhs);

#endif
```

```
/**
 * Definitions of data types and functions required to read and store the
 * data about the problem.
 *
 * @author Rui Carlos Gonçalves
 * @file read.c
 * @version 1.5
 * @date 08/2015
 */
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "read.h"

/**
 * Given a row (<tt>R</tt>), a column (<tt>C</tt>), and the number of columns
 * (<tt>NC</tt>) of a matrix, computes an equivalent 1D position.
 */
#define POS(R,C,NC) ((R)*(NC)+(C))

/// Error messages.
static char* errors[]={"ERROR! Function \'setCoefHead\' -> \'malloc\'.\n"
                      ,"ERROR! Function \'setCoefHead\' -> \'listInsertLst\'.\n"
                      ,"ERROR! Function \'setCoefHead\' -> \'arrayInsert\'.\n"
                      ,"ERROR! Function \'setCoefHead\' -> \'newExp\'.\n"
                      ,"ERROR! Function \'setCoefHead\' -> \'hashInsert\'.\n"
                      ,"ERROR! Function \'setCoefTail\' -> \'malloc\'.\n"
                      ,"ERROR! Function \'setCoefTail\' -> \'hashInsert\' or"
                      "\'arrayInsert\'.\n"
                      ,"ERROR! Function \'setCoefTail\' -> \'arrayInsert\'.\n"
                      };

//################################################################################

Exp newExp(int size)
{
  Exp res;

  res=malloc(sizeof(SExp));

  if(res)
  {
    res->coefs=newArray(size);
    res->op=0;
    res->rhs=0;
  }

  return res;
}

//################################################################################

Prob newProb()
{
  Prob res;

  res=malloc(sizeof(SProb));

  if(res)
  {
    res->pos=newHash(32,0.6,(int(*)(void*))varhash,(int(*)(void*,void*))strcmp);
    res->invpos=newArray(10);
    res->exps=newList();
  }

  return res;
}

//################################################################################

int varhash(const char* varid)
{
  int i,res;

  for(i=0,res=0;varid[i];i++)
    res+=varid[i];
  return res;
}

//###############################################################################

int addCoefHead(Prob prob,const char* varid,int varc,double coef)
{
  int* vpos,pos,erro,res;
  double* tmp;
  Exp exp;

  vpos=&pos;
  res=0;

  if(hashGet(prob->pos,(void*)varid,(void**)&vpos))
  {
    vpos=malloc(sizeof(int));

    if(!vpos){
      fputs(errors[0],stderr);
      exit(1);
    }

    *vpos=varc;
    erro=hashInsert(prob->pos,(void**)varid,vpos,0);
    erro+=arrayInsert(prob->invpos,varc,(void**)varid,0);

    if(erro){
      fputs(errors[4],stderr);
      exit(5);
    }

    res=1;
  }

  exp=newExp((varc+1>10)?(varc+1):10);

  if(!exp){
    fputs(errors[3],stderr);
    exit(4);
  }

  erro=listInsertLst(prob->exps,exp);

  if(erro){
    fputs(errors[1],stderr);
    exit(2);
  }

  tmp=malloc(sizeof(double));

  if(!tmp){
    fputs(errors[0],stderr);
    exit(1);
  }

  *tmp=coef;
  erro=arrayInsert(exp->coefs,*vpos,tmp,0);

  if(erro){
    fputs(errors[2],stderr);
    exit(3);
  }

  return res;
}

//##############################################################################

int addCoefTail(Prob prob,const char* varid,int varc,double coef)
{
  int* vpos,pos,res,error;
  double* tmp;
  Exp exp;

  vpos=&pos;
  res=0;

  if(hashGet(prob->pos,(void*)varid,(void**)&vpos))
  {
    vpos=malloc(sizeof(int));

    if(!vpos){
      fputs(errors[5],stderr);
      exit(6);
    }

    *vpos=varc;
    error=hashInsert(prob->pos,(void**)varid,vpos,0);
    error+=arrayInsert(prob->invpos,varc,(void**)varid,0);

    if(error)
    {
      fputs(errors[6],stderr);
      exit(7);
    }

    res=1;
  }

  listLst(prob->exps,(void**)&exp);
  tmp=malloc(sizeof(double));

  if(!tmp){
    fputs(errors[5],stderr);
    exit(6);
  }

  *tmp=coef;
  error=arrayInsert(exp->coefs,*vpos,tmp,0);

  if(error)
  {
    fputs(errors[7],stderr);
    fprintf(stderr,"\n%d\n\n",error);
    exit(8);
  }

  return res;
}

//##############################################################################
```

```
int setOpRHS(Prob prob,int op,double rhs)
{
  Exp exp;

  listLst(prob->exps,(void**)&exp);

  exp->op=op;
  exp->rhs=rhs;

  return 0;
}
```

A.4 main.* (other auxiliary functions)

```
/**
 * Definition of functions used by main.
 *
 * @author Rui Carlos Gonçalves
 * @file main.h
 * @version 1.5
 * @date 08/2015
 */
#ifndef _MAIN_
#define _MAIN_

#include "array.h"
#include "read.h"

/**
 * From the info collected by the parser, creates a matrix representing the
 * problem to solve, according to the format required by function
 * <tt>simplex</tt>.
 *
 * @param prob  info about the problem.
 * @param varc  number of variables.
 * @param condc number of conditions.
 * @param eqc   number of conditions with an equality operator.
 * @param type  determines whether we have a maximization or minimization
 * problem (max.: -1, min.: 1).
 *
 * @return the matrix created.
 */
double* loadMatrix(Prob prob,int varc,int condc,int eqc,int type);

/**
 * Prints the result.
 *
 * @param vars  name of the variables of each position of the table.
 * @param tab   table resulting from applying the <em>Simplex</em> algorithm.
 * @param varob variable to maximize/minimize.
 * @param varc  number of variables.
 * @param condc number of conditions
 * @param type  determines whether we have a maximization or minimization
 * problem (max.: -1, min.: 1).
 */
void printRes(Array vars,double* tab,const char* varob,int varc,int condc,int type);

/**
 * Checks the options specified by the user.
 *
 * @param argc number of options.
 * @param argv value of the options.
 * @param in   address for the input option.
 * @param out  address for the output option.
 *
 * @return 1 if the <tt>tables</tt> option was set; 0 otherwise.
 */
int opt(int argc,char** argv,char** in,char** out);

#endif
```

```
/**
 * Definition of functions used by main.
 *
 * @author Rui Carlos Gonçalves
 * @file main.c
 * @version 1.5
 * @date 08/2015
 */
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include "main.h"

/**
 * Given a row (<tt>R</tt>), a column (<tt>C</tt>), and the number of columns
 * (<tt>NC</tt>) of a matrix, computes an equivalent 1D position.
 */
#define POS(R,C,NC) ((R)*(NC)+(C))

/// Error messages.
static char* errors[]={"ERROR! Function \'loadMatrix\' -> \'calloc\'.\n"
                      ,"ERROR! Function \'printRes\' -> \'newArray\'.\n"
                      ,"ERROR! Function \'printRes\' -> \'malloc\'.\n"
                      ,"ERROR! Function \'printRes\' -> \'arrayInsert\'.\n"
                      ,"ERROR! Invalid parameters.\n"
                      };

//################################################################################

double* loadMatrix(Prob prob,int varc,int condc,int eqc,int type)
{
  double* res,*tmp;
  int row,col,k;
  Exp exp;

  k=varc+condc+eqc+2;
  res=calloc((condc+1+eqc)*(varc+condc+eqc+2),sizeof(double));

  if(!res){
    fputs(errors[0],stderr);
    exit(1);
  }

  listRemoveFst(prob->exps,(void**)&exp);

  res[POS(0,k-2,k)]=0;
  res[POS(0,k-1,k)]=0;

  for(col=0;col<k-2;col++)
  {
    arrayAt(exp->coefs,col,(void**)&tmp);
    res[POS(0,col,k)]=tmp?type*(*tmp):0;
  }

  arrayMap(exp->coefs,free);
  arrayDelete(exp->coefs);
  free(exp);

  for(row=1;!listRemoveFst(prob->exps,(void**)&exp);)
  {
    if(exp->op>-1)
    {
      res[POS(row,k-2,k)]=exp->rhs;
      res[POS(row,k-1,k)]=row+varc-1;

      for(col=0;col<varc;col++)
      {
        arrayAt(exp->coefs,col,(void**)&tmp);
        res[POS(row,col,k)]=tmp?*tmp:0;
      }

      row++;

      arrayMap(exp->coefs,free);
      arrayDelete(exp->coefs);
      free(exp);
    }

    if(exp->op<1)
    {
      res[POS(row,k-2,k)]=-1*(exp->rhs);
      res[POS(row,k-1,k)]=row+varc-1;

      for(col=0;col<varc;col++)
      {
        arrayAt(exp->coefs,col,(void**)&tmp);
        res[POS(row,col,k)]=tmp?-(*tmp):0;
      }

      row++;
    }
  }

  for(row=0;row<condc;row++)
    for(col=0;col<condc;col++)
      res[POS(row+1,col+varc,k)]=row==col;

  return res;
}

//################################################################################

void printRes(Array vars,double* tab,const char* varob,int varc,int condc,int type)
{
  int i,k,erro;
  double* tmp;
  char* str;
  Array res;

  k=varc+condc+2;
  res=newArray(varc);

  if(!res){
    fputs(errors[1],stderr);
    exit(2);
  }

  for(i=1;i<condc+1;i++)
    if(tab[POS(i,k-1,k)]<varc)
    {
      tmp=malloc(sizeof(double));

      if(!tmp){
        fputs(errors[2],stderr);
        exit(3);
      }

      *tmp=tab[POS(i,k-2,k)];
      erro=arrayInsert(res,tab[POS(i,k-1,k)],tmp,0);

      if(erro){
        fputs(errors[3],stderr);
        exit(4);
      }
    }

  puts("====================");
  printf("%s = %f\n",varob,-type*tab[POS(0,k-2,k)]);
  puts("--------------------");
  for(i=0;i<varc;i++)
  {
    arrayAt(vars,i,(void**)&str);
    printf("%s =",str);
    if(!arrayAt(res,i,(void**)&tmp)) printf(" %f\n",*tmp);
    else printf(" 0.0\n");
  }
  puts("====================");

  arrayMap(res,free);
  arrayDelete(res);
}

//##############################################################################

int opt(int argc,char** argv,char** in,char** out)
{
  int c,res;

  static struct option options[]={{"output" ,required_argument,NULL,'o'}
                                 ,{"input"  ,required_argument,NULL,'i'}
                                 ,{"tables" ,no_argument      ,NULL,'t'}
                                 ,{"help"   ,no_argument      ,NULL,'h'}
                                 ,{"version",no_argument      ,NULL,'v'}
                                 ,{0,0,0,0}
                                 };

  res=0;

  while((c=getopt_long(argc,argv,"o:i:thv",options,NULL))!=-1)
  {
    switch(c)
    {
      case ’o’ : *out=optarg;
                 break;
      case ’i’ : *in=optarg;
                 break;
      case ’t’ : res+=1;
                 break;
      case ’h’ : puts("rlp version 1.5, Copyright (C) 2006, 2009, 2015 Rui Carlos Goncalves\n"
                      "rlp comes with ABSOLUTELY NO WARRANTY.  This is free software, and\n"
                      "you are welcome to redistribute it under certain conditions.  See the GNU\n"
                      "General Public Licence for details.\n\n"
                      "Solve linear programming problems using simplex method.\n\n"
                      "Options:\n"
                      "  --help                Print this help\n"
                      "  --version             Print version info\n"
                      "  --input=<file>        Input file (default: stdin)\n"
                      "  --tables              Print all tables\n"
                      "  --output=<file>       Output file (default: stdout)"
                     );
                 res-=2;
                 break;
      case ’v’ : puts("rlp version 1.5, Copyright (C) 2006, 2009, 2015 Rui Carlos Goncalves\n"
                      "rlp comes with ABSOLUTELY NO WARRANTY.  This is free software, and\n"
                      "you are welcome to redistribute it under certain conditions.  See the GNU\n"
                      "General Public Licence for details."
                     );
                 res-=2;
                 break;
      default  : break;
    }
  }

  if(optind<argc){
    fputs(errors[4],stderr);
    exit(5);
  }

  return res;
}
```

A.5 rlp.* (simplex algorithm)

```
/**
 * Implementation of linear programming functions.
 *
 * @author Rui Carlos Gonçalves
 * @file rlp.h
 * @version 3.0
 * @date 07/2012
 */
#ifndef _RLP_H_
#define _RLP_H_

/**
 * Applies the <em>Simplex Algorithm</em> to an optimization problem.
 *
 * Given a problem with <em>n</em> variables (<em>x<sub>1</sub></em>, ...,
 * <em>x<sub>n</sub></em>) and <em>m</em> conditions (<em>c<sub>1</sub> =
 * b<sub>1</sub></em>, ..., <em>c<sub>m</sub> = b<sub>m</sub></em>), the
 * function must receive a matrix <tt>a</tt> of size <em>(m+1)*(n+m+2)</em>,
 * containing:
 * <ul>
 *   <li>at <tt>a[0][i-1]</tt> (for <em>i</em> in <em>[1, n]</em>) the
 *   coefficient of variable <em>x<sub>i</sub></em> in the expression to
 *   minimize;</li>
 *   <li>at <tt>a[0][i]</tt> (for <em>i</em> in <em>[n, n+m+1]</em>) the value
 *   0;</li>
 *   <li>at <tt>a[i][j-1]</tt> (for <em>i</em> in <em>[1, m]</em>, and for
 *   <em>j</em> in <em>[1, n]</em>) the coefficient of variable
 *   <em>x<sub>j</sub></em> in condition <em>c<sub>i</sub></em>;</li>
 *   <li>at <tt>a[i][j]</tt> (for <em>i</em> in <em>[1, m]</em>, and for
 *   <em>j</em> in <em>[n, n+m-1]</em>) the identity matrix;</li>
 *   <li>at <tt>a[i][n+m]</tt> (for <em>i</em> in <em>[1, m]</em>) the value of
 *   <em>b<sub>i</sub></em>;</li>
 *   <li>at <tt>a[i][n+m+1]</tt> (for <em>i</em> in <em>[1, m]</em>) the value
 *   of <em>n+i</em>.</li>
 * </ul>
 *
 * Allows to specify the file where the table resulting from the application of
 * the algorithm (using the parameter <em>file</em>).
 *
 * @param a    matrix that represents the problem
 * @param n    number of variable of the function
 * @param m    number restrictions
 * @param file file where the tables will be saved (or <tt>NULL</tt>)
 *
 * @return
 * 0 if it is possible to solve the problem\n
 * 1 otherwise
 */
int simplex(double* a,int n,int m,FILE* file);

/**
 * Applies the <em>Simplex algorithm</em> to a primal optimization problem.
 *
 * The input matrix (<tt>a</tt>) must follow the format defined in function
 * <tt>\ref simplex</tt>.
 *
 * Allows to specify the file where the table resulting from the application of
 * the algorithm (using the parameter <em>file</em>).
 *
 * @param a    matrix that represents the problem
 * @param n    number of variables of the function
 * @param m    number restrictions
 * @param pos  position of the minimum value of the restrictions’ column (it
 *                must be a negative value)
 * @param file file where the tables will be saved (or <tt>NULL</tt>)
 *
 * @return
 * 0 if it is possible to solve the problem\n
 * 1 otherwise
 */
int simplexp(double* a,int n,int m,int pos,FILE* file);

/**
 * Applies the <em>Simplex algorithm</em> to a dual optimization problem.
 *
 * The input matrix (<tt>a</tt>) must follow the format defined in function
 * <tt>\ref simplex</tt>.
 *
 * Allows to specify the file where the table resulting from the application of
 * the algorithm (using the parameter <em>file</em>).
 *
 * @param a    matrix that represents the problem
 * @param n    number of variables of the function
 * @param m    number restrictions
 * @param pos  position of the minimum value of the first row (it must be a
 *             negative value)
 * @param file file where the tables will be saved (or <tt>NULL</tt>)
 *
 * @return
 * 0 if it is possible to solve the problem\n
 * 1 otherwise
 */
int simplexd(double* a,int n,int m,int pos,FILE* file);

#endif
```

```
/**
 * Implementation of linear programming functions.
 *
 * @author Rui Carlos Gonçalves
 * @file rlp.c
 * @version 3.0
 * @date 07/2012
 */
#include <stdio.h>
#include <float.h>
#include <math.h>
#include "rlp.h"

/**
 * Given a row (<tt>R</tt>), a column (<tt>C</tt>), and the number of columns
 * (<tt>NC</tt>) of a matrix, computes an equivalent 1D position.
 */
#define POS(R,C,NC) ((R)*(NC)+(C))

/**
 * Prints a matrix associated to an optimization problem.
 *
 * The file <tt>file</tt>, where the matrix will be printed, must be opened
 * before calling this function.
 *
 * @param matrix the matrix to print
 * @param nrows  the number of rows
 * @param ncols  the number of columns
 * @param file where the tables will be printed
 */
static void fmprint(double* matrix,int nrows,int ncols,FILE* file)
{
  int i,j;
  fprintf(file,"\n  +");
  for(i=0;i<ncols;i++)
  {
    fprintf(file,"--------------+");
  }
  fputs("\n",file);
  for(i=0;i<nrows;i++)
  {
    fprintf(file,"  |");
    for(j=0;j<ncols;j++)
    {
      fprintf(file," %12.4f |",matrix[POS(i,j,ncols)]);
    }
    fputs("\n",file);
  }
  fprintf(file,"  +");
  for(i=0;i<ncols;i++)
  {
    fprintf(file,"--------------+");
  }
  fputs("\n\n",file);
}

//===============================================================================

/**
 * Finds the minimum value of a matrix's column.
 *
 * @param matrix the matrix
 * @param nrows  the number of rows
 * @param ncols  the number of columns
 * @param col    the column index
 * @param row    pointer where the row that contains the minimum value will be
 *               put
 *
 * @return
 * minimum value of the specified column
 */
static double minimumc(double* matrix,int nrows,int ncols,int col,int* row)
{
  int i;
  double res=DBL_MAX;
  for(i=0;i<nrows;i++)
  {
    if(matrix[POS(i,col,ncols)]<res)
    {
      res=matrix[POS(i,col,ncols)];
      *row=i;
    }
  }
  return res;
}

//===============================================================================

/**
 * Finds the minimum value of a matrix's row.
 *
 * @param matrix the matrix row
 * @param ncols  the number of columns
 * @param row    the row index
 * @param col    pointer where the row that contains the minimum value will be
 *               put
 *
 * @return
 * minimum value of the specified row
 */
static double minimumr(double* matrix,int ncols,int row,int* col)
{
  int i;
  double res=DBL_MAX;
  for(i=0;i<ncols;i++)
  {
    if(matrix[POS(row,i,ncols)]<res)
    {
      res=matrix[POS(row,i,ncols)];
      *col=i;
    }
  }
  return res;
}

//===============================================================================

int simplex(double* a,int n,int m,FILE* file)
{
  double aux;
  int pos=0,nm2,err=0;
  nm2=n+m+2;
  aux=minimumc(&a[nm2],m,nm2,n+m,&pos);
  if(file) fmprint(a,m+1,nm2,file);
  if(aux<0) err=simplexd(a,n,m,pos+1,file);
  if(!err)
  {
    aux=minimumr(a,nm2,0,&pos);
    if(aux<0) err=simplexp(a,n,m,pos,file);
  }
  return err;
}

//===============================================================================

int simplexp(double* a,int n,int m,int pos,FILE* file)
{
  int nm,pivotc,pivotr,i,j,k,result=0;
  double pivot,coef,aux,r;
  nm=n+m;
  k=nm+2;
  aux=-1;
  pivotc=pos;
  while(aux<0)
  {
    pivotr=-1;
    aux=DBL_MAX;
    for(i=1;i<m+1;i++)
    {
      if(a[POS(i,pivotc,k)]>0)
      {
        r=a[POS(i,nm,k)]/a[POS(i,pivotc,k)];
        if(r<aux)
        {
          aux=r;
          pivotr=i;
        }
      }
    }
    if(pivotr==-1) result=1;
    else
    {
      pivot=a[POS(pivotr,pivotc,k)];
      for(i=0;i<nm+1;i++)
      {
        a[POS(pivotr,i,k)]/=pivot;
      }
      for(i=1;i<m+1;i++)
      {
        if(i!=pivotr)
        {
          coef=-a[POS(i,pivotc,k)];
          for(j=0;j<nm+1;j++)
            a[POS(i,j,k)]+=coef*a[POS(pivotr,j,k)];
        }
      }
      aux=DBL_MAX;
      coef=-a[POS(0,pivotc,k)];
      a[POS(pivotr,nm+1,k)]=pivotc;
      for(i=0;i<nm+1;i++)
      {
        a[POS(0,i,k)]+=coef*a[POS(pivotr,i,k)];
        if(a[POS(0,i,k)]<aux)
        {
          aux=a[POS(0,i,k)];
          pivotc=i;
        }
      }
      if(file) fmprint(a,m+1,k,file);
    }
  }
  return result;
}

//================================================================================

int simplexd(double* a,int n,int m,int pos,FILE* file)
{
  int pivotc,pivotr,nm,i,j,k,result=0;
  double aux,pivot,r,coef;
  nm=n+m;
  k=nm+2;
  aux=-1;
  pivotr=pos;
  while(aux<0)
  {
    pivotc=-1;
    aux=DBL_MAX;

    for(i=0;i<nm;i++)
    {
      if(a[POS(pivotr,i,k)]<0)
      {
        r=a[POS(0,i,k)]/a[POS(pivotr,i,k)];
        if(fabs(r)<aux)
        {
          aux=r;
          pivotc=i;
        }
      }
    }
    if(pivotc==-1) result=1;
    else
    {
      pivot=a[POS(pivotr,pivotc,k)];
      for(i=0;i<nm+1;i++)
      {
        a[POS(pivotr,i,k)]/=pivot;
      }
      aux=a[POS(pivotr,nm,k)];
      for(i=0;i<m+1;i++)
      {
        if(i!=pivotr)
        {
          coef=a[POS(i,pivotc,k)]>0?a[POS(i,pivotc,k)]:-a[POS(i,pivotc,k)];
          for(j=0;j<nm+1;j++)
          {
            a[POS(i,j,k)]+=coef*a[POS(pivotr,j,k)];
          }
          if(a[POS(i,nm,k)]<aux)
          {
            aux=a[POS(i,nm,k)];
            pivotr=i;
          }
        }
      }
      a[POS(pivotr,nm+1,k)]=pivotc;
      if(file) fmprint(a,m+1,k,file);
    }
  }
  return result;
}
```

B Source Code Documentation

B.1 read.h File Reference

Definitions of data types and functions required to read and store the data about the problem.

Data Structures

- struct SExp

  Expression structure.
- struct SProb

  Problem structure.

Typedefs

- typedef SExp * Exp

  Expression definition.
- typedef SProb * Prob

  Problem definition.

Functions

- Exp newExp (int arraysize)

  Creates an expression.
- Prob newProb ()

  Creates a problem.
- int varhash (const char *varid)

  Hash function for variables names.
- int addCoefHead (Prob prob, const char *varid, int varc, double coef)

Adds a new condition to the problem, and adds the coefficients of a variable.

- int addCoefTail (Prob prob, const char *varid, int varc, double coef)

  Adds the coefficient of a variable.
- int setOpRHS (Prob prob, int op, double rhs)

  Adds the relational operator and the right-hand side to a condition.

B.1.1 Detailed Description

Definitions of data types and functions required to read and store the data about the problem.

Author

Rui Carlos Gonçalves

Version

1.5

Date

08/2015

Definition in file read.h.

B.1.2 Typedef Documentation

typedef SExp* Exp Expression definition.
Definition at line 33 of file read.h.

typedef SProb* Prob Problem definition.
Definition at line 51 of file read.h.

B.1.3 Function Documentation

int addCoefHead (Prob *prob,* **const char *** *varid,* **int** *varc,* **double** *coef* **)** Adds a new condition to the problem, and adds the coefficients of a variable.
If the variable did not exist in the variables set, it is added.
Parameters

prob	the problem.
varid	the variable identifier.
varc	the index of the variable if it does not exist.
coef	the coefficient of the variable.

Returns

0 if the variable already existed; 0 otherwise.

Definition at line 83 of file read.c.

int addCoefTail (Prob *prob,* **const char *** *varid,* **int** *varc,* **double** *coef* **)** Adds the coefficient of a variable.
If the variable did not exist in the variables set, it is added.

Parameters

prob	the problem.
varid	the variable identifier.
varc	the index of the variable if it does not exist.
coef	the coefficient of the variable.

Returns

0 if the variable already existed; 0 otherwise.

Definition at line 147 of file read.c.

Exp newExp (int *arraysize*) Creates an expression.
Parameters

arraysize	size of the coefficients array.

Returns

`NULL` if an error occurred; the new expression otherwise.

Definition at line 35 of file read.c.

Prob newProb ( ) Creates a problem.

Returns

`NULL` if an error occurred; the new problem otherwise.

Definition at line 53 of file read.c.

int setOpRHS (Prob *prob,* int *op,* double *rhs*) Adds the relational operator and the right-hand side to a condition.
Parameters

prob	the problem.
op	the identifier of the relational operator.
rhs	the right-hand side of the condition.

Returns

0.

Definition at line 201 of file read.c.

int varhash (const char ∗ *varid*) Hash function for variables names.
Parameters

varid	the variable identifier.

Returns

the hash code of the variable.

Definition at line 71 of file read.c.

B.2 read.c File Reference

Definitions of data types and functions required to read and store the data about the problem.

Macros

- #define POS(R, C, NC) ((R)*(NC)+(C))

  Given a row (`R`), a column (`C`), and the number of columns (`NC`) of a matrix, computes an equivalent 1D position.

Functions

- Exp newExp (int size)

  Creates an expression.
- Prob newProb ()

  Creates a problem.
- int varhash (const char *varid)

  Hash function for variables names.
- int addCoefHead (Prob prob, const char *varid, int varc, double coef)

  Adds a new condition to the problem, and adds the coefficients of a variable.
- int addCoefTail (Prob prob, const char *varid, int varc, double coef)

  Adds the coefficient of a variable.
- int setOpRHS (Prob prob, int op, double rhs)

  Adds the relational operator and the right-hand side to a condition.

Variables

- static char * errors [ ]

  Error messages.

B.2.1 Detailed Description

Definitions of data types and functions required to read and store the data about the problem.

Author

Rui Carlos Gonçalves

Version

1.5

Date

08/2015

Definition in file read.c.

B.2.2 Macro Definition Documentation

#define POS(*R*, *C*, *NC* **) ((R)*(NC)+(C))** Given a row (`R`), a column (`C`), and the number of columns (`NC`) of a matrix, computes an equivalent 1D position.

Definition at line 19 of file read.c.

B.2.3 Function Documentation

int addCoefHead (Prob *prob*, **const char *** *varid*, **int** *varc*, **double** *coef* **)** Adds a new condition to the problem, and adds the coefficients of a variable.

If the variable did not exist in the variables set, it is added.

Parameters

prob	the problem.
varid	the variable identifier.
varc	the index of the variable if it does not exist.
coef	the coefficient of the variable.

Returns

0 if the variable already existed; 0 otherwise.

Definition at line 83 of file read.c.

int addCoefTail (Prob ***prob,*** **const char** ∗ ***varid,*** **int** ***varc,*** **double** ***coef*** **)** Adds the coefficient of a variable.

If the variable did not exist in the variables set, it is added.

Parameters

prob	the problem.
varid	the variable identifier.
varc	the index of the variable if it does not exist.
coef	the coefficient of the variable.

Returns

0 if the variable already existed; 0 otherwise.

Definition at line 147 of file read.c.

Exp newExp (int ***arraysize*** **)** Creates an expression.

Parameters

arraysize	size of the coefficients array.

Returns

`NULL` if an error occurred; the new expression otherwise.

Definition at line 35 of file read.c.

Prob newProb ( ) Creates a problem.

Returns

`NULL` if an error occurred; the new problem otherwise.

Definition at line 53 of file read.c.

int setOpRHS (Prob ***prob,*** **int** ***op,*** **double** ***rhs*** **)** Adds the relational operator and the right-hand side to a condition.

Parameters

prob	the problem.
op	the identifier of the relational operator.

rhs	the right-hand side of the condition.

Returns

0.

Definition at line 201 of file read.c.

int varhash (const char * *varid*) Hash function for variables names.

Parameters

varid	the variable identifier.

Returns

the hash code of the variable.

Definition at line 71 of file read.c.

B.2.4 Variable Documentation

char* errors[ ] `[static]` Initial value:

```
={"ERROR! Function \'setCoefHead\' -> \'malloc\'.\n"
                     ,"ERROR! Function \'setCoefHead\' -> \'listInsertLst\'.\n"
                     ,"ERROR! Function \'setCoefHead\' -> \'arrayInsert\'.\n"
                     ,"ERROR! Function \'setCoefHead\' -> \'newExp\'.\n"
                     ,"ERROR! Function \'setCoefHead\' -> \'hashInsert\'.\n"
                     ,"ERROR! Function \'setCoefTail\' -> \'malloc\'.\n"
                     ,"ERROR! Function \'setCoefTail\' -> \'hashInsert\' or"
                     "\'arrayInsert\'.\n"
                     ,"ERROR! Function \'setCoefTail\' -> \'arrayInsert\'.\n"
                     }
```

Error messages.
Definition at line 22 of file read.c.

B.3 main.h File Reference

Definition of functions used by main.

Functions

- double * loadMatrix (Prob prob, int varc, int condc, int eqc, int type)
  From the info collected by the parser, creates a matrix representing the problem to solve, according to the format required by function `simplex`*.*
- void printRes (Array vars, double *tab, const char *varob, int varc, int condc, int type)
  Prints the result.
- int opt (int argc, char **argv, char **in, char **out)
  Checks the options specified by the user.

B.3.1 Detailed Description

Definition of functions used by main.

Author

Rui Carlos Gonçalves

Version

1.5

Date

08/2015

Definition in file main.h.

B.3.2 Function Documentation

double* loadMatrix (Prob *prob,* int *varc,* int *condc,* int *eqc,* int *type*) From the info collected by the parser, creates a matrix representing the problem to solve, according to the format required by function `simplex`.

Parameters

prob	info about the problem.
varc	number of variables.
condc	number of conditions.
eqc	number of conditions with an equality operator.
type	determines whether we have a maximization or minimization problem (max.: -1, min.: 1).

Returns

the matrix created.

Definition at line 30 of file main.c.

int opt (int *argc,* char ** *argv,* char ** *in,* char ** *out*) Checks the options specified by the user.

Parameters

argc	number of options.
argv	value of the options.
in	address for the input option.
out	address for the output option.

Returns

1 if the `tables` option was set; 0 otherwise.

Definition at line 155 of file main.c.

void printRes (Array *vars,* double * *tab,* const char * *varob,* int *varc,* int *condc,* int *type*) Prints the result.

Parameters

vars	name of the variables of each position of the table.
tab	table resulting from applying the *Simplex* algorithm.
varob	variable to maximize/minimize.
varc	number of variables.
condc	number of conditions

type	determines whether we have a maximization or minimization problem (max.: -1, min.: 1).

Definition at line 103 of file main.c.

B.4 main.c File Reference

Definition of functions used by main.

Macros

- #define POS(R, C, NC) ((R)*(NC)+(C))
  Given a row (`R`), a column (`C`), and the number of columns (`NC`) of a matrix, computes an equivalent 1D position.

Functions

- double * loadMatrix (Prob prob, int varc, int condc, int eqc, int type)
  From the info collected by the parser, creates a matrix representing the problem to solve, according to the format required by function `simplex`.
- void printRes (Array vars, double *tab, const char *varob, int varc, int condc, int type)
  Prints the result.
- int opt (int argc, char **argv, char **in, char **out)
  Checks the options specified by the user.

Variables

- static char * errors [ ]
  Error messages.

B.4.1 Detailed Description

Definition of functions used by main.

Author

Rui Carlos Gonçalves

Version

1.5

Date

08/2015

Definition in file main.c.

B.4.2 Macro Definition Documentation

#define POS(***R, C, NC*** **) ((R)*(NC)+(C))** Given a row (`R`), a column (`C`), and the number of columns (`NC`) of a matrix, computes an equivalent 1D position.

Definition at line 18 of file main.c.

B.4.3 Function Documentation

double* loadMatrix (Prob ***prob,*** **int** ***varc,*** **int** ***condc,*** **int** ***eqc,*** **int** ***type*** **)** From the info collected by the parser, creates a matrix representing the problem to solve, according to the format required by function `simplex`.

Parameters

prob	info about the problem.
varc	number of variables.
condc	number of conditions.
eqc	number of conditions with an equality operator.
type	determines whether we have a maximization or minimization problem (max.: -1, min.: 1).

Returns

the matrix created.

Definition at line 30 of file main.c.

int opt (int ***argc,*** **char** ** ***argv,*** **char** ** ***in,*** **char** ** ***out*** **)** Checks the options specified by the user.

Parameters

argc	number of options.
argv	value of the options.
in	address for the input option.
out	address for the output option.

Returns

1 if the `tables` option was set; 0 otherwise.

Definition at line 155 of file main.c.

void printRes (Array ***vars,*** **double** * ***tab,*** **const char** * ***varob,*** **int** ***varc,*** **int** ***condc,*** **int** ***type*** **)** Prints the result.

Parameters

vars	name of the variables of each position of the table.
tab	table resulting from applying the *Simplex* algorithm.
varob	variable to maximize/minimize.
varc	number of variables.
condc	number of conditions
type	determines whether we have a maximization or minimization problem (max.: -1, min.: 1).

Definition at line 103 of file main.c.

B.4.4 Variable Documentation

char* errors[ ] `[static]` **Initial value:**

```
={"ERROR! Function \’loadMatrix\’ -> \’calloc\’.\n"
                      ,"ERROR! Function \’printRes\’ -> \’newArray\’.\n"
                      ,"ERROR! Function \’printRes\’ -> \’malloc\’.\n"
                      ,"ERROR! Function \’printRes\’ -> \’arrayInsert\’.\n"
                      ,"ERROR! Invalid parameters.\n"
                      }
```

Error messages.
Definition at line 21 of file main.c.

B.5 rlp.h File Reference

Implementation of linear programming functions.

Functions

- int simplex (double *a, int n, int m, FILE *file)
  Applies the Simplex Algorithm to an optimization problem.
- int simplexp (double *a, int n, int m, int pos, FILE *file)
  Applies the Simplex algorithm to a primal optimization problem.
- int simplexd (double *a, int n, int m, int pos, FILE *file)
  Applies the Simplex algorithm to a dual optimization problem.

B.5.1 Detailed Description

Implementation of linear programming functions.

Author

Rui Carlos Gonçalves

Version

3.0

Date

07/2012

Definition in file rlp.h.

B.5.2 Function Documentation

int simplex (double * *a*, int *n*, int *m*, FILE * *file*) Applies the *Simplex Algorithm* to an optimization problem.

Given a problem with n variables (x_1 , ..., x_n) and m conditions ($c_1 = b_1$, ..., $c_m = b_m$), the function must receive a matrix a of size *(m+1)*(n+m+2)*, containing:

- at `a[0][i-1]` (for i in *[1, n]*) the coefficient of variable x_i in the expression to minimize;
- at `a[0][i]` (for i in *[n, n+m+1]*) the value 0;
- at `a[i][j-1]` (for i in *[1, m]*, and for j in *[1, n]*) the coefficient of variable x_j in condition c_i ;
- at `a[i][j]` (for i in *[1, m]*, and for j in *[n, n+m-1]*) the identity matrix;
- at `a[i][n+m]` (for i in *[1, m]*) the value of b_i ;
- at `a[i][n+m+1]` (for i in *[1, m]*) the value of $n+i$.

Allows to specify the file where the table resulting from the application of the algorithm (using the parameter *file*).

Parameters

a	matrix that represents the problem
n	number of variable of the function
m	number restrictions
file	file where the tables will be saved (or `NULL`)

Returns

0 if it is possible to solve the problem
1 otherwise

Definition at line 118 of file rlp.c.

int simplexd (double ∗ *a*, **int** *n*, **int** *m*, **int** *pos*, **FILE** ∗ *file* **)** Applies the *Simplex algorithm* to a dual optimization problem.

The input matrix (`a`) must follow the format defined in function `simplex`.

Allows to specify the file where the table resulting from the application of the algorithm (using the parameter *file*).

Parameters

a	matrix that represents the problem
n	number of variables of the function
m	number restrictions
pos	position of the minimum value of the first row (it must be a negative value)
file	file where the tables will be saved (or `NULL`)

Returns

0 if it is possible to solve the problem
1 otherwise

Definition at line 197 of file rlp.c.

int simplexp (double ∗ *a*, **int** *n*, **int** *m*, **int** *pos*, **FILE** ∗ *file* **)** Applies the *Simplex algorithm* to a primal optimization problem.

The input matrix (`a`) must follow the format defined in function `simplex`.

Allows to specify the file where the table resulting from the application of the algorithm (using the parameter *file*).

Parameters

a	matrix that represents the problem
n	number of variables of the function
m	number restrictions
pos	position of the minimum value of the restrictions' column (it must be a negative value)
file	file where the tables will be saved (or `NULL`)

Returns

0 if it is possible to solve the problem
1 otherwise

Definition at line 136 of file rlp.c.

B.6 rlp.c File Reference

Implementation of linear programming functions.

Macros

- #define POS(R, C, NC) ((R)∗(NC)+(C))
  Given a row (`R`), a column (`C`), and the number of columns (`NC`) of a matrix, computes an equivalent 1D position.

Functions

- static void fmprint (double ∗matrix, int nrows, int ncols, FILE ∗file)
  Prints a matrix associated to an optimization problem.
- static double minimumc (double ∗matrix, int nrows, int ncols, int col, int ∗row)
  Finds the minimum value of a matrix's column.
- static double minimumr (double ∗matrix, int ncols, int row, int ∗col)

Finds the minimum value of a matrix's row.

- int simplex (double *a, int n, int m, FILE *file)

  Applies the Simplex Algorithm to an optimization problem.
- int simplexp (double *a, int n, int m, int pos, FILE *file)

  Applies the Simplex algorithm to a primal optimization problem.
- int simplexd (double *a, int n, int m, int pos, FILE *file)

  Applies the Simplex algorithm to a dual optimization problem.

B.6.1 Detailed Description

Implementation of linear programming functions.

Author

Rui Carlos Gonçalves

Version

3.0

Date

07/2012

Definition in file rlp.c.

B.6.2 Macro Definition Documentation

#define POS(***R, C, NC*** **) ((R)*(NC)+(C))** Given a row (`R`), a column (`C`), and the number of columns (`NC`) of a matrix, computes an equivalent 1D position.

Definition at line 18 of file rlp.c.

B.6.3 Function Documentation

static void fmprint (double * ***matrix,*** **int** ***nrows,*** **int** ***ncols,*** **FILE** * ***file*** **)** **`[static]`** Prints a matrix associated to an optimization problem.

The file `file`, where the matrix will be printed, must be opened before calling this function.

Parameters

matrix	the matrix to print
nrows	the number of rows
ncols	the number of columns
file	where the tables will be printed

Definition at line 31 of file rlp.c.

static double minimumc (double * ***matrix,*** **int** ***nrows,*** **int** ***ncols,*** **int** ***col,*** **int** * ***row*** **)** **`[static]`** Finds the minimum value of a matrix's column.

Parameters

matrix	the matrix
nrows	the number of rows
ncols	the number of columns

col	the column index
row	pointer where the row that contains the minimum value will be put

Returns

minimum value of the specified column

Definition at line 72 of file rlp.c.

static double minimumr (double ∗ *matrix*, int *ncols*, int *row*, int ∗ *col*) `[static]` Finds the minimum value of a matrix's row.

Parameters

matrix	the matrix row
ncols	the number of columns
row	the row index
col	pointer where the row that contains the minimum value will be put

Returns

minimum value of the specified row

Definition at line 101 of file rlp.c.

int simplex (double ∗ *a*, int *n*, int *m*, FILE ∗ *file*) Applies the *Simplex Algorithm* to an optimization problem.

Given a problem with *n* variables (x_1 , ..., x_n) and *m* conditions ($c_1 = b_1$, ..., $c_m = b_m$), the function must receive a matrix a of size *(m+1)*(n+m+2)*, containing:

- at `a[0][i-1]` (for *i* in *[1, n]*) the coefficient of variable x_i in the expression to minimize;
- at `a[0][i]` (for *i* in *[n, n+m+1]*) the value 0;
- at `a[i][j-1]` (for *i* in *[1, m]*, and for *j* in *[1, n]*) the coefficient of variable x_j in condition c_i ;
- at `a[i][j]` (for *i* in *[1, m]*, and for *j* in *[n, n+m-1]*) the identity matrix;
- at `a[i][n+m]` (for *i* in *[1, m]*) the value of b_i ;
- at `a[i][n+m+1]` (for *i* in *[1, m]*) the value of *n+i*.

Allows to specify the file where the table resulting from the application of the algorithm (using the parameter *file*).

Parameters

a	matrix that represents the problem
n	number of variable of the function
m	number restrictions
file	file where the tables will be saved (or `NULL`)

Returns

0 if it is possible to solve the problem
1 otherwise

Definition at line 118 of file rlp.c.

int simplexd (double ∗ *a*, int *n*, int *m*, int *pos*, FILE ∗ *file*) Applies the *Simplex algorithm* to a dual optimization problem.

The input matrix (`a`) must follow the format defined in function `simplex`.

Allows to specify the file where the table resulting from the application of the algorithm (using the parameter *file*).

Parameters

a	matrix that represents the problem
n	number of variables of the function
m	number restrictions
pos	position of the minimum value of the first row (it must be a negative value)
file	file where the tables will be saved (or `NULL`)

Returns

0 if it is possible to solve the problem
1 otherwise

Definition at line 197 of file rlp.c.

int simplexp (double ∗ ***a*, int *n*, int *m*, int *pos*, FILE** ∗ ***file*)** Applies the *Simplex algorithm* to a primal optimization problem.

The input matrix (`a`) must follow the format defined in function `simplex`.

Allows to specify the file where the table resulting from the application of the algorithm (using the parameter *file*).

Parameters

a	matrix that represents the problem
n	number of variables of the function
m	number restrictions
pos	position of the minimum value of the restrictions' column (it must be a negative value)
file	file where the tables will be saved (or `NULL`)

Returns

0 if it is possible to solve the problem
1 otherwise

Definition at line 136 of file rlp.c.