UNIVERSIDADE DO PORTO
FACULDADE DE ENGENHARIA
Departamento de Engenharia Eletrotécnica e Computadores

Estudo do Problema de Fluxo de Potência Óptimo Impreciso: - Programação Multiparamétrica - Programação Paramétrica

Relatório de Estágio

Ana Cristina da Silva Anacleto

1992/93

PRODEP

Parecer Científico sobre o estágio de

Ana Cristina da Silva Anacleto

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

REFERÊNCIA DO PROJECTO 4.3/1.107/92/93

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)

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.

FASE INICIAL

Uma primeira fase do meu estágio consistiu no estudo de métodos, conceitos e técnicas, indispensáveis para uma melhor compreensão de vários pontos exigidos no meu plano de trabalhos e também, para uma melhor adaptação às técnicas de programação usadas para o desenvolvimento do trabalho proposto. Este período, teve 4 pontos importantes a considerar:

1*MÉTODOS DE FACTORIZAÇÃO DE MATRIZES

Destaco os métodos directos, baseados na manipulação directa das equações, e os métodos iterativos, baseados na resolução das equações por aproximações sucessivas.

Usando um método directo, implementei, em C, o método da factorização à esquerda. Estes métodos são menos eficientes que métodos como da bi-factorização; porém, é o método adequado para manter factorizada a inversa da base em métodos de Programação Linear do tipo Simplex.

2*TÉCNICAS DE ESPARSIDADE

A obtenção da solução de um Problema Linear envolve directamente o cálculo da inversa de uma matriz. Esta terá características que tornam as técnicas de esparsidade muito eficientes, tanto em termos de espaço de memória, como de tempo de computação. Em geral, estas técnicas aplicam-se a matrizes de grande dimensão e com uma grande percentagem de elementos nulos (a percentagem de elementos não nulos, raramente excede 5% do total de elementos não nulos possíveis).

Para cálculo desta inversa, usei o método de factorização à esquerda, que consiste em construir uma sequência de n matrizes factores (n é a dimenção da matriz), tais que:

$$A^{-1}=T_n*T_{n-1}*...*T_2*T_1$$

onde T_i , i=1,...,n , difere da identidade na coluna i e A é a matriz da qual queremos a inversa.

3*ESTUDO DE CONCEITOS BÁSICOS DE CONJUNTOS IMPRECISOS (FUZZY SETS)

Conceito de imprecisão. Definição de FUZZY SETS e operações com fuzzy sets.

4*ESTUDO DO ALGORITMO DO SIMPLEX E DO DUAL SIMPLEX
EXTENSÃO À PROGRAMAÇÃO LINEAR FUZZY.

Tratamento de problemas de formulação linear em que os termos independentes são dados qualitativamente (por exemplo, a partir de declarações linguísticas) e traduzidos sob a forma de números imprecisos, com as respectivas funções de pertença.

ASPECTO GERAL DO TRABALHO

O trabalho elaborado pode ser aplicado no planeamento de um problema físico qualquer, que possa ser traduzido por um **modelo linear**. No nosso caso, destina-se a determinar qual a melhor decisão a tomar em problemas de consumo de energia e, mais concretamente, no *problema do fluxo de potência óptimo impreciso*. Essa decisão vai indicar qual a produção necessária, face a consumos exigidos e que, ao mesmo tempo, minimize o custo de produção.

Neste planeamento, idealizamos um modelo linear, que será o mais aproximado possível da realidade. O modelo possui um carácter não probabilístico. São utilizados conjuntos imprecisos (*FUZZY SETS*) para caracterizar, através de distribuições de possibilidades, os cenários de potências, que poderão ocorrer no futuro.

Ao tomar um modelo determinístico como ideal, corremos o risco dele falhar a longo prazo. Temos, por isso, que dar ao nosso modelo uma certa liberdade para que se possa tomar a melhor decisão e, ao mesmo tempo, permitir obter índices de risco, de modo que possa ser medido o *«grau de arrependimento»* que o planeador poderá sentir, após tomar uma decisão,se se vier a verificar que essa decisão não foi adequada, por o futuro ser diferente do esperado.

A introdução de imprecisões nos consumos, que podem ser números imprecisos triangulares ou trapezoidais, resulta simples e eficiente do ponto de vista computacional. Essa imprecisão, vai projectar-se quer no espaço de decisões, quer no espaço de atributos, que se tornam também imprecisos, o que nos traz um domínio impreciso para a solução.

O problema de **Fluxo de Potência Óptimo** pode ser considerado uma metodologia de optimização, através da qual se pretende identificar os valores das potências produzidas, sujeitas a restrições, que permitem optimizar um determinado

critério. Quando, pelo menos, uma potência produzida é representada por uma função de pertença, o problema será denominado de **Fluxo de Potência Óptimo Impreciso-FOPF.**

FASES DO ALGORITMO PARA TRATAMENTO DE IMPRECISÕES

A primeira etapa do algoritmo é constituída pela resolução de um problema determinístico de fluxo de potência óptimo, para identificar a solução óptima e admissível.

»»TRANSFORMAÇÃO DO PROBLEMA NUMA FORMULAÇÃO MULTI-PARAMÉTRICA

As incertezas associadas às potências de carga são integradas no problema adoptando-se uma *formulação multi-paramétrica*. Essa integração é feita, acrescentando ao vector $\underline{b}$, do problema linear, um conjunto de parâmetros, Δ, afectados de um coeficiente não nulo. Cada parâmetro tem uma gama de variação, que corresponde a desvios associados às funções de pertença do tipo das incertezas que se querem introduzir.

O problema é então formulado como sendo de programação linear multiparamétrica:

$$\min\ z=c^{t}X$$
$$\text{suj}\ AX=b+M\Delta$$
$$\Delta_{imin}<=\Delta_{i}<=\Delta_{imax}$$

»»IDENTIFICAÇÃO DE VÉRTICES

Foi necessário o desenvolvimento de um algoritmo eficiente, possuindo *índole Heurística,* que permite identificar um conjunto de vértices do hipervolume associado ao domínio das potências de carga possíveis.

A solução óptima do problema determinístico inicial, poderá ser não admissível para alguns valores dos desvios Δ. A identificação de regiões do hipervolume, onde a solução é não admissível, pode ser feita considerando que o valor de cada variável básica é dado por uma expressão linear, função dos Δ_i.

Para cada uma destas variáveis, é possível identificar o conjunto de valores de Δ_i que *minimiza e/ou maximiza* a expressão linear. Cada conjunto de valores de Δ_i identificado, corresponde a um vértice do hipervolume. Para cada vértice é possível calcular o valor das variáveis básicas do problema. Se o valor de pelo menos uma delas for negativo, poderá concluir-se que a solução óptima desse problema é não admissível numa região do hipervolume que contém o vértice em causa.

»»ESTUDO DE CADA VÉRTICE POR PROGRAMAÇÃO PARAMÉTRICA

Consiste em analisar cada um desses vértices através de um modelo de programação paramétrica.

A sua resolução, partindo da solução óptima e admissível, permite obter as soluções do problema de fluxo de potências óptimo associado às combinações de potências de carga situadas sobre o segmento OX, onde X é um vértice identificado e O é o ponto onde a solução é óptima. Este estudo parametrizado é feito integrando termos dependentes de um parâmetro δ, no problema inicial.

O problema de programação paramétrica toma a forma:

$$\min\ z=c^{t}X$$
$$\text{suj}\ AX=b+b'\delta$$
$$0.0<=\delta<=1.0$$

A resolução de cada problema de programação paramétrica permite obter, para cada variável, um conjunto de valores óptimos e, para cada elemento, está associado um grau de pertença que corresponde ao valor do parâmetro δ.

»»CÁLCULO DE ÍNDICES DE ROBUSTEZ E EXPOSIÇÃO

Os conceitos de exposição e robustez de um sistema, integram-se nas metodologias de análise de risco.

Podemos dizer que um sistema é **robusto**, se é possível alimentar as cargas sem recorrer à realização de corte de carga qualquer que seja o cenário de cargas. Na ausência de um plano robusto, o conceito de exposição permite quantificar o *«grau de arrependimento»* que planeador pode vir a experimentar.

O índice de robustez, assume o valor $(1-\delta)$, em que δ representa o menor corte de nível δ para o qual o sistema ainda consegue alimentar toda a incerteza das cargas, sem

ocorrer corte de carga. O índice de exposição, assume o valor δ e o índice de robustez corresponde ao complementar para 1.0 do índice de exposição.

Os conceitos de robustez e exposição poderão ser utilizados no planeamento da expansão de sistemas elétricos de energia, utilizando elementos da teoria dos conjuntos imprecisos de modo a identificar estratégias de reforço do sistema no âmbito de uma análise de risco.

CONCLUSÃO

A utilização de conjuntos imprecisos, para representar incertezas, permite estudar um conjunto de cenários representando, por exemplo, valores possíveis de potências activas de carga de um sistema eléctrico. A utilização de metodologias de carácter impreciso, garante uma avaliação mais eficiente e completa do comportamento do sistema.

Os algoritmos de solução desenvolvidos, revelam-se muito eficientes do ponto de vista de tempo de execução computacional. Por outro lado, a integração de conceitos de robustez e exposição, permite desenvolver estratégias de modo a diminuir o risco das decisões de planeamento.

COMENTÁRIOS

O estágio deu um contributo fundamental para a minha integração em equipas de investigação e desenvolvimento de software.

Foi um trabalho bastante interessante e útil, onde consegui aplicar conhecimentos de Investigação Operacional e de programação e, também, onde consegui alguma prática e experiência que, certamente, me será muito favorável num futuro emprego.

BIBLIOGRAFIA

1* SPARSITY

A. Brameller; R. N. Allan; Y. M. Hamam

2* FUZZY SET THEORY- and its applications

H. J. Zimmermann

3* FUZZY SETS AND SYSTEMS

Didler Dubois; Henry Prade

4* VÁRIOS ARTIGOS PUBLICADOS PELOS PROFESSORES VLADIMIRO MIRANDA E JOÃO PAULO TOMÉ SARAIVA

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
```

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

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

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

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

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

truct 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);
```

```
struct vertice{
  int *ver;
  struct vertice *next;
  double *vbas;
  };

int com_ver(void);
void mostra_ver(struct matriz *,int);
void cinver(struct matriz *);
void exp_linear(struct matriz *);
void matrizi(struct matriz *);
void le_inter(struct matriz *);
void ve_sinal(struct matriz *);
void analise(struct matriz *);
void ve_bas(struct matriz *);
void guar_verA(struct matriz *);
void guar_verNA(struct matriz *);
/*
le_pontos(struct matriz *);
ate_verA(struct matriz *);
ate_verNA(struct matriz *);
ve_adm(struct matriz *);
ve_alfa(struct matriz *);
pontos(struct matriz *);
next_alfa(struct matriz *);
actual(struct matriz *);
int bneg(struct matriz *);
int dual(struct matriz *,int,int);
int pivot(struct matriz *);
ind_rob(void);
*/
```

```
#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<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<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<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"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<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 tab1 *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);
    }
}
```

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

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

/* variaveis ja definidas */
extern struct tt *matt;

char matriz[20];
FILE *fich;

int np; /* n. de parametros a introduzir*/
double **m_par=NULL; /*matriz dos coeficientes dos parametros*/
double **m_parx=NULL; /*matriz_x dos coeficientes dos parametros*/
double **m_parxl=NULL; /*matriz_xl das exp_linearx*/
double *bxl;

/*apontadores necessarios*/
struct tab1 *p0;
struct tt *t;


matrizi(matrix)
struct matriz *matrix;
{
  int i,j;
  int nl,nlx;
  nl=matrix->nl;
  nlx=matrix->nlx;

  printf("\nQual o ficheiro dos coeficientes?\n");
  scanf(" %s",matriz);
  fich=fopen(matriz,"r+");
  fscanf(fich," %d",&np);
  printf("np=%d\n",np);
  printf("nl=%d\n",nl);

  m_par=(double **)mymalloc(nl*sizeof(double *));
  for(i=0;i<nl;i++)
    m_par[i]=(double *)mymalloc(np*sizeof(double));

  m_parx=(double **)mymalloc((nlx-nl)*sizeof(double *));
  m_parxl=(double **)mymalloc((nlx-nl)*sizeof(double *));
  for(i=0;i<nlx-nl;i++){
    m_parx[i]=(double *)mymalloc(np*sizeof(double));
    m_parxl[i]=(double *)mymalloc(np*sizeof(double));
    }

  bxl=(double *)mymalloc((nlx-nl)*sizeof(double));

  printf("Coeficientes:\n");
  for(i=0;i<nl;i++)
    for(j=0;j<np;j++){
      fscanf(fich," %lf",&m_par[i][j]);
      printf("m_par(%d,%d)=%f\n",i,j,m_par[i][j]);
    }
  printf("\n");
  for(i=0;i<nlx-nl;i++)
    for(j=0;j<np;j++){
      fscanf(fich," %lf",&m_parx[i][j]);
      printf("m_parx(%d,%d)=%f\n",i,j,m_parx[i][j]);
    }
  cinver(matrix);
}
```

```
/* Vou calcular B(-1)*m_par = Tr*...*T1*m_par */
cinver(matrix)
struct matriz *matrix;
{
  int i;
  double tot;
  t=matt;
  while(t!=NULL){    /* para cada uma das matrizes T's */
    for(i=0;i<np;i++){
      tot=0.;
      p0=t->icapt;
      while(p0!=NULL){  /* para cada um dos elementos existentes na coluna de T
        if(p0->irow==t->ncolt)
          tot=m_par[t->ncolt][i]*p0->value;
        else
          m_par[p0->irow][i]+=p0->value*m_par[t->ncolt][i];
        p0=p0->next;
        }
      m_par[t->ncolt][i]=tot;
      } /*endfor*/
    t=t->next;
    }
  exp_linear(matrix);
}


exp_linear(matrix)
struct matriz *matrix;
{
  int i,j;
  int nl;
  double *b;

  nl=matrix->nl;
  b=matrix->b;

  printf("\n");
  for(i=0;i<nl;i++){
    printf("\n");
    printf("%f  ",b[i]);
    for(j=0;j<np;j++)
      printf("+  %f G%d  ",m_par[i][j],j);
    }
  exp_linearx(matrix);
}


exp_linearx(matrix)
struct matriz *matrix;
{
  int i,j;
  int nl,nlx;
  int *xbx;
  double *b,*bx;
  char *c1;
  struct aax *matx;
  struct tabx *tx;

  nl=matrix->nl;
  nlx=matrix->nlx;
  b=matrix->b;
  c1=matrix->c1;
  bx=matrix->bx;
  matx=matrix->matx;
```

```
  xbx=(int *)mymalloc((nlx-nl)*sizeof(int));

  for(i=0;i<nlx-nl;i++)

    for(tx=matx[i].icapx;tx!=NULL;tx=tx->next){

      if(tx->itok>nci)
        xbx[i]=tx->itok;
      else
        if(c1[tx->itok]){
          for(j=0;xb[j]!=tx->itok && j<nl ;j++);
          if(j=nl){
            printf("ERRO!!\n");
            exit(1);
            }

}
```

```
#include <stdio.h>
#include <malloc.h>
#include "vert.h"
#include "simplex.h"
#include "common.h"

#define ADM  0
#define NADM 1

#define IGUAL 0
#define DIF   1

#define MIN 0
#define MAX 1

/* variaveis ja definidas */

int **interr=NULL; /*intervalos dos parametros*/
int *gamou=NULL; /*contem vertice a analisar*/
double *bsub=NULL;

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

FILE *fich;
char matriz[20];

int nva,nvna; /*n. de vertices ad. ou nad.*/

struct vertice *vad,*p0;  /*contem vertices admissiveis*/
struct vertice *vnad,*p1; /*contem vertices nao admissiveis*/


void le_inter(matrix)
struct matriz *matrix;
{
  int i;
  printf("\n\nQual o ficheiro dos intervalos?\n");
  scanf(" %s",matriz);
  fich=fopen(matriz,"r+");

  p1=vnad=NULL;
  p0=vad=NULL;
  interr=(int **)mymalloc(np*sizeof(int *));
  for(i=0;i<np;i++)
    interr[i]=(int *)mymalloc(2*sizeof(int));

  printf("\nIntervalos:\n");
  for(i=0;i<np;i++){
    if(fscanf(fich," %d %d",&interr[i][MAX],&interr[i][MIN])<2){
      printf("ERRO1:\n\tO ficheiro nao esta de acordo com o n§ de parametros!\n"
      exit(1);
      }
    printf("\t%d <= G%d <= %d\n",interr[i][MIN],i,interr[i][MAX]);
    printf("\n");
  }
  ve_sinal(matrix);
}


void ve_sinal(matrix)
struct matriz *matrix;
{
  int i,j;
  int nl,nlx;
```

```
  nva=0;
  nvna=0;
  nl=matrix->nl;
  nlx=matrix->nlx;

  gamou=(int *)mymalloc(np*sizeof(int));
  bsub=(double *)mymalloc(nl*sizeof(double));

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

/******VALORES DE GAMA QUE MAXIMIZAM CADA VARIAVEL******/
    printf("\nmax. a v.b. %d\n",i);
    for(j=0;j<np;j++){
      if(m_par[i][j]>0.)
        gamou[j]=interr[j][MAX];
      else{
        if(m_par[i][j]<0.)
          gamou[j]=interr[j][MIN];
        else
          gamou[j]=0;
        }
      printf("gamou[%d]=%d\n",j,gamou[j]);
      }
    if(com_ver())
      analise(matrix);

/******VALORES DE GAMA QUE MINIMIZAM CADA VARIAVEL******/
    printf("\nmin. a v.b. %d\n",i);
    for(j=0;j<np;j++){
      if(m_par[i][j]<0.)
        gamou[j]=interr[j][MAX];
      else{
        if(m_par[i][j]>0.)
          gamou[j]=interr[j][MIN];
        else
          gamou[j]=0;
        }
      printf("gamou[%d]=%d\n",j,gamou[j]);
      }
    if(com_ver())
      analise(matrix);

  } /*endfor*/
  if(nva>0){
    mostra_ver(matrix,ADM);
    getch();
    }
  if(nvna>0)
    mostra_ver(matrix,NADM);
}


int com_ver(){
  int i,j;
  struct vertice *pp;

  for(pp=vad;pp!=NULL;pp=pp->next){
    for(j=0;j<np && pp->ver[j]==gamou[j];j++)
      ;
    if(j==np){
      printf("IGUAL!!\n");
      getch();
      return(IGUAL);
      }
    }
```

```
  for(pp=vnad;pp!=NULL;pp=pp->next){
    for(j=0;j<np && pp->ver[j]==gamou[j];j++)
      ;
    if(j==np){
      printf("IGUAL\n");
      getch();
      return(IGUAL);
      }
    }
  printf("DIFERENTE\n");
  return(DIF);
}


void analise(matrix)
struct matriz *matrix;
{

  int i,j;
  int nl;
  double *b;
  nl=matrix->nl;
  b=matrix->b;

  for(i=0;i<nl;i++){
    bsub[i]=b[i];
    for(j=0;j<np;j++)
      bsub[i]+=m_par[i][j]*gamou[j];
    printf("bsub[%d]=%f\n",i,bsub[i]);
  }
  ve_bas(matrix);
  getch();
  return;
}


void ve_bas(matrix)
struct matriz *matrix;
{
  int i;
  int nl;
  nl=matrix->nl;

  for(i=0;i<nl && bsub[i]>=0.; i++)
    ;
  if(i<nl){
    nvna++;
    printf("Vertice nao admissivel!\n\n");
    guar_verNA(matrix);
    }
  else{
    nva++;
    printf("Vertice admissivel!\n\n");
    guar_verA(matrix);
    }
  printf("nvna=%d\n",nvna);
  printf("nva=%d\n\n",nva);
}

void guar_verA(matrix)
struct matriz *matrix;
{
  int i;
  int nl;
  nl=matrix->nl;
```

```
  if(nva==1){
    vad=(struct vertice *)mymalloc(sizeof(struct vertice));
    p0=vad;
    }
  else{
    p0->next=(struct vertice *)mymalloc(sizeof(struct vertice));
    p0=p0->next;
    }

  p0->ver=(int *)mymalloc(np*sizeof(int));
  p0->vbas=(double *)mymalloc(nl*sizeof(double));
  p0->next=NULL;
  for(i=0;i<np;i++){
    p0->ver[i]=gamou[i];
/*    printf("ver[%d]=%d\n",i,p0->ver[i]); */
    }
  for(i=0;i<nl;i++){
    p0->vbas[i]=bsub[i];
/*    printf("vbas[%d]=%f\n",i,p0->vbas[i]);*/
    }
}


void guar_verNA(matrix)
struct matriz *matrix;
{
  int i;
  int nl;
  nl=matrix->nl;

  if(nvna==1){
    vnad=(struct vertice *)mymalloc(sizeof(struct vertice));
    p1=vnad;
    }
  else{
    p1->next=(struct vertice *)mymalloc(sizeof(struct vertice));
    p1=p1->next;
    }

  p1->ver=(int *)mymalloc(np*sizeof(int));
  p1->vbas=(double *)mymalloc(nl*sizeof(double));
  p1->next=NULL;
  for(i=0;i<np;i++){
    p1->ver[i]=gamou[i];
/*    printf("ver[%d]=%d\n",i,p1->ver[i]); */
    }
  for(i=0;i<nl;i++){
    p1->vbas[i]=bsub[i];
/*    printf("vbas[%d]=%f\n",i,p1->vbas[i]);*/
    }
}
```

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

#define ADM  0
#define NADM 1

extern int np; /* n. de parametros a introduzir*/

extern struct vertice *vad;  /*contem vertices admissiveis*/
extern struct vertice *vnad; /*contem vertices nao admissiveis*/

void mostra_ver(matrix,id)
struct matriz *matrix;
int id;
{
  int i,j;
  int nl;
  struct vertice *pt;

  nl=matrix->nl;

  if(id==NADM){
    pt=vnad;
    printf("\n\nVertices nao admissiveis!\n");
    }
  else{
    if(id==ADM){
      pt=vad;
      printf("\n\nVertices admissiveis!\n");
      }
    else{
      fprintf(stderr,"Chamei mal a funcao 'mostra_ver'\n");
      exit(1);
      }
    }
  j=0;
  while(pt!=NULL){
    j++;
    printf("Vertice %d:\n",j);
    for(i=0;i<np;i++)
      printf("\tver[%d]=%d\n",i,pt->ver[i]);
    for(i=0;i<nl;i++)
      printf("\tvbas[%d]=%f\n",i,pt->vbas[i]);
    pt=pt->next;
    }
}
```

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

/* variaveis ja definidas */

extern int np; /* n§ de parametros introduzidos*/
extern double **m_par; /*matriz dos coeficientes dos parametros*/
double *ca_aux=NULL;
char *c1_aux=NULL;
double *b_aux=NULL;

extern struct vertice *vad; /*contem vertices admissiveis*/
extern struct vertice *vnad; /*contem vertices nao admissiveis*/

struct vertice *p0;
struct vertice *p1;

extern struct tt *last;
struct tt *last0;
struct tt **apo=NULL;

float **inter=NULL; /*intervalos dos parametros*/
float *g_aux=NULL;
float *gama=NULL;
float *Pmed=NULL;
float *dif=NULL;
float ir,ex;
float AL,alfa;

double *baux=NULL;

int lpiv;
int v;
float u;

FILE *fich;
char matriz[20];


le_pontos(matrix)
struct matriz *matrix;
{
  int i,nl,nci;
  nl=matrix->nl;
  nci=matrix->nci;

  printf("\n\nQual o ficheiro dos pontos?\n");
  scanf(" %s",matriz);
  fich=fopen(matriz,"r+");

  inter=(float **)mymalloc(np*sizeof(float *));
  for(i=0;i<np;i++)
    inter[i]=(float *)mymalloc(4*sizeof(float));

  printf("\nPontos lidos:\n");
  for(i=0;i<np;i++){
    if(fscanf(fich," %f %f %f %f",&inter[i][0],&inter[i][1],&inter[i][2],&inter[
      printf("ERRO1:\n\tO ficheiro nao esta de acordo com o n§ de parametros!\n"
      exit(1);
      }
    printf(" %f  %f  %f  %f\n",inter[i][0],inter[i][1],inter[i][2],inter[i][3]);
  }
  Pmed=(float *)mymalloc(np*sizeof(float));
  for(i=0;i<np;i++)
```

```
    Pmed[i]=0;

  gama=(float *)mymalloc(np*sizeof(float));
  g_aux=(float *)mymalloc(np*sizeof(float));
  baux=(double *)mymalloc(nl*sizeof(double));
  b_aux=(double *)mymalloc(nl*sizeof(double));
  cl_aux=(char *)mymalloc(nci*sizeof(char));
  ca_aux=(double *)mymalloc(nci*sizeof(double));
  dif=(float *)mymalloc(nl*sizeof(float));

  ate_vertA(matrix);
  ate_vertNA(matrix);
}


ate_vertA(matrix)
struct matriz *matrix;
{
  int i,j;
  int nl;
  double *b;
  v=0;
  nl=matrix->nl;
  b=matrix->b;

  printf("\nPARA VERTICES ADMISSIVEIS:\n");
  for(p0=vad;p0!=NULL;p0=p0->next){

    printf("\t\tVertice intermedio:\n");
    for(i=0;i<np;i++){
      if(p0->ver[i]==inter[i][0])
        g_aux[i]=inter[i][1];
      else
        g_aux[i]=inter[i][2];
      printf("g_aux[%d]=%f\n",i,g_aux[i]);
      }

    for(i=0;i<nl;i++){
      baux[i]=b[i];
      for(j=0;j<np;j++)
        baux[i]+=m_par[i][j]*g_aux[j];
      printf("baux[%d]=%f\n",i,baux[i]);
    }

    u=1.;
    if(ve_adm()){
      ve_alfa();
      getch();
      actual();
      if(pivot())
        continue;
      do{
        next_alfa();
        i=pivot();
        }while(alfa>0. && i==0);
      if(i) continue;

      for(i=0;i<np;i++){
        Pmed[i]=g_aux[i];
        printf("\tPmed[%d]=%f\t",i,Pmed[i]);
        g_aux[i]=p0->ver[i];
        printf("g_aux[%d]=%f\n",i,g_aux[i]);
        }
      for(i=0;i<nl;i++){
        baux[i]=p0->vbas[i];
        printf("baux[%d]=%f\n",i,baux[i]);
```

```
        }

      ve_alfa();
      u=alfa;
      printf("Credibilidade  %f\n",u);
      getch();
      if(pivot())
        continue;
      do{
        next_alfa();
        u=alfa;
        printf("Credibilidade  %f\n",u);
        i=pivot();
        }while(alfa>0. && i==0);
      if(i) continue;
      }

    apo[v-1]=last->next;
    last->next=NULL;

  }/*endfor*/
}

ate_vertNA(matrix)
struct matriz *matrix;
{
  int i,j;
  int nl;
  double *b;
  nl=matrix->nl;
  b=matrix->b;

  printf("\nPARA VERTICES NAO ADMISSIVEIS:\n");
  for(p1=vnad;p1!=NULL;p1=p1->next){

    printf("\t\tVertice intermedio:\n");
    for(i=0;i<np;i++){
      if(p1->ver[i]==inter[i][0])
        g_aux[i]=inter[i][1];
      else
        g_aux[i]=inter[i][2];
      printf("g_aux[%d]=%f\n",i,g_aux[i]);
      }

    for(i=0;i<nl;i++){
      baux[i]=b[i];
      b_aux[i]=b[i];
      for(j=0;j<np;j++)
        baux[i]+=m_par[i][j]*g_aux[j];
      printf("baux[%d]=%f\n",i,baux[i]);
    }

    u=1.;
    if(ve_adm()){
      ve_alfa();
      getch();
      actual();
      if(pivot())
        continue;
      do{
        next_alfa();
        i=pivot();
        }while(alfa>0. && i==0);
      if(i) continue;
      }/*endif*/
```

```
    else
      actual();

    for(i=0;i<np;i++){
      Pmed[i]=g_aux[i];
      printf("\tPmed[%d]=%f\t",i,Pmed[i]);
      g_aux[i]=p1->ver[i];
      printf("g_aux[%d]=%f\n",i,g_aux[i]);
      }
    for(i=0;i<nl;i++){
      baux[i]=p1->vbas[i];
      printf("baux[%d]=%f\n",i,baux[i]);
    }

    ve_alfa();
    u=alfa;
    printf("Credibilidade  %f\n",u);
    getch();
    if(pivot())
      continue;
    do{
      next_alfa();
      u=alfa;
      printf("Credibilidade  %f\n",u);
      i=pivot();
      }while(alfa>0. && i==0);
     if(i) continue;

    apo[v-1]=last->next;
    last->next=NULL;
  }/*endfor*/
}


ve_adm(matrix)
struct matriz *matrix;
{
  int i,nl;
  nl=matrix->nl;

  for(i=0;i<nl && baux[i]>=0.; i++)
    ;
  if(i<nl){
    printf("Ponto nao admissivel!\n\n");
    return 1;
    }
  else{
    printf("Ponto admissivel!\n\n");
    getch();
    return 0;
    }
}

ve_alfa(matrix)
struct matriz *matrix;
{
  int i,nl;
  float a_aux;
  alfa=0.;
  nl=matrix->nl;

  for(i=0;i<nl;i++){
    dif[i]=b_aux[i]-baux[i];
    if(dif[i]==0)
      printf("dif[%d]  0\n",i);
    else{
```

```
      a_aux=-baux[i]/dif[i];
      if(a_aux<=1. && a_aux>alfa)
        alfa=a_aux;
      }
    }
  AL=alfa;
  printf("alfa=%f\t\tAL=%f\n",alfa,AL);
  pontos(matrix);
}

pontos(matrix)
struct matriz *matrix;
{
  int i,j;
  int nl;
  nl=matrix->nl;

  for(i=0;i<np;i++){
    gama[i]=(1-alfa)*(g_aux[i]-Pmed[i]);
    printf("GAMA[%d]=%f\n",i,gama[i]);
    }

  for(i=0;i<nl;i++)
    b_aux[i]=baux[i];
  for(i=0;i<nl;i++){
    b_aux[i]+=dif[i]*alfa;
    if(b_aux[i]<1.0e-5 && b_aux[i]>-1.0e-5)
      b_aux[i]=0.;
    printf("b_aux[%d]=%.5f\n",i,b_aux[i]);
    }

  for(i=0;i<nl && b_aux[i]!=0.; i++)
   ;
  if(i<nl){
    lpiv=i;
    printf("lpiv=%d\n",lpiv);
    }
}


next_alfa(matrix)
struct matriz *matrix;
{
  int i;
  int nl;
  float a_aux;
  alfa=0.;
  nl=matrix->nl;

  for(i=0;i<nl;i++){
    dif[i]=b_aux[i]-baux[i];
    if(dif[i]==0)
      printf("dif[%d] e 0\n",i);
    else{
      a_aux=-baux[i]/dif[i];
      if(a_aux<AL && a_aux>alfa)
        alfa=a_aux;
      }
    }
  printf("alfa=%f\t\tAL=%f\n",alfa,AL);
  AL=alfa;
  pontos(matrix);
}


actual(matrix)
```

```
struct matriz *matrix;
{
  int i,nci;
  char *c1;
  double *ca;

  nci=matrix->nci;
  c1=matrix->c1;
  ca=matrix->ca;
  v++;
  printf("***\t***\tv=%d\n",v);
  last0=last;

  if(apo==NULL)
    apo=(struct tt **)mymalloc(sizeof(struct tt *));
  else
    apo=(struct tt **)myrealloc(apo,v*sizeof(struct tt *));

  for(i=0;i<nci;i++){
    c1_aux[i]=c1[i];
    ca_aux[i]=ca[i];
    }
}

ind_rob(){

  printf("u=%f\n",u);
  if(u==1.)
    ex=1.;
  else
    if(alfa>ex)
      ex=alfa;

  ir=1.-ex;

  printf("\tINDICE DE ROBUSTEZ = %f\n",ir);
  printf("\tINDICE DE EXPOSIAO = %f\n",ex);

}
```

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

/* variaveis ja definidas */
extern float u,ir,alfa;
extern int lpiv;
extern char *c1_aux;
double *cb_aux=NULL;
extern double *b_aux;
extern double *ca_aux;

extern struct tt *matt,*last,*last0;


int pivot(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;
  int nl,nci;
  struct aa *mata;

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

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

  imins=lpiv;/*INDICE DA VARIAVEL QUE SAI DA BASE->bp[lpiv]=0*/
  maxi=0.;

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

    printf("c1_aux[%d]=%d\n",i,c1_aux[i]);
    if(c1_aux[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_aux[i]/p[imins] > maxi){
          imine=i;
          maxi=ca_aux[i]/p[imins];
          printf("\t\tpivot=%f\n",maxi);
          }
        }
      else
        if(p[imins]<0.){
          imine=i;
          printf("ca_aux[%d]=%f\n",i,ca_aux[i]);
          printf("p[%d]=%f\n",imins,p[imins]);
          maxi=ca_aux[i]/p[imins];
          printf("\t\tmaxi=pivot=%f\n",maxi);
          }
      }
    }

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



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

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

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

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

/* 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(last0!=NULL){     /* ligaao aos elementos da lista */
  t->last=last0;
  last0->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;
last0=t;

c1_aux[xb[sss]]=0; /*desmarco a vari vel que vai sair da base */
c1_aux[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_aux[%d]=%d\n",i,c1_aux[i]);

/* Vou calcular (actualizar) b'= B(-1)*b = T(r)*...*T(1)*b */
```

```
  tot=0.;
  p0=last0->icapt;            /* para a ultima das matrizes T's */
  while(p0!=NULL){           /* para cada um dos elementos existentes na coluna d
    if(p0->irow==last0->ncolt)
      tot=b_aux[p0->irow]*p0->value;
    else
      b_aux[p0->irow]+=p0->value*b_aux[last0->ncolt];
    p0=p0->next;
    }
  b_aux[last0->ncolt]=tot;

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

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

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

  t=last0;
  while(t!=NULL){
    tot=0.;
    p0=t->icapt;
    while(p0!=NULL){
      tot+=cb_aux[p0->irow]*p0->value;
      p0=p0->next;
      }
    cb_aux[t->ncolt]=tot;
    t=t->last;
    }
  for(i=0;i<nl;i++)
    printf("CB_aux[%d]=%.2f\n",i,cb_aux[i]);
  getch();

       /* C'a = Ca - Cb*A' */
  for(i=0;i<nci;i++){
    while(c1_aux[i])
      ca_aux[i++]=0.;
    if(i<nci){
      p0=mata[i].icapa;
      while(p0!=NULL){
        ca_aux[i]-=p0->value*cb_aux[p0->irow];
        p0=p0->next;
        }
      }
    }

  for(i=0;i<nci;i++)
    printf("CA_aux[%d]=%.2f\n",i,ca_aux[i]);

  if(bneg(matrix))
    return(pivot(matrix));
  else return(0);
}


int bneg(matrix)
struct matriz *matrix;
{
```

```
  int i,j,nl;
  double bxx;
  nl=matrix->nl;

  for(i=0;i<nl && b_aux[i]>=0.; i++)
    ;
  if(i<nl){
    printf("Ponto ainda nao admissivel!\n\n");
    getch();
    bxx=b_aux[i];
    lpiv=i;
    for(j=i+1;j>nl;j++)
      if(bxx>b_aux[j]){
        bxx=b_aux[j];
        lpiv=j;
        }
    return(1);
    }
  else{
    printf("Ponto ja admissivel!\n\n");
    getch();
    return(0);
    }
}
```