UNIVERSIDADE DO PORTO

FACULDADE DE ENGENHARIA

D. E. E. C.

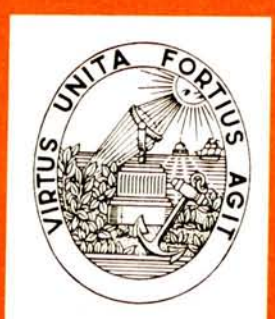
Estudo do problema de fluxo de potência Óptimo Impreciso:

- Adição de restrições
- Agregação de resultados parciais

Relatório de Estágio

Susana Maria Afonso Chaves

1992/93



REFERÊNCIA DO PROJECTO 4.3/1.117/92/93

Parecer Técnico

Acompanhei os trabalhos de estágio realizados no INESC pelo aluno e apreciei o relatório final apresentado, considerando que aquele relatório reflecte o trabalho efectivamente realizado.

Sou de parecer que foram atingidos os objectivos inicialmente propostos no plano de trabalhos.

O Supervisor do Estágio (INESC)

PRODEP

Parecer Científico sobre o estágio de

Susana Maria Afonso Chaves

A estagiária realizou sob minha orientação um trabalho de estágio sobre o tema que lhe foi fixado, tendo sido de uma forma geral alcançados os objectivos científicos inicialmente propostos.

A orientação científica incidiu sobre a definição de objectivos, fornecimento de bibliografia ou acesso à mesma e verificação dos progressos conseguidos.

Esta orientação foi exercida em coordenação com o supervisor da estagiária, na instituição onde fisicamente foi realizada a parte substancial dos trabalhos.

O nível de qualidade do trabalho executado, que se pode deduzir do relatório elaborado pela estagiária, é bom, ao nível do grau de formação académica da estagiária. O relatório resume e descreve, de forma correcta, as fases dos estudos realizados e dos desenvolvimentos algorítmicos propostos e implementados.

Manuel António Matos

Professor Associado da FEUP/DEEC

AGRADECIMENTOS

Dou o meu agradecimento ao INESC, por me ter concedido as instalações e o equipamento necessário ao desenvolvimento do estágio.

Agradeço, também, à FEUP, aos Profs. Vladimiro Miranda e João Paulo Saraiva o tempo dispensado na orientação do trabalho proposto, bem como ao PRODEP pelo apoio concedido e necessário ao bom funcionamento do estágio.

ÍNDICE

1. Aspecto geral do trabalho proposto

O estágio frequentado, destina-se a qualquer planeamento de um problema físico que possa ser traduzido por um Modelo Linear. Neste caso, serve para determinar, qual a melhor decisão a tomar no problema de *FLUXO DE POTÊNCIA ÓPTIMO IMPRECISO*. Esta decisão vai indicar qual a melhor produção face a consumos exígidos e que ao mesmo tempo minimizam o custo.

Neste planeamento, idealizamos um Modelo Linear, que será o mais aproximado possível da realidade. Ao tomar este modelo como ideal, corremos o risco de falhar a longo prazo. Então, a solução é dar ao Modelo uma certa *liberdade*, para que possa tomar a melhor decisão. Essa liberdade é conseguida com a introdução de incertezas nas cargas (e gerações) do sistema eléctrico. O efeito destas incertezas reflete-se sobre a capacidade do sistema cumprir o objectivo básico de alimentar os clientes, ou melhor, de oferecer garantia de serviço face a um conjunto alargado de cenários de carga, possíveis no futuro.

A incerteza nas cargas de um sistema é modelizada por conjuntos imprecisos (fuzzy sets), o que permite obter o Modelo de Fluxo de Potência Óptimo Impreciso.

Este Modelo permite de acordo com um critério económico, identificar possibilidades de potências geradas, trânsitos de potência e potência de corte de carga.

A aliciante destes modelos de planeamento é que são Modelos Lineares, e algoritmos como o Simplex podem ser aplicados. Mas o Simplex apenas dá a solução óptima de um problema determinístico. A introdução de imprecisões torna o problema não determinístico, o que requer técnicas de programação essenciais ao tratamento das imprecisões. O meu estágio consistiu no tratamento algorítmico para o problema não determinístico.

2.

1ª Fase

Esta envolveu o estudo de vários conceitos necessários a uma melhor compreensão do trabalho proposto, foi também necessária uma adaptação ao estilo de programação exigido. Esta fase teve vários pontos importantes a considerar:

2.1. Métodos de resolução de equações lineares

Estudei alguns métodos usados na resolução de conjuntos de equações lineares. Estes dividem-se em duas categorias:

- *Métodos directos*, são baseados na manipulação directa das equações, a solução é obtida num número finito de passos e é exacta. Na prática, para problemas de grande dimensão este método requer grande espaço de memória e tempos de execução longos.
- *Métodos iterativos*, são baseados na resolução das equações por sucessivas aproximações até o resultado ser aceitável, isto é, ter um erro menor do que um fixado à partida. Este método requer habitualmente de pouca memória e pouco tempo de execução, pois só necessita de guardar os valores da matriz original.

Contudo os métodos directos têm uma aplicação mais geral, é possível determinar a inversa de uma matriz de forma explícita ou factorizada num nº finito de operações aritméticas. A grande vantagem é utilizar este método juntamente com técnicas de esparsidade, conseguindo-se assim reduzir o espaço de memória gasto, tempo de execução e o número de cálculos, tornando-se os algoritmos mais eficientes em termos de espaço e tempo. Esta técnica é bastante eficiente para matrizes de grande dimensão. Estudos realizados indicam que a dimensão aceitável é a partir de 30.

Estudei também vários métodos de factorização de matrizes, a factorização à esquerda, à direita e também a bifactorização.

2.2. Estudo de técnicas de esparsidade aplicadas à resolução de problemas lineares

Para obter a solução de um problema linear é necessário o cálculo da inversa da matriz. O problema é da seguinte forma

$$\min \ C^t X$$
$$\text{s. a. } AX=b,$$

e a solução é dada por $X=A^{-1}b$.

A matriz que representa o problema do Fluxo de Potência Óptimo tem determinadas características que tornam uns métodos mais eficientes que outros, dessas

características ressaltam a sua dimensão (geralmente muito elevada), a elevada quantidade de elementos nulos.

Implementei o método de factorização à esquerda, para obter a inversa, usando conjuntamente técnicas de esparsidade. Este método consiste em construir uma sequência de n matrizes factores T_j, $1 \leq j \leq n$ (onde n é a dimensão da matriz), tais que :

$$A^{-1}=T_n T_{n-1} \ldots T_2 T_1$$

Estas matrizes T_j, chamadas matrizes de transformação, apenas diferem da matriz identidade numa coluna.

A grande vantagem em usar este método de factorização, juntamente com as técnicas de esparsidade, é a redução do espaço de memória gasto e do tempo de execução, tornando assim o algoritmo mais eficiente. Estas técnicas são bastante eficientes para matrizes de grande dimensão.

Esta 1ª fase incluiu, a compreensão da formulação do *Problema de Fluxo de Potências Óptimo Determinístico Inicial*, e o estudo de conceitos da teoria dos *Fuzzy-Sets*.

3.

2ª Fase

Esta fase envolveu a parte de programação propriamente dita, a qual se divide em três módulos:

3.1 Desenvolvimento de um módulo de extensão a um núcleo de simplex para adição de restrições e recálculo da solução óptima pelo dual simplex

A formulação do problema determinístico inicial, integra a equação de equilíbrio de potências activas, restrições associadas aos limites mínimos e máximos das potências produzidas e dos trânsitos de potências nos ramos, e restrições correspondentes ao valor máximo de potência de corte de carga em cada barramento. Para um problema real com um número considerável de geradores e linhas, o nº de restrições é muito elevado. O problema inicial, não entra em linha de conta com todas as restrições, sendo necessário ao longo da resolução do problema, introduzir novas restrições, no caso destas, não satisfizerem a solução inicial.

3.2. Desenvolvimento de um módulo de agregação de resultados parciais de um problema paramétrico

A resolução de cada problema de programação paramétrica, permite obter, para cada variável, um conjunto de valores óptimos. Verifica-se, além disso, que a cada elemento de um desses conjuntos está associado um grau de pertença correspondente ao valor do parâmetro d. As soluções obtidas, para cada variável, nos diversos estudos parametrizados de fluxo de potências óptimo, são agregados utilizando o operador Reunião.

Procedendo deste modo, foi possível encontrar um conjunto de valores óptimos para cada variável, a esse conjunto de valores está associado o grau de credibilidade que varia entre 1 e zero.

3.3.Estratégias de reforço da capacidade de componentes do sistema

Este módulo permite possíveis alterações no sistema; alterar capacidades de geração e limite de linhas. Pode ainda ser adoptado um novo modelo de fluxo de potência óptimo, derivado do anterior, em que são adicionadas novas variáveis, representando os possíveis incrementos de capacidade de elementos do sistema. É construida uma nova função objectivo, com base nestas variáveis, tendo por coeficientes de custo os valores marginais de investimento associados a cada elemento reforçável.

4.Considerações sobre a utilidade do estágio

A minha opinião sobre o estágio que frequentei, é que ele vai ser muito útil num futuro emprego, pois contribuiu bastante para minha adaptação e integração em equipas de investigação e desenvolvimento de software.

5.Bibliografia

- **"Sparsity - Its pratical application to systems analysis"**
  A. Brameller, R. N. Allan
- **"Fuzzy Set Teory - and its applications"**
  H. J. Zimmermann, Kluwer-Nijhoff Publishing, Boston, 1985
- **"Fuzzy Set And System"**
  Didler Dubois e Henry Prade
- "**An Improved Fuzzy Optimal Load Flow Model for Operation Analysis**"
  J.Tomé Saraiva, Vladimiro Miranda e L.M.V.G.Pinto - IEEE
- "**Fuzzy Flows in Linear Networks**"
  Vladimiro Miranda, Manuel A.C.C. Matos e J. Tomé Saraiva - IEEE
-"**Índices de Risco e Impacto de Investimentos Avaliados por um Modelo Integrando Incertezas de Natureza Possibilística**"
  J.Tomé Saraiva, Vladimiro Miranda

APÊNDICE

LISTAGENS DE PROGRAMAS

```
#ifndef COMMON1_H
#define COMMON1_H

#ifdef __STDC__
#define ANSI_C
#endif

#define ANSI_C

#define OK              0
#define SORRY           1
#define DOSOPERR        2  // file open error
#define ARGERROR        3  // argument error
#define SOFTBUG         4
#define OUTOFMEM        5  // out of memory

#define RADEBUG_MEMORY  8
#define RADEBUG_NOFREE  16

#define RADEBUG
#ifdef RADEBUG
extern int radebug;
#endif

#define FALSE   0
#define TRUE    1

void FatalErr(int,char *);

#endif
```

```
#define    NO_OBJECT     100
#define    FILE_ERROR    110
#define    SIGN_ERROR    120
```

```
struct tab1 {
  double value;
  int irow;
  struct tab1 *next;
  };

struct tabx {
  double value;
  int itok;
  struct tabx *next;
  };

struct aa{
  struct tab1 *icapa;
  int nozea;
  };

struct aax{
  struct tabx *icapx;
  int nlx;
  };

struct tt{
  struct tab1 *icapt;
  int nozet;
  int ncolt;
  int entra;
  struct tt *next;
  struct tt *last;
  };

struct matriz{
  struct aa *mata;
  struct aax *matx;
  int nl;
  int nlx;
  int nci;
  int ncix;
  double *b;
  double *bx;
  double *c;
  int *xb;
  double *ca;
  char *c1;
  };

typedef struct tt2 {
  int restr;
  int ivar;
  struct tt2 *next;
  } TT2 ;


void copia_vector(double *,double *,int);
void procura(struct matriz *,char *,double *,double *);
char *ver_restr(struct matriz *,int *);
void fase_um(struct matriz *,char *,int);
void fase_dois(struct matriz *,char *,int);
void simplex(struct matriz *);

void *mymalloc(int);
void *myrealloc(void *,int);
void myfree(void *);
void liberta_matriz(struct matriz *);

int Le_palavra(FILE *,char *);
```

```
int Le_palavra2(FILE *,char *);
double Le_num(FILE *);
void Filtragem(void);
void Le_restricoes(FILE *);
struct matriz *lindo_fich(char *);

struct matriz *linear_fich(char *);

void des_mata(struct matriz *,int);
void des_matt(void);
```

```
#include<stdio.h>
#include<malloc.h>
#include<process.h>
#include<stdlib.h>

#include"common.h"
#include"simplex.h"

/* Aloca espao e faz testes
 mem¢ria */
void *mymalloc(size)
int size;
{
  void *ap=malloc(size);

  if(radebug & RADEBUG_MEMORY){
    fprintf(stderr,"%ap=mymalloc(%d)\n",ap,size);
    if(ap==NULL)
      fprintf(stderr,"\nWARNING: NULL POINTER !!!!!!!!\n");
    }
  return(ap);
}


/* Aloca espao e faz testes
 mem¢ria */
void *myrealloc(p,size)
void *p;
int size;
{
  void *ap=realloc(p,size);

  if(radebug & RADEBUG_MEMORY){
    fprintf(stderr,"%ap=myrealloc(%d)\n",ap,size);
    if(ap==NULL)
      fprintf(stderr,"\nWARNING: NULL POINTER !!!!!!!!\n");
    }
  return(ap);
}


void myfree(p)
void *p;
{
  int DummyFree;
  if(p==NULL)
    FatalErr(SOFTBUG,"Attempt to free a NULL pointer");
  if(!(radebug & RADEBUG_NOFREE)){
    DummyFree=0;
    free(p);
    }
  else
    DummyFree=1;
  if(radebug & RADEBUG_MEMORY){
    if(DummyFree)
      fprintf(stderr,"Free disabled, it would be: ");
    fprintf(stderr,"myfree(%p)\n",p);
    }
}

void FatalErr(errorcode,errormessage)
int errorcode;
char *errormessage;
{
  fprintf(stderr,"%s\n",errormessage);
  exit(errorcode);
```

}

```
#include<stdio.h>
#include<string.h>
#include<process.h>
#include<stdlib.h>

#include"simplex.h"
#include"common.h"

#ifdef RADEBUG
int radebug=0;
#endif

extern int **interr; /*intervalos dos parametros*/
extern int *gamou; /*contem vertice a analisar*/
extern int tamx;
extern double *bsub;

extern int np; /* n. de parametros a introduzir*/
extern double **m_par; /*matriz dos coeficientes dos parametros*/

extern float **inter; /*intervalos dos parametros*/
extern float *gama;
extern float *g_aux;
extern float *Pmed;
extern float *dif;

extern double *baux;

struct matriz *matriz1=NULL;

int nci,nl,nlx,*xb=NULL;
double *b=NULL;
double *bx=NULL;
double *c=NULL;
extern char **tokens;
struct aa *mata=NULL;
struct aax *matx=NULL;
extern struct tt *matt;

double *cb=NULL;
double *ca=NULL;
char *c1=NULL;
double *sol=NULL;

void main (argc, argv)
    int argc;
    char *argv[];
{
int i;
char z[80];
char s[13];
char is[13];
struct tt *t,*t0;
struct tab1 *p0,*pp1;

if(argc==1){
  printf("Qual o ficheiro (LINDO) ?\n ");
  scanf("%s",s);
  }
else{
  if(argc==2)
    strcpy(s,argv[1]);
  else{
    printf("Usage : %s <nome de ficheiro>\n",argv[0]);
    exit(1);
    }
```

```
  }

printf("Vou abrir o ficheiro '%s'\n",s);

matriz1=lindo_fich(s);

simplex(matriz1);

mata=matriz1->mata;
matx=matriz1->matx;
nl=matriz1->nl;
nlx=matriz1->nlx;
nci=matriz1->nci;
b=matriz1->b;
bx=matriz1->bx;
xb=matriz1->xb;
ca=matriz1->ca;
c1=matriz1->c1;
c=matriz1->c;
/*
des_mata(matriz1,nci);
*/
des_matt();
/*
des_matx(matriz1,nlx-nl);
*/

sol=(double *)mymalloc(nci*sizeof(double));
for(i=0;i<nci;i++)
  sol[i]=0.;

for(i=0;i<nl;i++)
  if(xb[i]<nci)
    sol[xb[i]]=b[i];

printf("Solu
            o  :\n");
for(i=0;i<nci;i++)
  printf("%s = %f\n",tokens[i],sol[i]);

/*testa soluao nas restantes restrioes*/
if((nlx-nl)!=0)
  tes_sol(matriz1);


/************************************************/
printf("\nQuer introduzir parametros em b?(s/n) ");
scanf(" %s",z);
if(z[0]=='s' || z[0]=='S'){
  matrizi(matriz1);  /*EXPRESSOES LINEARES*/
  le_inter(matriz1); /*PROG. MULTIPARAMETRICA->ANALISA OS VERTICES*/
  le_pontos(matriz1); /*PROG. PARAMETRICA*/
  }


printf("Vou libertar tudo!\n");
fflush(stdout);


  t=matt;
  while(t!=NULL){
    p0=t->icapt;
    while(p0!=NULL){
      pp1=p0->next;
      myfree(p0);
      p0=pp1;
```

```
      }
    t0=t->next;
    myfree(t);
    t=t0;
    }
  if(matt!=NULL){
    myfree(matt);
    matt=NULL;
    }
  else
    FatalErr(SOFTBUG,"matt not allocated (simplex)");

if(sol!=NULL){
  myfree(sol);
  sol=NULL;
  }
else
  FatalErr(SOFTBUG,"sol not allocated (main)");

if(bsub!=NULL){
  myfree(bsub);
  bsub=NULL;
  }
else
  FatalErr(SOFTBUG,"bauxx not allocated (main)");

if(gamou!=NULL){
  myfree(gamou);
  gamou=NULL;
  }
else
  FatalErr(SOFTBUG,"gamou not allocated (main)");

if(Pmed!=NULL){
  myfree(Pmed);
  Pmed=NULL;
  }
else
  FatalErr(SOFTBUG,"Pmed not allocated (main)");

if(dif!=NULL){
  myfree(dif);
  dif=NULL;
  }
else
  FatalErr(SOFTBUG,"dif not allocated (main)");

if(baux!=NULL){
  myfree(baux);
  baux=NULL;
  }
else
  FatalErr(SOFTBUG,"baux not allocated (main)");

if(gama!=NULL){
  myfree(gama);
  gama=NULL;
  }
else
  FatalErr(SOFTBUG,"gama not allocated (main)");

if(g_aux!=NULL){
  myfree(g_aux);
  g_aux=NULL;
  }
else
```

```
  FatalErr(SOFTBUG,"g_aux not allocated (main)");

if(interr!=NULL){
  for(i=0;i<np;i++){
    if(interr[i]!=NULL)
      myfree(interr[i]);
    else
      FatalErr(SOFTBUG,"interr not allocated properly (main)");
    }
  myfree(interr);
  interr=NULL;
  }
else
  FatalErr(SOFTBUG,"interr not allocated (main)");

if(inter!=NULL){
  for(i=0;i<np;i++){
    if(inter[i]!=NULL)
      myfree(inter[i]);
    else
      FatalErr(SOFTBUG,"inter not allocated properly (main)");
    }
  myfree(inter);
  inter=NULL;
  }
else
  FatalErr(SOFTBUG,"inter not allocated (main)");

if(m_par!=NULL){
  for(i=0;i<np;i++){
    if(m_par[i]!=NULL)
      myfree(m_par[i]);
    else
      FatalErr(SOFTBUG,"m_par not allocated properly (main)");
    }
  myfree(m_par);
  m_par=NULL;
  }
else
  FatalErr(SOFTBUG,"m_par not allocated (main)");

if(matriz1!=NULL){
  liberta_matriz(matriz1);
  myfree(matriz1);
  matriz1=NULL;
  }
else
  FatalErr(SOFTBUG,"matriz1 not allocated (main)");

for(i=0;i<nci;i++){
  if(tokens[i]!=NULL)
    myfree(tokens[i]);
  else
    FatalErr(SOFTBUG,"tokens[i] not allocated (main)");
  }
if(tokens!=NULL){
  myfree(tokens);
  tokens=NULL;
  }
else
  FatalErr(SOFTBUG,"tokens not allocated (main)");

printf("Vou acabar!\n");
fflush(stdout);

}
```

```
#include<stdio.h>
#include<string.h>
#include<process.h>
#include<stdlib.h>

#include"simplex.h"
#include"common.h"

/* variaveis ja definidas */

extern int nl;
extern double *c;
extern int *xb;
extern int nci;
extern char *c1;
extern double *cb;
extern double *b;
extern double *ca;

extern struct aa *mata;
extern struct tt *matt,*last;

extern char **tokens;

int it;  /*n§ de iteraäes*/
double *b1;

/***********************************/
/***funÆo principal-> l novo b ***/
/***********************************/
novo_b()
{
int i,x;
for(i=0;i<nl;i++)
  printf("b_inicial[%d]=%lf\n",i,b1[i]);
printf("introduza o novo b\n");
for(i=0;i<nl;i++){
  printf("binicial[%d]=%lf ? ",i,b1[i]);
  scanf("%lf",&b1[i]);
  }
for(i=0;i<nl;i++)
  printf("b1[%d]=%lf\n",i,b1[i]);

nova_sol();

for(i=0;i<nl;i++)
  printf("b1_final[%d]=%lf\n",i,b1[i]);
it=0;
for(i=0;i<nl;i++)
  if(b1[i]<0.)x=i;
it=1;
it_dual(x);

printf("N§ de iteracoes pelo dual ?? %d\n",it);
}

/*****************************************/
/**** Vou calcular b'= Tr*...*T1*b1[] ****/
/*****************************************/
nova_sol()
{
int i,j;
double tot,tot1;
struct tab1 *p0;
struct tt *t,*last;
```

```
t=matt;
while(t!=NULL){      /* para cada uma das matrizes T's */
  tot=0.;
  p0=t->icapt;       /* para a ultima das matrizes T's*/
  while(p0!=NULL){   /* para cada um dos elementos existentes na coluna de T != d
    if(p0->irow==t->ncolt)
      tot=b1[p0->irow]*(p0->value);
    else
      b1[p0->irow]+=p0->value*b1[t->ncolt];
    p0=p0->next;
    }
  b1[t->ncolt]=tot;
  t=t->next;
  }
}

/*********************/
/*******it_dual*******/
/*********************/
it_dual(lpiv)
int lpiv;
{
  struct tab1 *p0;
  struct tt *t;
  int i,j;
  int imine;          /* indice do m¡nimo (entrar na base) */
  int imins;          /* indice do m¡nimo (sair da base) */
  double maxi;
  double *p=NULL;
  double tot;

  p=(double *)mymalloc(nl*sizeof(double));

  imins=lpiv;         /*indice da variave que sai b1[lpiv]<0.*/
  maxi=0;
  for(i=0;i<nci;i++){
/*    printf("c1[%d]=%d\n",i,c[i]);*/
    if(c1[i]==0){
      for(j=0;j<nl;j++)
        p[j]=0.;

      p0=mata[i].icapa;

      /*** Vou calcular o P(r) ***/
      while(p0!=NULL){
        p[p0->irow]=p0->value;
        p0=p0->next;
        }

      /*** Vou calcular P'(r) = B(-1)*P(r)  = Tr*...*T1*P(r) ***/
      t=matt;
      while(t!=NULL){    /* para cada uma das matrizes T's */
        tot=0.;
        p0=t->icapt;
        while(p0!=NULL){  /* para cada um dos elementos existentes na coluna de
          if(p0->irow==t->ncolt)
            tot=p[p0->irow]*p0->value;
          else
            p[p0->irow]+=p0->value*p[t->ncolt];
          p0=p0->next;
          }
        p[t->ncolt]=tot;
        t=t->next;
        }

      /*** vou achar a vari vel a entrar na base ***/
```

```
      if(maxi!=0.){
        if(p[imins]<0. && ca[i]/p[imins] > maxi){
          imine=i;
          maxi=ca[i]/p[imins];
          }
        }
      else
        if(p[imins]<0.){
          imine=i;
          maxi=ca[i]/p[imins];
          }

    }
  }
  if(maxi==0){
    printf("nao tem soluao! nÆo ha pivot\n");
    exit(0);
    }
  printf("sai %d\tentra %d\n",imins,imine);
  pivotagem(imins,imine);
}

/****************/
/*** pivotagem***/
/****************/
pivotagem(sai,entra)
int sai;
int entra;
{

  struct tab1 *p0,*p1,*fi; /* apontadores necess rios */
  struct tt *t;      /* apontadores necess rios */
  int i,j;
  double *p=NULL;  /* vector usado no algoritmo */
  double tot;

  p=(double *)mymalloc(nl*sizeof(double));
  cb=(double *)mymalloc(nl*sizeof(double));

  /*  ITERAAO  */
  /* Vou calcular o P(r) */
  for(i=0;i<nl;i++)
    p[i]=0.;
  p0=mata[entra].icapa;
  while(p0!=NULL){
    p[p0->irow]=p0->value;
    p0=p0->next;
    }

  /*** Vou calcular P'(r) = B(-1)*P(r)  = Tr*...*T1*P(r)   ***/
  t=matt;
  while(t!=NULL){    /* para cada uma das matrizes T's */
    tot=0.;
    p0=t->icapt;
    while(p0!=NULL){  /* para cada um dos elementos existentes na coluna de T !=
      if(p0->irow==t->ncolt)
        tot=p[p0->irow]*p0->value;
      else
        p[p0->irow]+=p0->value*p[t->ncolt];
      p0=p0->next;
      }
    p[t->ncolt]=tot;
    t=t->next;
    }

  /*** calculo da matriz de transforma
```

```
                                        o T(r) ***/
  j=0;     /* j - n§ de elementos != de zero */
  t=(struct tt *)mymalloc(sizeof(struct tt));

  if(last!=NULL){     /* ligaao aos elementos da lista */
    t->last=last;
    last->next=t;
    }
  else
    t->last=NULL;   /* se for o 1§ elemento da lista */
  t->next=NULL;
  fi=p1=(struct tab1 *)mymalloc(sizeof(struct tab1));
  for(i=0;i<nl;i++){
    if(p[i]){
      if(i==sai)
        p1->value=1/p[sai];
      else
        p1->value=-p[i]/p[sai];
      p1->irow=i;
      p0=p1;
      p1=(struct tab1 *)mymalloc(sizeof(struct tab1));
      p0->next=p1;
      j++;
      }
    }
  p0->next=NULL;
  t->icapt=fi;
  t->nozet=j;
  t->ncolt=sai;
  last=t;

  c1[xb[sai]]=0;     /* desmarco a vari vel que vai sair da base */
  c1[entra]=1;       /* marco vari vel que entra na base */
  xb[sai]=entra;     /* introduzo nova vari vel na base */

/*  for(i=0;i<nci;i++)
    printf(" c1[%d]=%d\n",i,c1[i]);*/

  /*** Vou calcular (actualizar) b'= B(-1)*b = T(r)*...*T(1)*b ***/
  tot=0.;
  p0=last->icapt;            /* para a ultima das matrizes T's */
  while(p0!=NULL){           /* para cada um dos elementos existentes na coluna d
    if(p0->irow==last->ncolt)
      tot=b1[p0->irow]*p0->value;
    else
      b1[p0->irow]+=p0->value*b1[last->ncolt];
    p0=p0->next;
    }
  b1[last->ncolt]=tot;

  for(i=0;i<nl;i++)
    printf("b1[%d]=%f\n",i,b1[i]);

  /***   vou inicializar Cb e Ca   ***/
  for(i=0;i<nl;i++){
    cb[i]=c[xb[i]];
    ca[i]=c[i];
    }
  for(;i<nci;i++)
    ca[i]=c[i];

  /*** vou calcular C'a = Ca - Cb*B(-1)*A'  ***/
       /* Cb*B(-1) ---> Cb */
  t=last;
  while(t!=NULL){
    tot=0.;
```

```
    p0=t->icapt;
    while(p0!=NULL){
      tot+=cb[p0->irow]*p0->value;
      p0=p0->next;
      }
    cb[t->ncolt]=tot;
    t=t->last;
    }
/*
  for(i=0;i<nl;i++)
    printf("CB[%d]=%f\n",i,cb[i]);
  printf("\n");
 */
       /* C'a = Ca - Cb*A' */
  for(i=0;i<nci;i++){
    while(c1[i])
      ca[i++]=0.;
    if(i<nci){
      p0=mata[i].icapa;
      while(p0!=NULL){
        ca[i]-=p0->value*cb[p0->irow];
        p0=p0->next;
        }
      }
    }

  for(i=0;i<nci;i++)
    printf("Zj-Cj[%d]=%f\n",i,ca[i]);

  for(i=0;i<nl && b1[i]>=0.; i++);
  if(i<nl){
    it_dual(i);
    it++;
    }
  else;

}
```

```
#include<stdio.h>
#include"simplex.h"

extern struct aa *mata1;
extern struct tt *matt;

/* Desenha a matriz com n§ de colunas indiçado */
void des_mata(matrix,tam)
struct matriz *matrix;
int tam;
{
struct aa *mata;
int nci;

int i;
struct tab1 *p;

mata=matrix->mata;
nci=matrix->nci;

printf("desenha §§ nci=%d\n",nci);
printf("desenha §§ tam=%d\n",tam);

printf("Matriz A :\n");
for(i=0;i<tam;i++){
  if(i<nci){
    printf("MATA\n");
    printf("\tIcapa = %ld\n",mata[i].icapa);
    printf("\tNozea = %d\n",mata[i].nozea);
    p=mata[i].icapa;
    }
  else{
    printf("MATA1\n");
    printf("\tIcapa = %ld\n",mata1[i-nci].icapa);
    printf("\tNozea = %d\n",mata1[i-nci].nozea);
    p=mata1[i-nci].icapa;
    }
  while(p!=NULL){
    printf("Endereo : %ld ** ",p);
    printf("Valuea - %lf , Irowa - %d ",p->value,p->irow);
    printf("++ Next : %ld \n",p->next);
    p=p->next;
    }
  }
printf("\n\n");
}


void des_matx(matrix,tam)
struct matriz *matrix;
int tam;
{
struct aax *matx;
int nci;

int i;
struct tabx *p;

matx=matrix->matx;
nci=matrix->nci;

printf("desenha §§ nci=%d\n",nci);
printf("desenha §§ tamx=%d\n",tam);

printf("Matriz X :\n");
for(i=0;i<tam;i++){
```

```
  printf("MATX\n");
  printf("\tIcapx = %ld\n",matx[i].icapx);
  printf("\tNLX = %d\n",matx[i].nlx);

  for(p=matx[i].icapx; p!=NULL; p=p->next){
    printf("Endereo : %ld ** ",p);
    printf("Value - %lf , Itok - %d ",p->value,p->itok);
    printf("++ Next : %ld \n",p->next);
    }
  }
printf("\n\n");
}


void des_matt()
{
int i;
struct tab1 *p;
struct tt *p1;
p1=matt;
printf("Matriz T :\n");
while(p1!=NULL){
  printf("\n Endereo %ld : \n",p1);
  printf("Icapt = %ld\n",p1->icapt);
  printf("Nozet = %d\n",p1->nozet);
  printf("Entra = %d\n",p1->entra);
  printf("Ncolt = %d\n",p1->ncolt);
  printf("Next = %ld\n",p1->next);
  printf("Last = %ld\n",p1->last);
  p=p1->icapt;
  while(p!=NULL){
    printf("Valuet - %lf , Irowt - %d\n",p->value,p->irow);
    p=p->next;
    }
  p1=p1->next;
  }
printf("\n\n");
}
```

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<io.h>
#include<errno.h>
#include<string.h>

#include"common.h"
#include"simplex.h"
#include"error.h"

#define MAXVARS      50  /* n§ m ximo de vari veis */
#define MAXVARS_R    50  /* n§ m ximo de vari veis por restri
                                                        o */
#define MAXSIZE      10  /* n§ m ximo de caracteres de cada vari vel */
#define MAXRESTR    100  /* n§ m ximo de restries */

#define END             -5
#define SUBJECT         -10

#define LESS_EQUAL       1
#define LESS             2
#define EQUAL            3
#define MORE             4
#define MORE_EQUAL       5

#define BLOCK       100  /* n§ de vari veis que aloca de uma s¢ vez */

#define MIN           0
#define MAX           1

/* declara
         o das estruturas que v
                              o manter a informa
                                                  o */

struct aa *mata;
struct aax *matx;

double *b;
double *bx;
double *c;


/* n§ linhas , n§ colunas , n§ de elementos nao nulos */
int nl;
int nlx;
int nc;
int nci;
int ncix;
int nn;
int tam;


/*********************************************************/
/************ vari veis internas deste m¢dulo ************/
/*********************************************************/
int varnum=0;
int varnumx=0;
char **tokens=NULL;
int nobj=0;
int nrestr=0;
int nresx=0;

char **vars=NULL;
double *coef=NULL;
```

```
int lvars;
double right;
int nvd=0;
int nrx=0;

int nmax1=MAXVARS;
int nmax2=MAXRESTR;
int count=1;
int count2=1;

/*******************************************************/
/************* L uma palavra do ficheiro *************/
/** se for um sinal de desigualdade l-o e retorna-o **/
/*******************************************************/
int Le_palavra(fp,string)
FILE *fp;
char *string;
{
  int nc=0,n;
  int ch;

  ch=getc(fp);
  while(isspace(ch) && ch!=EOF)
    ch=getc(fp);
  if(ch==EOF)
    return(-2);

  while(!isspace(ch) && nc<10){
    string[nc]=toupper(ch);
    ch=getc(fp);
    nc++;
    }
  string[nc]=0;
  n=strlen(string);

  if(!strcmp(string,"END"))
    return(END);

  if(n<3){
    switch(string[0]){
      case '=':
        if(n==1)
          return(EQUAL);
        else
          return(-1);
      case '<':
        if(n==1)
          return(LESS);
        else{
          if(string[1]=='=')
            return(LESS_EQUAL);
          else
            return(-1);
          }
      case '>':
        if(n==1)
          return(MORE);
        else{
          if(string[1]=='=')
            return(MORE_EQUAL);
          else
            return(-1);
          }
      default:
        return(0);
      }
```

```
    }
  return(0);
}


/*********************************************/
/******* L uma palavra do ficheiro *********/
/** se for a palavra chave SUBJECT retorna **/
/*********************************************/
int Le_palavra2(fp,string)
FILE *fp;
char *string;
{
  int nc=0;
  int ch;

  ch=getc(fp);
  while(isspace(ch) && ch!=EOF)
    ch=getc(fp);
  if(ch==EOF)
    return(-2);

  while(!isspace(ch) && nc<10){
    string[nc]=toupper(ch);
    ch=getc(fp);
    nc++;
    }
  string[nc]=0;

  if(strcmp(string,"SUBJECT")==0)
    return(SUBJECT);
  return(0);
}



/*************************************************/
/************* L um numero do ficheiro *************/
/************ se n
                o existir retorna 1. *************/
/*************************************************/
double Le_num(fp)
FILE *fp;
{
  int ch;
  double num=1.;
  double num1=1.;

  ch=getc(fp);
  while(isspace(ch) && ch!=EOF)
    ch=getc(fp);
  if(ch==EOF)
    return(-2);
  if(ch=='-' || ch=='+'){
    if(ch=='-')
      num1=num=-1.;
    else
      num1=num=1.;
    ch=getc(fp);
    while(isspace(ch) && ch!=EOF)
      ch=getc(fp);
    }
  if(isdigit(ch)){
    ungetc(ch,fp);
    fscanf(fp,"%lf",&num1);
    num1=num1*num;
```

```
    }
  else
    ungetc(ch,fp);

  return(num1);
}

/*******************************************************/
/************ Filtra a restri
                                o lida e **************/
/************ trata de a inserir na matriz ************/
/*******************************************************/
void Filtragem()
{
  int i,j;
  struct tab1 *p=NULL,*p0=NULL;
  int sign;
  char slack[10];
  char append[5];

  append[0]=0;

  if(strcmp(vars[lvars],">=")==0){
    for(i=0;i<lvars;i++)
      coef[i]=-coef[i];
    right=-right;
    strcpy(vars[lvars],"<=");
    }

  if(strcmp(vars[lvars],"<=")==0){
    strcpy(slack,"S_");
    itoa(nvd++,append,10);
    strcat(slack,append);
    strcpy(vars[lvars],slack);
    coef[lvars]=1.;
    lvars++;
    }

  if(right<0){
    for(i=0;i<lvars;i++)
      coef[i]=-coef[i];
    right=-right;
    }

  for(i=0;i<lvars;i++){
    p=(struct tab1 *)mymalloc(sizeof(struct tab1));
    p->irow=nrestr;
    p->value=coef[i];
    p->next=NULL;
    nn++;
    b[nrestr]=right;
    for(j=0;j<varnum;j++)   /*vou ver se  uma das vari veis j  existentes*/
      if(!strcmp(vars[i],tokens[j]))break;
    if(j==varnum){  /*se ainda n
                                o existe*/
      varnum++;
      tam++;
      if(varnum==nmax1){  /*se cheguei ao limite da mem¢ria aloco mais*/
        count++;
        nmax1+=MAXVARS;
        tokens=(char **)myrealloc((char *)tokens,nmax1*sizeof(char *));
        mata=(struct aa *)myrealloc((char *)mata,nmax1*sizeof(struct aa));
        c=(double *)myrealloc((char *)c,nmax1*sizeof(double));
        }
      c[j]=0.;
```

```
      mata[j].icapa=p;
      mata[j].nozea=1;
      tokens[j]=(char *)mymalloc((strlen(vars[i])+1)*sizeof(char));
      strcpy(tokens[j],vars[i]);
      }
    else{
      if(mata[j].icapa==NULL){
        mata[j].icapa=p;
        mata[j].nozea=1;
        }
      else{
        p0=mata[j].icapa;
        while(p0->next!=NULL)
          p0=p0->next;
        p0->next=p;
        mata[j].nozea++;
        }
      }
    }
}


void outras()
{
  int i,j;
  struct tabx *y=NULL;
  int sign;
  char slack[10];
  char append[5];

  append[0]=0;

  if(strcmp(vars[lvars],">=")==0){
    for(i=0;i<lvars;i++)
      coef[i]=-coef[i];
    right=-right;
    strcpy(vars[lvars],"<=");
    }

  if(strcmp(vars[lvars],"<=")==0){
    strcpy(slack,"S_");
    itoa(nvd++,append,10);
    strcat(slack,append);
    strcpy(vars[lvars],slack);
    coef[lvars]=1.;
    lvars++;
    }

  if(right<0){
    for(i=0;i<lvars;i++)
      coef[i]=-coef[i];
    right=-right;
    }

  if(nresx==nrx+1){
    matx=(struct aax *)mymalloc(sizeof(struct aax));
    matx[0].icapx=NULL;
    }
  else{
    matx=(struct aax *)myrealloc((struct aax *)matx,(nresx-nrestr)*sizeof(struct
    matx[nresx-nrestr-1].icapx=NULL;

    }

  for(i=0;i<lvars;i++){
    for(j=0;j<varnumx;j++)   /*vou ver se  uma das vari veis j  existentes*/
```

```
      if(!strcmp(vars[i],tokens[j]))break;

    if(j==varnumx){ /* se ainda n
                                o existe*/
      varnumx++;
      if(varnumx==nmax1){  /*se cheguei ao limite da mem¢ria aloco mais*/
        nmax1+=MAXVARS;
        tokens=(char **)myrealloc((char *)tokens,nmax1*sizeof(char *));
        }
      tokens[j]=(char *)mymalloc((strlen(vars[i])+1)*sizeof(char));
      strcpy(tokens[j],vars[i]);
      }

    y=(struct tabx *)mymalloc(sizeof(struct tabx));
    y->itok=j;
    y->value=coef[i];
    bx[nresx-nrestr-1]=right;

    if(matx[nresx-nrestr-1].icapx==NULL)
      matx[nresx-nrestr-1].nlx=nresx;
    y->next=matx[nresx-nrestr-1].icapx;
    matx[nresx-nrestr-1].icapx=y;
    }
}


/*********************************************************/
/************* L as restries do ficheiro **************/
/*********************************************************/
void Le_restricoes(fp)
FILE *fp;
{
  int code;
  char palavra[MAXSIZE];  /* guarda uma palavra */
  double num;
  int i;
  int ccc=1;

  lvars=0;
  code=0;

  num=Le_num(fp);
  code=Le_palavra(fp,palavra);
  if(code<0){
    fprintf(stderr,"Erro na estrutura do ficheiro!\n");
    exit(1);
    }

  while(code>=0){
    do{
      strcpy(vars[lvars],palavra);
      coef[lvars]=num;
      lvars++;
      num=Le_num(fp);
      coef[lvars]=num;
      code=Le_palavra(fp,palavra);
      }while(code==0);
    strcpy(vars[lvars],palavra);

    if(code>0)
      right=Le_num(fp);
    else{
      fprintf(stderr,"Erro na estrutura do ficheiro!\n");
      exit(1);
      }
```

```
/*
    for(i=0;i<lvars;i++)
      printf("%f %s ",coef[i],vars[i]);
     printf("%s %f\n\n",vars[lvars],right);
*/
    if(ccc<=nrx){
      ccc++;
      Filtragem();   /* analisa a restri
                                          o e junta-a
 matriz */

      nresx++;
      varnumx=varnum;
      if(++nrestr==nmax2){
        count2++;
        nmax2+=MAXRESTR;
        b=(double *)myrealloc((char *)b,nmax2*sizeof(double));
        }
      num=Le_num(fp);
      code=Le_palavra(fp,palavra);
      lvars=0;
      }
    else{
      nresx++;
      outras();
      if(nresx==nmax2){
        nmax2+=MAXRESTR;
        bx=(double *)myrealloc((char *)bx,nmax2*sizeof(double));
        }
      num=Le_num(fp);
      code=Le_palavra(fp,palavra);
      lvars=0;
      }
    }/*end while*/

}


/*********************************************************/
/****** Fun
          o principal e l objectivo a minimizar ******/
/*********************************************************/
struct matriz *lindo_fich(name)
char *name;
{
FILE *fp;
int ch,k,i;
double val;
char type;
char string[MAXSIZE+1];
char palavra[MAXSIZE];  /* guarda uma palavra */
double num;
int code=0;
int sign;
struct matriz *matrix=NULL;
matrix=(struct matriz *)mymalloc(sizeof(struct matriz));
ch=0;
lvars=0;

if((fp=fopen(name,"r"))==NULL){
  fprintf(stderr,"\nCouldn't open %s !\n",name);
  exit(1);
  }
fscanf(fp,"%*[^M]M%c",&ch);

vars=(char **)mymalloc(MAXVARS_R*sizeof(char *));
```

```
for(i=0;i<MAXVARS_R;i++)
  vars[i]=(char *)mymalloc(MAXSIZE*sizeof(char));
coef=(double *)mymalloc(MAXVARS_R*sizeof(double));
c=(double *)mymalloc(nmax1*sizeof(double));
b=(double *)mymalloc(nmax2*sizeof(double));
bx=(double *)mymalloc(nmax2*sizeof(double));
tokens=(char **)mymalloc(nmax1*sizeof(char *));
mata=(struct aa *)mymalloc(nmax1*sizeof(struct aa));

for(i=0;i<nmax1;i++){
  mata[i].icapa=NULL;
  mata[i].nozea=0;
  c[i]=0.;
  }
tam=0;
nn=0;
ch=toupper(ch);

if(ch=='I'){      /* se for uma minimiza
                                        o */
  ch=getc(fp);
  ch=toupper(ch);
  if(ch!='N'){
    fprintf(stderr,"Erro na leitura do ficheiro '%s'\n",name);
    exit(FILE_ERROR);
    }
  type=MIN;
  }
else{
  if(ch=='A'){    /* se for uma maximiza
                                        o */
    ch=getc(fp);
    ch=toupper(ch);
    if(ch!='X'){
      fprintf(stderr,"Erro na leitura do ficheiro '%s'\n",name);
      exit(FILE_ERROR);
      }
    type=MAX;
    }
  else{
    fprintf(stderr,"Erro na leitura do ficheiro '%s'\n",name);
    exit(FILE_ERROR);
    }
  }
if(type==MAX){
  sign=-1;
  printf("Maximiza
                  o !\n");
  }
else{
  if(type==MIN){
    printf("Minimiza
                    o !\n");
    sign=1;
    }
  else{
    fprintf(stderr,"Nem uma nem outra !\n");
    exit(1);
    }
  }

ch=getc(fp);

num=Le_num(fp);
code=Le_palavra2(fp,palavra);
if(code<0){
```

```
   fprintf(stderr,"Erro na estrutura do ficheiro!\n");
   exit(1);
   }

 do{
   strcpy(vars[lvars],palavra);
   c[lvars]=num*sign;
   lvars++;
   num=Le_num(fp);
   code=Le_palavra2(fp,palavra);
   }while(code==0);

 for(i=0;i<lvars;i++){
   tokens[i]=(char *)mymalloc( (strlen(vars[i])+1)*sizeof(char) );
   strcpy(tokens[i],vars[i]);
   mata[i].icapa=NULL;
   }

 tam=varnum=lvars;
 Le_palavra2(fp,palavra);

 if(strcmp(palavra,"TO")!=0){
   fprintf(stderr,"Erro na frase SUBJECT TO\n");
   exit(1);
   }

 printf("fun
           o objectivo  :\n");
 for(i=0;i<varnum;i++)
   printf("%f %s ",c[i],tokens[i]);

 printf("\n\n");

 printf("INTRODUZA QUAL O N§ DE RESTRIOES A LER DE INICIO!\n");
 scanf(" %d",&nrx);
 printf("nrx=%d\n",nrx);

 Le_restricoes(fp);

 tokens=(char **)myrealloc((char *)tokens,varnumx*sizeof(char *));
 mata=(struct aa *)myrealloc((char *)mata,varnum*sizeof(struct aa));
 c=(double *)myrealloc((char *)c,varnum*sizeof(double));
 b=(double *)myrealloc((char *)b,nrestr*sizeof(double));
 bx=(double *)myrealloc((char *)bx,(nresx-nrestr)*sizeof(double));

 if(coef!=NULL){
   myfree(coef);
   coef=NULL;
   }
 else
   FatalErr(SOFTBUG,"coef not allocated (lindo_fich)");
 for(i=0;i<MAXVARS_R;i++){
   if(vars[i]!=NULL)
     myfree(vars[i]);
   else
     FatalErr(SOFTBUG,"vars[i] not allocated (lindo_fich)");
   }
 if(vars!=NULL){
   myfree(vars);
   coef=NULL;
   }
 else
   FatalErr(SOFTBUG,"vars not allocated (lindo_fich)");
 /*
 for(i=0;i<varnum;i++)
   printf("vari vel %d ♈ %s\n",i,tokens[i]);
```

```
for(;i<varnumx;i++)
  printf("Nova vari vel %d ??? %s\n",i,tokens[i]);

for(i=0;i<nrestr;i++)
  printf("b[%d] ?? %f\n",i,b[i]);

for(i=0;i<nresx-nrestr;i++)
  printf("bx[%d]=%f\n",i,bx[i]);
getch();
*/
nl=nrestr;
nlx=nresx;
tam=nci=varnum;
ncix=varnumx;

matrix->mata=mata;
matrix->matx=matx;
matrix->nl=nl;
matrix->nlx=nlx;
matrix->nci=nci;
matrix->ncix=ncix;
matrix->bx=bx;
matrix->b=b;
matrix->c=c;
matrix->c1=NULL;
matrix->ca=NULL;

return(matrix);
}
```

```
#include<stdio.h>
#include<malloc.h>

#define MAXINT 20
/*
extern int nl;
*/
int l,nl;
double **q;
double *bi;
double *bf;
double *vm;
char *xa;
int miu;/*indica qual o estudo, se miu=1 ou miu=1 ate 0 */
int flag;
int flag1;/*indica qual o gama*/

/************************************/
/*******inicio a matriz q[][]*********/
/************************************/
ini_q()
{
  int i,j,ni;              /*ni- n§ de intervalos*/
  double ampl;              /*ampl- amplitude dos intervalos*/
  nl=3;                    /*nl- n§ de variaveis basicas*/
  printf("Quantos intervalos?\n");
  scanf("%d",&ni);
  if(ni>MAXINT){
    ni=20;
    printf("n§ maximo de intervalos --20\n");
    getch();
    }
  printf("ni-n§ de intervalos=%d\n",ni);
  l=ni+1;                  /*l-n§ de linhas de q[][]*/
  vm=(double *)malloc(l*sizeof(double));
  ampl=1./(double)ni;
  printf("amplitude=%f\n",ampl);
  for(i=0;i<l;i++)
    vm[i]=1.0-(double)i/(double)ni;
  for(i=0;i<l;i++)
    printf("vm[%d]=%.4f ",i,vm[i]);
  printf("\n");

/*******alocar matriz*************************************************************/
  q=(double **)malloc(l*sizeof(double *));
  for(i=0;i<l;i++)
    q[i]=(double *)malloc((nci*2)*sizeof(double));
  for(i=0;i<l;i++)
    for(j=0;j<nci*2;j++)
      q[i][j]=0.;

/****valores das variaveis basicas*********************************************/
  bi=(double *)malloc(nl*sizeof(double));
  bf=(double *)malloc(nl*sizeof(double));
  for(i=0;i<nl;i++){
    bi[i]=0.;
    bf[i]=0.;
    }
  for(i=0;i<nl;i++){
    bi[i]=2+i;
    printf("bi[%d]=%lf",i,bi[i]);
    }
  printf("\n");
  for(i=0;i<nl;i++){
    bf[i]=4+i;
    printf("bf[%d]=%lf",i,bf[i]);
```

```
    }
  printf("\n");
}
/************************************/
/***introduzir os valores no quadro***/
/************************************/
valores_q(g0,g1)
double g1,g0;
{
int i,j,k,a,indice;
double x2,m;
for(k=0;k<nci;k++){
  for(j=0;j<nl;j++){
    if(xb[j]=k){ /*alterar a posicao 2*k e 2*k+1 */

      indice=2*k;
      if(miu==1){*
        q[0][indice]=bi[k]+g0*bf[k];
        q[0][indice+1]=bi[k]+g1*bf[k];
        if(ax[k]==1)                    /*compara valores ax[k]==1*/
          for(i=1;i<l;i++){
            if(q[i][indice]>q[i-1][indice])q[i][indice]=q[i-1][indice];
            if(q[i][indice+1]<q[i-1][indice+1])q[i][indice+1]=q[i-1][indice+1];
            }
      }
      else{           /* miu=1 ate miu=0*/
        m=bi[k]+g1*bf[k];
        for(i=0;i<l;i++)
          if((vm[i]<g1 || g1==1.) && vm[i]>=g0){
            x2=bi[k]+vm[i]*bf[k];
            if(x2>m){
              if(ax[k]==0){             /*nao e preciso comparar*/
                q[i][indice]=m;
                q[i][indice+1]=x2;
                }
              else
                if(x2>q[i][indice+1])q[i][indice+1]=x2;
                else ;
              }
            else
              if(x2<m){
                if(ax[k]==0){
                  q[i][indice]=x2;
                  q[i][indice+1]=m;
                  }
                else
                  if(x2<q[i][indice])q[i][indice]=x2;
                  else ;
                }
              else{/***x2==m***/
                if(vm[i]!=1.){
                  q[i][indice]=q[i-1][indice];
                  q[i][indice+1]=q[i-1][indice+1];
                  }
                else{
                  q[i][indice]=x2;
                  q[i][indice+1]=x2;
                  }
                }
            }
          if(ax[k]==1){
            for(i=0;i<l && vm[i]>=g0;i++);
            if(vm[i]<g0)
              for(a=i;a<l;a++){
                if(q[a][indice]>q[a-1][indice])q[a][indice]=q[a-1][indice];
                if(q[a][indice+1]<q[a-1][indice+1])q[a][indice+1]=q[a-1][indice+
```

```
                }
            }/*endif*/
        }
      }
    }
  }
printf("n- de linhas=%d\n",l);
for(i=0;i<l;i++){
  printf("v_miu[%d]=%f\n",i,vm[i]);
  for(j=0;j<nl*2;j++)
    printf("%10.5f",q[i][j]);
  printf("\n");
  }
result(l,nl);
}
/**********apresenta
                    o dos resultados*****************************************/

result(l,nl)
int l,nl;
{
  int i,j;
  printf("\nvalores minimos\n");
  for(i=l-1;i>=0;i--)
    printf("  %10.5f",vm[i]);
  printf("\n\n");
  for(j=0;j<nl*2;j+=2){
    printf("x%d",j);
    i=l-1;
    printf("%10.5f",q[i][j]);
    for(i=l-2;i>=0;i--)
      printf("  %10.5f",q[i][j]);
    printf("\n");
    }
  printf("\nvalores Maximos\n");
  for(i=l-1;i>=0;i--)
    printf("  %10.5f",vm[i]);
  printf("\n\n");
  for(j=0;j<nl*2;j+=2){
    printf("x%d",j);
    i=l-1;
    printf("%10.5f",q[i][j+1]);
    for(i=l-2;i>=0;i--)
      printf("  %10.5f",q[i][j+1]);
    printf("\n");
    }
}
/*********************************/
/******intervalo gam0,gam1********/
/*********************************/
ler_inter()
{
  double gam0,gam1;
/*************/
  gam0=0.75;
  flag1=1;
/*************/
/*  miu=1;*/
  if(flag1==0){
    gam1=1.;
    printf("introduza o 1§gama\n");
    scanf("%lf",&gam0);
    printf("intervalo [%f,%f]\n",gam0,gam1);
    }
  else{
    gam1=gam0;
```

```
    printf("introduza o gam0\n");
    scanf("%lf",&gam0);
    printf("intervalo [%f,%f[\n",gam0,gam1);
    }
  valores_q(gam0,gam1);
}
/**********************/
/***fun
      o principal***/
/**********************/
main()
{
  int i;

  flag1=0;/*se 0 intervalo [...,...]*/
          /*se 1 intervalo [...,...[*/
  nci=6
  ax=(char *)malloc(nci*sizeof(char));
  for(i;i<nci;i++)
    xa[i]=0;
  ini_q();/*aloca quadro com dimensÆo l*nci */
  ler_inter();

}
```

```
#include<stdio.h>
#include<process.h>
#include<stdlib.h>

#include"common.h"
#include"simplex.h"

#define LIM   (double)0.0000001

/* declara
          o das estruturas que v
                                o manter a informa
                                                    o */

struct aa *mata1=NULL;
struct tt *matt=NULL,*last;

double *chelp=NULL;
int flag;

void liberta_matriz(matrix)
struct matriz *matrix;
{
  struct tab1 *p,*p1;
  int i;
  struct aa *mata;

  mata=matrix->mata;

  for(i=0;i<matrix->nci;i++){
    p=mata[i].icapa;
    while(p!=NULL){
      p1=p->next;
      myfree(p);
      p=p1;
      }
    }
  if(matrix->mata!=NULL){
    myfree(matrix->mata);
    matrix->mata=NULL;
    }
  else
    FatalErr(SOFTBUG,"matriz->mata not allocated (liberta_matriz)");
  if(matrix->b!=NULL){
    myfree(matrix->b);
    matrix->b=NULL;
    }
  else
    FatalErr(SOFTBUG,"matriz->b not allocated (liberta_matriz)");
  if(matrix->c!=NULL){
    myfree(matrix->c);
    matrix->c=NULL;
    }
  else
    FatalErr(SOFTBUG,"matriz->c not allocated (liberta_matriz)");
  if(matrix->xb!=NULL){
    myfree(matrix->xb);
    matrix->xb=NULL;
    }
  else
    FatalErr(SOFTBUG,"matriz->xb not allocated (liberta_matriz)");
}



/*******************************************************/
```

```
/*Elimina equaoes redundantes e retira vas. art. da base*/
/*********************************************************/
void procura(matrix,c1,ca,cb)
struct matriz *matrix;
char *c1;
double *ca;
double *cb;
{
  struct aa *mata;
  int nl;
  int nci;
  double *b;
  double *c;
  int *xb;

  struct tab1 *p0,*p1,*fi;
  double *p=NULL;
  int imin,imin2;
  struct tt * t;
  int i,j,k,l;
  double tot;

  mata=matrix->mata;
  nl=matrix->nl;
  nci=matrix->nci;
  b=matrix->b;
  c=matrix->c;
  xb=matrix->xb;

  p=(double *)mymalloc(nl*sizeof(double));

  for(k=0;k<nl;k++){
    if(xb[k]>=nci){
      imin2=k;

      for(l=0;l<nci;l++){

        if(c1[l]==0 && ca[l]>-LIM && ca[l]<LIM){
          imin=l;

          for(i=0;i<nl;i++)
            p[i]=0.;
          p0=mata[imin].icapa;
          while(p0!=NULL){
            p[p0->irow]=p0->value;
            p0=p0->next;
            }

          t=matt;
          while(t!=NULL){
            tot=0.;
            p0=t->icapt;
            while(p0!=NULL){
              if(p0->irow==t->ncolt)
                tot=p[p0->irow]*p0->value;
              else
                p[p0->irow]+=p0->value*p[t->ncolt];
              p0=p0->next;
              }
            p[t->ncolt]=tot;
            t=t->next;
            }

          if(p[k]>LIM || p[k]<-LIM){

            j=0;
```

```
            if(matt==NULL)
              t=matt=(struct tt *)mymalloc(sizeof(struct tt));
            else
              t=(struct tt *)mymalloc(sizeof(struct tt));
            if(last!=NULL){
              t->last=last;
              last->next=t;
              }
            else
              t->last=NULL;
            t->next=NULL;
            fi=p1=(struct tab1 *)mymalloc(sizeof(struct tab1));
            for(i=0;i<nl;i++){
              if(p[i]){
                if(i==imin2)
                  p1->value=1/p[imin2];
                else
                  p1->value=-p[i]/p[imin2];
                p1->irow=i;
                p0=p1;
                p1=(struct tab1 *)mymalloc(sizeof(struct tab1));
                p0->next=p1;
                j++;
                }
              }
            p0->next=NULL;
            t->icapt=fi;
            t->nozet=j;
            t->ncolt=imin2;
            t->entra=imin;
            c1[xb[imin2]]=0;
            c1[imin]=1;
            xb[imin2]=imin;
            last=t;

            tot=0.;
            p0=last->icapt;
            while(p0!=NULL){
              if(p0->irow==last->ncolt)
                tot=b[p0->irow]*p0->value;
              else
                b[p0->irow]+=p0->value*b[last->ncolt];
              p0=p0->next;
              }
            b[last->ncolt]=tot;
            break;
            }
          }
        }
      if(l==nci)
        printf("Existe uma equacao redundante \n");
      }
    }

  if(p!=NULL){
    myfree(p);
    p=NULL;
    }
  else
    FatalErr(SOFTBUG,"p not allocated");
}


/*********************************************/
/*          Copia um vector para outro       */
```

```
/*************************************************/
void copia_vector(c,c1,dim)
double *c;
double *c1;
int dim;
{
  int i;
  for(i=0;i<dim;i++)
    c1[i]=c[i];
}



/*************************************************/
/* Verifica se precisa de vari veis artificiais */
/*************************************************/
char *ver_restr(matrix,tam)
struct matriz *matrix;
int *tam;
{
  struct aa *mata;
  int nl;
  int nci;
  double *b;
  double *c;
  int *xb=NULL;
  char *c1=NULL;

  int i,j,k,l;
  long t0,t1;
  struct tab1 *p0;

  mata=matrix->mata;
  nl=matrix->nl;
  nci=matrix->nci;
  b=matrix->b;
  c=matrix->c;

  *tam=nci;

  xb=(int *)mymalloc(nl*sizeof(int));
  matrix->xb=xb;

  c1=mymalloc((*tam)*sizeof(char));

  for(i=0;i<nl;i++){       /* inicializa
                                  o dos vectores xb e c1 */
    xb[i]=-1;
    c1[i]=0;
    }
  for(;i<(*tam);i++)
    c1[i]=0;

  /* marca
        o das vari veis que v
                             o pertencer
 base */
  /* k conta as vari veis b sicas que j  temos */
  for(i=0,k=0;i<nci;i++){
    if(mata[i].nozea==1){
      p0=mata[i].icapa;
      if(p0->value==1. && xb[p0->irow]<0){
        k++;
        xb[p0->irow]=i;
        c1[i]=1;
        }
```

```
      }
    }

  /* vou alocar espaco de alguns vectores e inicializar a base */
  j=nl-k; /* j = n§ de vari veis aleat¢rias a criar */
  if(j){ /* se precisar de vari veis artificiais */
    chelp=(double *)mymalloc(nci*sizeof(double));
    copia_vector(c,chelp,nci);
    myfree(c);
    c=NULL;
    *tam=*tam+j;
    c=(double *)mymalloc((*tam)*sizeof(double));
    copia_vector(chelp,c,nci);
    matrix->c=c;
    c1=(char *)myrealloc(c1,(*tam)*sizeof(char));
    mata1=(struct aa *)mymalloc((*tam-nci)*sizeof(struct aa));
    j=0;
    l=0;
    for(i=nci;i<(*tam);i++,k++,j++,l++){
      c[i]=1.;
      c1[i]=1;
      while(xb[j]>=0)j++;
      p0=(struct tab1 *)mymalloc(sizeof(struct tab1));
      mata1[l].icapa=p0;
      mata1[l].nozea=1;
      xb[j]=i;
      p0->value=1.;
      p0->irow=j;
      p0->next=NULL;
      }
    for(i=0;i<nci;i++)
      c[i]=0.;
    }

  return(c1);
}


/*************************************************/
/* Fase 1 - elimina
                  o de vari veis artificiais */
/*************************************************/
void fase_um(matrix,c1,tam)
struct matriz *matrix;
char *c1;
int tam;
{
  struct aa *mata;
  int nl;
  int nci;
  double *b;
  double *c;
  int *xb;

  struct tab1 *p0,*p1,*fi; /* apontadores necess rios */
  struct tt *t;      /* apontadores necess rios */
  int test; /* Assinala o fim das iteraes ( C'a todos > 0 ) */
  int i,j,l;
  double minimo; /* valor m¡nimo */
  int imin;   /* indice do m¡nimo (entrar na base) */
  int imin2;  /* indice do m¡nimo (sair da base) */
  double *p=NULL;  /* vector usado no algoritmo */
  double tot;
  int cont=0;
  double *ca=NULL,*cb=NULL;
  int nc;
```

```
mata=matrix->mata;
nl=matrix->nl;
nci=matrix->nci;
b=matrix->b;
c=matrix->c;
xb=matrix->xb;

matt=NULL;
last=NULL;

test=1;

ca=(double *)mymalloc(tam*sizeof(double)); /* Ca */
cb=(double *)mymalloc(nl*sizeof(double));  /* Cb */
p=(double *)mymalloc(nl*sizeof(double));   /* P  */

nc=tam-nl;  /* n§ de colunas n
                                  o b sicas */
while(test){   /* enquanto tiver coeficientes negativos */
  /******** PASSO ii *********/
  /*   vou inicializar Cb e Ca   */
  for(i=0;i<nl;i++){
    cb[i]=c[xb[i]];
    ca[i]=c[i];
    }
  for(;i<tam;i++)
    ca[i]=c[i];

  /* vou calcular C'a = Ca - Cb*B(-1)*A'  */
       /* Cb*B(-1) ---> Cb */

  t=last;
  while(t!=NULL){
    tot=0.;
    p0=t->icapt;
    while(p0!=NULL){
      tot+=cb[p0->irow]*p0->value;
      p0=p0->next;
      }
    cb[t->ncolt]=tot;
    t=t->last;
    }

       /* C'a = Ca - Cb*A' */
  j=0;
  for(i=0;i<nc;i++,j++){
    while(c1[j])
      ca[j++]=0;
    if(j<nci)
      p0=mata[j].icapa;
    else
      p0=mata1[j-nci].icapa;
    while(p0!=NULL){
      ca[j]-=p0->value*cb[p0->irow];
      p0=p0->next;
      }
    }

  /* vou achar o coeficiente negativo m¡nimo */
  i=0;
  flag=0;
  while( ca[i]>-LIM && i<nci )i++;
  if(i==nci){
    for(i=0;i<nl;i++){
       if(xb[i]>=nci)
```

```
        cont++;
      if(xb[i]>=nci && b[i])
        flag=1;
      }
  if(cont==0 || flag==1) break;
  procura(matrix,c1,ca,cb);
  break;
  }

minimo=ca[i];
imin=i;
while(++i<nci)
  if( ca[i]<minimo){
    imin=i;
    minimo=ca[i];
    }

for(i=0;i<nl;i++)
  p[i]=0.;
if(imin<nci)
  p0=mata[imin].icapa;
else
  p0=mata1[imin-nci].icapa;
while(p0!=NULL){
  p[p0->irow]=p0->value;
  p0=p0->next;
  }

/* Vou calcular P'(r) = B(-1)*P(r)  = Tr*...*T1*P(r)   */
t=matt;
while(t!=NULL){    /* para cada uma das matrizes T's */
  tot=0.;
  p0=t->icapt;
  while(p0!=NULL){  /* para cada um dos elementos existentes na coluna de T
    if(p0->irow==t->ncolt)
      tot=p[p0->irow]*p0->value;
    else
      p[p0->irow]+=p0->value*p[t->ncolt];
    p0=p0->next;
    }
  p[t->ncolt]=tot;
  t=t->next;
  }

/* vou achar a vari vel a retirar da base */
minimo=0;
i=0;
while(p[i]<LIM && i<nl )
  i++;
if(i==nl){
  printf("BRONCA DA GROSSA\n");
  getchar();
  }
imin2=i;
minimo=b[i]/p[i];
while(i<nl){
  if( p[i]>LIM )
    if(b[i]/p[i] < minimo){
      imin2=i;                    /* imin2 - coluna a retirar da base */
      minimo=b[i]/p[i];           /* xb[imin] - vari vel a retirar da base *
      }
  i++;
  }

/******* PASSO v *******/
/* c lculo da matriz de transforma
```

```
                                        o T(r) */
    j=0;     /* j - n§ de elementos != de zero */
    if(matt==NULL)
      t=matt=(struct tt *)mymalloc(sizeof(struct tt));
    else
      t=(struct tt *)mymalloc(sizeof(struct tt));
    if(last!=NULL){     /* ligaao aos elementos da lista */
      t->last=last;
      last->next=t;
      }
    else
      t->last=NULL;   /* se for o 1§ elemento da lista */
    t->next=NULL;
    fi=p1=(struct tab1 *)mymalloc(sizeof(struct tab1));
    for(i=0;i<nl;i++){
      if(p[i]){
        if(i==imin2)
          p1->value=1/p[imin2];
        else
          p1->value=-p[i]/p[imin2];
        p1->irow=i;
        p0=p1;
        p1=(struct tab1 *)mymalloc(sizeof(struct tab1));
        p0->next=p1;
        j++;
        }
      }

    p0->next=NULL;
    t->icapt=fi;
    t->nozet=j;
    t->ncolt=imin2;
    t->entra=imin;
    c1[xb[imin2]]=0; /* desmarco a vari vel que vai sair da base */
    c1[imin]=1;      /* marco vari vel que entra na base */
    xb[imin2]=imin;  /* introduzo nova vari vel na base */
    last=t;

    /* Vou calcular b'= B(-1)*b = T(r)*...*T(1)*b */
    tot=0.;
    p0=last->icapt;           /* para a ultima das matrizes T's */
    while(p0!=NULL){          /* para cada um dos elementos existentes na coluna
      if(p0->irow==last->ncolt)
        tot=b[p0->irow]*p0->value;
      else
        b[p0->irow]+=p0->value*b[last->ncolt];
      p0=p0->next;
      }
    b[last->ncolt]=tot;
    }

  if(p!=NULL){
    myfree(p);
    p=NULL;
    }
  else
    FatalErr(SOFTBUG,"p not allocated (phase 1)");
  if(cb!=NULL){
    myfree(cb);
    cb=NULL;
    }
  else
    FatalErr(SOFTBUG,"cb not allocated (phase 1)");
}
```

```
/*************************************************/
/******** Fase 2 - resolu
                          o do simplex *********/
/*************************************************/
void fase_dois(matrix,c1,tam)
struct matriz *matrix;
char *c1;
int tam;
{
  struct aa *mata;
  int nl;
  int nci;
  double *b;
  double *ca,*c;
  int *xb;

  struct tab1 *p0,*p1,*fi; /* apontadores necess rios */
  struct tt *t;      /* apontadores necess rios */
  int test; /* Assinala o fim das iteraes ( C'a todos > 0 ) */
  int i,j,l;
  double minimo; /* valor m¡nimo */
  int imin;   /* indice do m¡nimo (entrar na base) */
  int imin2;  /* indice do m¡nimo (sair da base) */
  double *p=NULL;  /* vector usado no algoritmo */
  double tot;
  double *cb=NULL;
  int nc;

  mata=matrix->mata;
  nl=matrix->nl;
  nci=matrix->nci;
  b=matrix->b;
  c=matrix->c;
  xb=matrix->xb;

  test=1;

  ca=(double *)mymalloc(tam*sizeof(double)); /* Ca */
  cb=(double *)mymalloc(nl*sizeof(double));  /* Cb */
  p=(double *)mymalloc(nl*sizeof(double));   /* P  */

  nc=tam-nl;  /* n§ de colunas n
                              o b sicas */
  while(test){   /* enquanto tiver coeficientes negativos */
    /******** PASSO ii *********/
    /*   vou inicializar Cb e Ca   */
    for(i=0;i<nl;i++){
      cb[i]=c[xb[i]];
      ca[i]=c[i];
      }
    for(;i<nci;i++)
      ca[i]=c[i];

    /* vou calcular C'a = Ca - Cb*B(-1)*A'  */
        /* Cb*B(-1) ---> Cb */
    t=last;
    while(t!=NULL){
      tot=0.;
      p0=t->icapt;
      while(p0!=NULL){
        tot+=cb[p0->irow]*p0->value;
        p0=p0->next;
        }
      cb[t->ncolt]=tot;
      t=t->last;
      }
```

```
        /* C'a = Ca - Cb*A' */
    j=0;
    for(i=0;i<nc;i++,j++){
      while(c1[j])
        ca[j++]=0.;
      if(j<nci)
        p0=mata[j].icapa;
      else{
        fprintf(stderr,"Erro - na 2ª fase ainda temos ainda vari veis artificiai
        exit(1);
        }
      while(p0!=NULL){
        ca[j]-=p0->value*cb[p0->irow];
        p0=p0->next;
        }
      }

    /* vou achar o coeficiente negativo m¡nimo */
    i=0;
    while( (c1[i] || ca[i] > -LIM ) && i<nci )i++;
    if(i==nci)
      break;
    minimo=ca[i];
    imin=i;

    while(++i<nci)
      if(!c1[i] && ca[i]<minimo){ /************ PASSO iii *************/
        imin=i;                   /* imin - vari vel a entrar na base */
        minimo=ca[i];
        }

    /******** PASSO iv ********/
    /* Vou calcular o P(r) */
    for(i=0;i<nl;i++)
      p[i]=0.;
    p0=mata[imin].icapa;
    while(p0!=NULL){
      p[p0->irow]=p0->value;
      p0=p0->next;
      }

    /* Vou calcular P'(r) = B(-1)*P(r)  = Tr*...*T1*P(r)   */
    t=matt;
/*  if(t==NULL) printf("Estupido esta mal\n");*/
    while(t!=NULL){    /* para cada uma das matrizes T's */
      tot=0.;
      p0=t->icapt;
      while(p0!=NULL){  /* para cada um dos elementos existentes na coluna de T
        if(p0->irow==t->ncolt)
          tot=p[p0->irow]*p0->value;
        else
          p[p0->irow]+=p0->value*p[t->ncolt];
        p0=p0->next;
        }
      p[t->ncolt]=tot;
      t=t->next;
      }

    /* vou achar a vari vel a retirar da base */
    minimo=0;
    i=0;
    while(p[i]<LIM && i<nl)
      i++;
    imin2=i;
    if(i==nl){
```

```
  fprintf(stderr,"Erro - N
                          o encontrei custos reduzidos negativos\n");
  exit(1);
  }
minimo=b[i]/p[i];
while(i<nl){
  if( p[i]>LIM )
    if( b[i]/p[i] < minimo ){
      imin2=i;                    /* imin2 - coluna a retirar da base */
      minimo=b[i]/p[i];           /* xb[imin] - vari vel a retirar da base *
      }
  i++;
  }

/******* PASSO v *******/
/* c lculo da matriz de transforma
                                  o T(r) */
j=0;     /* j - n§ de elementos != de zero */

if(matt==NULL)
  t=matt=(struct tt *)mymalloc(sizeof(struct tt));
else
  t=(struct tt *)mymalloc(sizeof(struct tt));
if(last!=NULL){     /* ligaao aos elementos da lista */
  t->last=last;
  last->next=t;
  }
else
  t->last=NULL;   /* se for o 1§ elemento da lista */
t->next=NULL;
fi=p1=(struct tab1 *)mymalloc(sizeof(struct tab1));
for(i=0;i<nl;i++){
  if(p[i]){
    if(i==imin2)
      p1->value=1/p[imin2];
    else
      p1->value=-p[i]/p[imin2];
    p1->irow=i;
    p0=p1;
    p1=(struct tab1 *)mymalloc(sizeof(struct tab1));
    p0->next=p1;
    j++;
    }
  }

p0->next=NULL;
t->icapt=fi;
t->nozet=j;
t->ncolt=imin2;
t->entra=imin;
c1[xb[imin2]]=0; /* desmarco a vari vel que vai sair da base */
c1[imin]=1;      /* marco vari vel que entra na base */
xb[imin2]=imin;  /* introduzo nova vari vel na base */
last=t;

/* Vou calcular b'= B(-1)*b = T(r)*...*T(1)*b */
tot=0.;
p0=last->icapt;           /* para a ultima das matrizes T's */
while(p0!=NULL){          /* para cada um dos elementos existentes na coluna
  if(p0->irow==last->ncolt)
    tot=b[p0->irow]*p0->value;
  else
    b[p0->irow]+=p0->value*b[last->ncolt];
  p0=p0->next;
  }
b[last->ncolt]=tot;
```

```
    }

  if(p!=NULL){
    myfree(p);
    p=NULL;
    }
  else
    FatalErr(SOFTBUG,"p not allocated (phase 2)");
  matrix->ca=ca;
  if(cb!=NULL){
    myfree(cb);
    cb=NULL;
    }
  else
    FatalErr(SOFTBUG,"cb not allocated (phase 2)");
}


void simplex(matrix)
struct matriz *matrix;
{
  struct tab1 *p0,*p1;   /* apontadores necess rios */
  struct tt *t,*t0;      /* apontadores necess rios */
  int i;
  int tam;
  int nci;
  int *xb;
  int nl;
  double *c;

  char *c1=NULL;  /* indices que fazem parte de C'a */

  nci=matrix->nci;

  c1=ver_restr(matrix,&tam);

  printf("Passei Ver_restricoes\n");

  if(nci<tam){

    fase_um(matrix,c1,tam);

    printf("Passei Fase 1\n");

    tam=nci;

    c=matrix->c;
    if(c!=NULL){
      myfree(c);
      c=NULL;
      }
    else
      FatalErr(SOFTBUG,"matrix->c not allocated (simplex)");

    c=(double *)mymalloc(nci*sizeof(double));
    copia_vector(chelp,c,nci);
    matrix->c=c;

    if(chelp!=NULL){
      myfree(chelp);
      chelp=NULL;
      }
    else
      FatalErr(SOFTBUG,"chelp not allocated (simplex)");

    if(flag==1)
```

```
      printf("EXISTEM EQUAOES INCOMPATIVEIS\n");
    else
      fase_dois(matrix,c1,tam);
    }
  else{
    if(nci==tam)
      fase_dois(matrix,c1,tam);
    else{
      printf("Ocorreu algum erro !\n");
      exit(4);
      }
    }

  for(i=nci;i<tam;i++){
    p0=mata1[i].icapa;
    while(p0!=NULL){
      p1=p0->next;
      myfree(p0);
      p0=p1;
      }
    }

  if(mata1!=NULL){
    myfree(mata1);
    mata1=NULL;
    }
/*
  else
    FatalErr(SOFTBUG,"mata1 not allocated (simplex)");
*/

  matrix->c1=c1;
  if(c1==NULL)
    FatalErr(SOFTBUG,"c1 not allocated (simplex)");
}
```

```
#include<stdio.h>
#include<process.h>
#include<stdlib.h>
#include"common.h"
#include"simplex.h"

extern double *sol;
extern char **tokens;
extern struct tt *matt,*last;
int sai,itpiv;
double vsai;
double **m;

tes_sol(matrix)
struct matriz *matrix;
{

  struct tabx *p0;
  int i;
  struct aax *matx;
  int nl,nlx;
  int nci;
  double *bx;
  double z,x;
  char *c1;

  matx=matrix->matx;
  nl=matrix->nl;
  nlx=matrix->nlx;
  c1=matrix->c1;
  nci=matrix->nci;
  bx=matrix->bx;
/*
  printf("nlx-nl=%d-%d\n",nlx ,nl);
  printf("nci=%d\n",nci);
  */
  sai=-1;
  vsai=0.;
  for(i=0;i<nlx-nl;i++){
    x=0.;
    for(p0=matx[i].icapx;p0!=NULL;p0=p0->next){
      if(p0->itok>=nci)
        z=p0->value;
      else
        if(c1[p0->itok]==1)
          x+=p0->value*sol[p0->itok];
      }/*endfor*/
    if(z!=0){
      if((bx[i]-x)/z < 0)
        if(((bx[i]-x)/z) < vsai){
          sai=i;
          vsai=(bx[i]-x)/z;
          printf("vsai=%f******sai=%d\n",vsai,sai);
          }
      }
  }
  if(sai!=-1){
    printf("sai a restricao %d e o valor da violacao e %f\n",sai,vsai);
    inser(matrix);
    }
  else{
    printf("A solucao ja satisfaz todas as outras restricoes\n");
    return (1);
    }
}
```

```
inser(matrix)
struct matriz *matrix;
{

  struct tabx *px,*py;
  struct tab1 *p0,*p1;
  int i;
  struct aax *matx;
  struct aa *mata;
  int nl;
  int nlx;
  int nci;
  double *bx;
  double *b;
  char *var;

  matx=matrix->matx;
  mata=matrix->mata;
  nl=matrix->nl;
  nlx=matrix->nlx;
  nci=matrix->nci;
  bx=matrix->bx;
  b=matrix->b;

  for(px=matx[sai].icapx; px!=NULL;px=px->next){
    p0=(struct tab1 *)mymalloc(sizeof(struct tab1));
    p0->irow=nl;
    p0->value=px->value;
    p0->next=NULL;

    if(px->itok>=nci){     /*nova variavel*/
      nci++;
      mata=(struct aa *)myrealloc((char *)mata,nci*sizeof(struct aa));
      mata[nci-1].nozea=1;
      mata[nci-1].icapa=p0;
      matrix->nci=nci;

      var=tokens[px->itok];
      tokens[px->itok]=tokens[nci-1];
      tokens[nci-1]=var;
      }
    else{
      p1=mata[px->itok].icapa;
      while(p1->next!=NULL)
        p1=p1->next;
      p1->next=p0;
      mata[px->itok].nozea++;
      }
    }
    nl++;
    matrix->nl=nl;

    for(px=matx[sai].icapx; px!=NULL; px=py){
      py=px->next;
      myfree(px);
      }
    matx[sai].icapx=matx[nlx-nl].icapx;
    matx[sai].nlx=matx[nlx-nl].nlx;
    bx[sai]=bx[nlx-nl];
    bx=(double *)myrealloc(bx,(nlx-nl)*sizeof(double));

/*    des_mata(matrix,nci);
    des_matx(matrix,nlx-nl);*/

  aumenta(matrix);
```

```
}


aumenta(matrix)
struct matriz *matrix;
{
  int i;
  int nl;
  int nci;
  double *b;
  int *xb;
  double *ca;
  double *c;
  char *c1;

  nl=matrix->nl;
  nci=matrix->nci;
  b=matrix->b;
  c1=matrix->c1;
  xb=matrix->xb;
  ca=matrix->ca;
  c=matrix->c;

  c1=(char *)myrealloc(c1,nci*sizeof(char));
  c1[nci-1]=1;  /*a base aumentou*/

  xb=(int *)myrealloc(xb,nl*sizeof(int));
  xb[nl-1]=nci-1;

  /*Zj-Cj*/
  ca=(double *)myrealloc(ca,nci*sizeof(double));
  ca[nci-1]=0.;

  c=(double *)myrealloc(c,nci*sizeof(double));
  c[nci-1]=0.;

  b=(double *)myrealloc(b,nl*sizeof(double));
  b[nl-1]=vsai;
  for(i=0;i<nl;i++)
    printf("b[%d]=%lf\n",i,b[i]);
  printf("\n");

  matrix->c1=c1;
  matrix->ca=ca;
  matrix->c=c;
  matrix->xb=xb;
  matrix->b=b;

  altera_t(matrix);
}

altera_t(matrix)
struct matriz *matrix;
{
  struct aa *mata;
  int i,j;
  int nl;
  int nci;
  double *b;
  double tot;
  double *nb,*nbr;

  struct tab1 *p0,*p1,*p2;
  struct tt *t;      /* aponta as matrizes t */

  mata=matrix->mata;
```

```
nl=matrix->nl;
nci=matrix->nci;
b=matrix->b;

nb=(double *)mymalloc((nl-1)*sizeof(double));
nbr=(double *)mymalloc((nl-1)*sizeof(double));
for(i=0;i<nl-1;i++)
  nb[i]=0.;

m=(double **)mymalloc((nl-1)*sizeof(double *));
for(i=0;i<nl-1;i++)
  m[i]=(double *)mymalloc((nl-1)*sizeof(double));
for(i=0;i<nl-1;i++){
  for(j=0;j<nl-1;j++)
    m[i][j]=0.;
  m[i][i]=1.;
  }

for(t=matt;t!=NULL;t=t->next){       /*Tn*Tn-1* ... *T0*M[][] */

  for(i=0;i<nl-1;i++)
    nbr[i]=0.;

  for(i=0;i<nl-1;i++){
    tot=0.;
    p0=t->icapt;
    while(p0!=NULL){
      if(p0->irow==t->ncolt)
        tot=m[t->ncolt][i]*(p0->value);
      else{
        m[p0->irow][i]+=p0->value*m[t->ncolt][i];
        if(m[p0->irow][i]<1.0e-6 && m[p0->irow][i]>-1.0e-6)
          m[p0->irow][i]=0.;
        }
      p2=p0;
      p0=p0->next;
    }/*endwhile*/

    m[t->ncolt][i]=tot;
    if(m[t->ncolt][i]<1.0e-6 && m[t->ncolt][i]>-1.0e-6)
      m[t->ncolt][i]=0.;
  }/*endfor*/


  for(p1=mata[t->entra].icapa; p1->irow!=nl-1 && p1!=NULL;p1=p1->next);
  if(p1->irow==nl-1)
    nb[t->ncolt]=-p1->value;
  else
    nb[t->ncolt]=0.;
  for(i=0;i<nl-1;i++)
    printf("????nb[%d]=%f\t",i,nb[i]);
  printf("\n");

  for(i=0;i<nl-1;i++){
    for(j=0;j<nl-1;j++)
      nbr[i]+=nb[j]*m[j][i];
    printf("????nbr[%d]=%f\t",i,nbr[i]);
  }
  printf("\n\n");
  getch();

  if(nbr[t->ncolt]!=0){
    p0=(struct tab1 *)mymalloc(sizeof(struct tab1));
    p0->irow=nl-1;
    p0->value=nbr[t->ncolt];
    p0->next=NULL;
```

```
      t->nozet++;
      p2->next=p0;
      }

  }/*endfor*/
  /***/

  itpiv=nl-1;
  itd(matrix);
}


itd(matrix)
struct matriz *matrix;
{

  struct tab1 *p0; /* apontador necess rio */
  struct tt *t;      /* apontador necess rio */
  int i,j;
  int imine;   /* indice do m¡nimo (entrar na base) */
  int imins;  /* indice do m¡nimo (sair da base) */
  double maxi;
  double *p=NULL;  /* vector usado no algoritmo */
  double tot;

  struct aa *mata;
  int nl;
  int nci;
  double *ca;
  char *c1;

  mata=matrix->mata;
  nl=matrix->nl;
  nci=matrix->nci;
  ca=matrix->ca;
  c1=matrix->c1;

  p=(double *)mymalloc(nl*sizeof(double));


  imins=itpiv;  /*INDICE DA VARIAVEL QUE SAI DA BASE*/
  maxi=0.;

  for(i=0;i<nci;i++){

    printf("c1[%d]=%d\n",i,c1[i]);
    if(c1[i]==0){

      for(j=0;j<nl;j++)
        p[j]=0.;

      p0=mata[i].icapa;

      /* Vou calcular o P(r) */
      while(p0!=NULL){
        p[p0->irow]=p0->value;
        p0=p0->next;
        }

      if(matt==NULL)
        calp(matrix,p);

      /* Vou calcular P'(r) = B(-1)*P(r)  = Tr*...*T1*P(r)   */
      t=matt;
      while(t!=NULL){    /* para cada uma das matrizes T's */
        tot=0.;
```

```
        p0=t->icapt;
        while(p0!=NULL){  /* para cada um dos elementos existentes na coluna de
          if(p0->irow==t->ncolt)
            tot=p[p0->irow]*p0->value;
          else
            p[p0->irow]+=p0->value*p[t->ncolt];
          p0=p0->next;
          }
        p[t->ncolt]=tot;
        t=t->next;
        }

      /* vou achar a vari vel a entrar na base */
      if(maxi!=0.){
        if(p[imins]<0. && ca[i]/p[imins] > maxi){
          imine=i;
          maxi=ca[i]/p[imins];
          printf("\t\tpivot=%f\n",maxi);
          getch();
          }
        }
      else
        if(p[imins]<0.){
          imine=i;
          maxi=ca[i]/p[imins];
          printf("\t\tmaxi=pivot=%f\n",maxi);
          getch();
          }
      }
    }

  if(maxi==0.){
    printf("*** nao tem soluao ***\n");
    getch();
    exit(1);
    }
  printf("sai:%d\tentra:%d\n",imins,imine);
  return(itdual(matrix,imins,imine));
}


int calp(matrix,p)
struct matriz *matrix;
double *p;
{
  int nl,i;
  int *xb;
  double gg;
  struct aa *mata;
  struct tabl *pp;

  mata=matrix->mata;
  xb=matrix->xb;
  nl=matrix->nl;
  gg=0.;

  for(i=0;i<nl-1;i++){
    for(pp=mata[xb[i]].icapa;pp->irow!=nl-1 && pp!=NULL;pp=pp->next);
    if(pp->irow==nl-1)
      gg+=p[i];
    }
  p[nl-1]=-gg;
  return (1);
}
```

```
int itdual(matrix,sss,eee)
struct matriz *matrix;
int sss;
int eee;
{

  struct aa *mata;
  int nl;
  int nci;
  int *xb;
  double *b;
  double *c;
  double *cb;
  double *ca;
  char *c1;
  struct tab1 *p0,*p1,*fi; /* apontadores necess rios */
  struct tt *t;      /* apontadores necess rios */
  int i,j;
  double *p=NULL;  /* vector usado no algoritmo */
  double tot;

  mata=matrix->mata;
  nl=matrix->nl;
  xb=matrix->xb;
  nci=matrix->nci;
  c1=matrix->c1;
  c=matrix->c;
  ca=matrix->ca;
  b=matrix->b;

  p=(double *)mymalloc(nl*sizeof(double));
  cb=(double *)mymalloc(nl*sizeof(double));


  /*CALCULA ITERAAO ...*/
  /* Vou calcular o P(r) */
  for(i=0;i<nl;i++)
    p[i]=0.;
  p0=mata[eee].icapa;
  while(p0!=NULL){
    p[p0->irow]=p0->value;
    p0=p0->next;
    }

  if(matt==NULL)
    calp(matrix,p);

  /* Vou calcular P'(r) = B(-1)*P(r)  = Tr*...*T1*P(r)   */
  t=matt;
  while(t!=NULL){    /* para cada uma das matrizes T's */
    tot=0.;
    p0=t->icapt;
    while(p0!=NULL){  /* para cada um dos elementos existentes na coluna de T !=
      if(p0->irow==t->ncolt)
        tot=p[p0->irow]*p0->value;
      else
        p[p0->irow]+=p0->value*p[t->ncolt];
      p0=p0->next;
      }
    p[t->ncolt]=tot;
    t=t->next;
    }

  /* calculo da matriz de transforma
                                    o T(r) */
```

```
  j=0;     /* j - n§ de elementos != de zero */

  if(matt==NULL)
    t=matt=(struct tt *)mymalloc(sizeof(struct tt));
  else
    t=(struct tt *)mymalloc(sizeof(struct tt));

  if(last!=NULL){      /* ligaao aos elementos da lista */
    t->last=last;
    last->next=t;
    }
  else
    t->last=NULL;    /* se for o 1§ elemento da lista */
  t->next=NULL;
  fi=p1=(struct tab1 *)mymalloc(sizeof(struct tab1));
  for(i=0;i<nl;i++){
    if(p[i]){
      if(i==sss)
        p1->value=1/p[sss];
      else
        p1->value=-p[i]/p[sss];
      p1->irow=i;
      p0=p1;
      p1=(struct tab1 *)mymalloc(sizeof(struct tab1));
      p0->next=p1;
      j++;
      }
    }
  p0->next=NULL;
  t->icapt=fi;
  t->nozet=j;
  t->ncolt=sss;
  t->entra=eee;
  last=t;

  c1[xb[sss]]=0; /*desmarco a vari vel que vai sair da base */
  c1[eee]=1;       /*marco vari vel que entra na base */
  xb[sss]=eee;  /* introduzo nova vari vel na base */

  for(i=0;i<nci;i++)
    printf(" c1[%d]=%d\n",i,c1[i]);

  /* Vou calcular (actualizar) b'= B(-1)*b = T(r)*...*T(1)*b */
  tot=0.;
  p0=last->icapt;              /* para a ultima das matrizes T's */
  while(p0!=NULL){             /* para cada um dos elementos existentes na coluna d
    if(p0->irow==last->ncolt)
      tot=b[p0->irow]*p0->value;
    else{
      b[p0->irow]+=p0->value*b[last->ncolt];
      if(b[p0->irow]<1.0e-6 && b[p0->irow]>-1.0e-6)
        b[p0->irow]=0.;
      }
    p0=p0->next;
    }
  b[last->ncolt]=tot;
  if(b[p0->irow]<1.0e-6 && b[p0->irow]>-1.0e-6)
    b[p0->irow]=0.;

  for(i=0;i<nl;i++)
    printf("b[%d]=%f\n",i,b[i]);
  getch();

  /*   vou inicializar Cb e Ca   */
  for(i=0;i<nl;i++){
    cb[i]=c[xb[i]];
```

```
    ca[i]=c[i];
    }
  for(;i<nci;i++)
    ca[i]=c[i];

  for(i=0;i<nci;i++)
    printf("\tCAi[%d]=%f\n",i,ca[i]);
  printf("\n");
  getch();

  for(i=0;i<nl;i++)
    printf("\tCBi[%d]=%f\n",i,cb[i]);
  printf("\n");
  getch();

  /* vou calcular C'a = Ca - Cb*B(-1)*A'   */
       /* Cb*B(-1) ---> Cb */

  t=last;
  while(t!=NULL){
    tot=0.;
    p0=t->icapt;
    while(p0!=NULL){
      tot+=cb[p0->irow]*p0->value;
      p0=p0->next;
      }
    cb[t->ncolt]=tot;
    t=t->last;
    }

  printf("\n");
  for(i=0;i<nl;i++)
    printf("CB[%d]=%f\n",i,cb[i]);
  printf("\n");
  getch();

       /* C'a = Ca - Cb*A' */
  for(i=0;i<nci;i++){
    while(c1[i])
      ca[i++]=0.;
    if(i<nci){
      p0=mata[i].icapa;
      while(p0!=NULL){
        ca[i]-=p0->value*cb[p0->irow];
        p0=p0->next;
        }
      }
    }

  for(i=0;i<nci;i++)
    printf("CA[%d]=%f\n",i,ca[i]);
  getch();

  matrix->xb=xb;
  matrix->ca=ca;
  matrix->c1=c1;
  matrix->b=b;

  if(itbneg(matrix))
    return(itd(matrix));
  else tes_sol(matrix);
}


int itbneg(matrix)
struct matriz *matrix;
```

```
{
  int nl,nci;
  int *xb;
  double *b;
  int i,j;
  double bxx;

  nl=matrix->nl;
  b=matrix->b;
  nci=matrix->nci;
  xb=matrix->xb;

  for(i=0;i<nl && b[i]>=0.; i++)
    ;
  if(i<nl){
    printf("Solucao ainda nao admissivel!\n\n");
    getch();
    bxx=b[i];
    itpiv=i;
    for(j=i+1;j>nl;j++)
      if(bxx>b[j]){
        bxx=b[j];
        itpiv=j;
        }
    return(1);
    }
  else{
    printf("Solucao ja admissivel!\n\n");
    sol=(double *)mymalloc(nci*sizeof(double));
    for(i=0;i<nci;i++)
     sol[i]=0.;

    for(i=0;i<nl;i++)
      if(xb[i]<nci)
        sol[xb[i]]=b[i];

    printf("Solu
                o  :\n");
    for(i=0;i<nci;i++)
      printf("%s = %f\n",tokens[i],sol[i]);
    getch();

    return(0);
    }
}
```