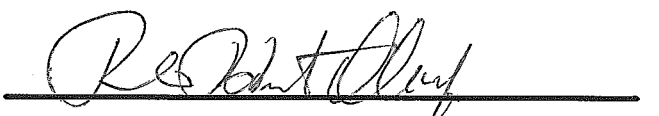
SIAMPLEX (SISTEMA INTERATIVO PARA APRENDIZAGEM DO MÉTODO SIMPLEX)

Walmir de Sousa Viana Junior

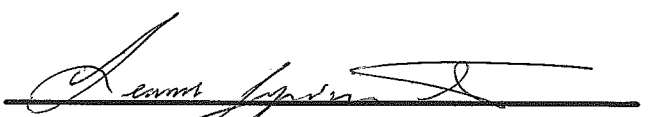
TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Cláudio Thomás Bornstein, D.Sc.
(Presidente)



Prof. Paulo Roberto de Oliveira, D.Sc.



Prof. Leonardo Junqueira Lustosa, Ph. D.

RIO DE JANEIRO, RJ - BRASIL
NOVEMBRO DE 1993

VIANA JUNIOR, WALMIR DE SOUSA

SIAMPLEX (Sistema Interativo para Aprendizagem do Método SIMPLEX) [Rio de Janeiro] 1993

VI, 88 p., 29.7 cm, (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 1993)

TESE - Universidade Federal do Rio de Janeiro, COPPE

1. Programação Linear

2. Algoritmo SIMPLEX

I. COPPE/UFRJ II. Título (série)

A meus pais

Walmir e Zoramyr

AGRADECIMENTOS:

- Ao professor Cláudio Thomás Bornstein, pela sua orientação e confiança em meu trabalho.
- Aos professores Antônio Clécio e Renato Craveiro, pelo incentivo à participação no curso de mestrado.
- Aos demais professores do curso, em especial, ao professor Paulo Roberto, pela contribuição dada à minha formação como aluno e cidadão.
- Aos amigos de todas as horas, Agostinho, Erivaldo, Evande, Marcelo e Paulo.
- Aos amigos das horas difíceis (e das agradáveis), Lucídio, Manoel e Plácido, que tanto me ajudaram na realização deste trabalho.
- A Walder, Zoraide e Amyr, pela ajuda e preocupação constante.
- A Walda, pela preocupação e ajuda que tornou a realização deste trabalho mais fácil.
- A Vládia, pelos inesquecíveis momentos que passamos juntos.
- A todos aqueles que direta ou indiretamente contribuíram para a realização deste trabalho.

Resumo da Tese apresentada à COPPE como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

SIAMPLEX (SISTEMA INTERATIVO PARA APRENDIZAGEM DO MÉTODO SIMPLEX)

Walmir de Sousa Viana Junior
Novembro de 1993

Orientador: Cláudio Thomás Bornstein
Programa: Engenharia de Sistemas e Computação

Este trabalho descreve um ambiente computacional denominado SIAMPLEX (Sistema Interativo para Aprendizagem do Método Simplex). O objetivo principal do SIAMPLEX é auxiliar, de forma interativa e didática, na aprendizagem do algoritmo SIMPLEX.

Analisa-se também, a mecânica do algoritmo SIMPLEX e técnicas empregadas no desenvolvimento do código computacional, para o tratamento dos erros numéricos introduzidos durante a execução do algoritmo.

Abstract of the Thesis presented to COPPE as partial fulfillment of the requirements for the Masters of Science degree

SIAMPLEX (INTERACTIVE SYSTEM FOR THE SIMPLEX METHOD)

Walmir de Sousa Viana Junior
November, 1993

Supervisor: Cláudio Thomás Bornstein
Department: Systems Engineering and Computer Science

We describe the SIAMPLEX code (Interactive System for the Simplex Method). The main purpose of the SIAMPLEX is to help the student to learn the SIMPLEX method.

We examine the SIMPLEX method and some techniques used in the development of the computer code for the treatment of the numerical errors that are generated.

ÍNDICE

CAPÍTULO 1

INTRODUÇÃO

Na área da Programação Linear, o algoritmo SIMPLEX ainda é o mais popular, apesar do desenvolvimento de novos algoritmos teoricamente superiores.Constatando, então, o seu uso e ensino generalizado em cursos de programação linear, nos propomos a desenvolver, neste trabalho um ambiente computacional de fácil utilização mesmo para quem não tem experiência com computadores, para o auxílio à aprendizagem do SIMPLEX.

Na etapa inicial do nosso trabalho, verificamos a existência de um pacote que implementa o algoritmo SIMPLEX com fins eminentemente didáticos, o CONPLEX , programa desenvolvido em linguagem FORTRAN no início dos anos oitenta. Encontramos também programas bastante eficientes e confiáveis do ponto de vista computacional, como LINDO[1991], MINOS[1983], MPSX e outros.Estes últimos, no entanto, não permitem uma interação adequada com o usuário bem como não ajudam no aprendizado da mecanica do algoritmo SIMPLEX.

Nosso trabalho, o SIAMPLEX (**S**istema **I**nterativo para **A**prendizagem do **M**étodo Sim**plex**), tem características semelhantes ao CONPLEX, porém oferece novas facilidades.

O SIAMPLEX destina-se ao auxílio na resolução de problemas de programação linear (PPL) para fases posteriores à modelagem do problema (atribuição exclusiva do usuário). O sistema dispõe de um editor de texto próprio onde o usuário digita o seu PPL podendo salvá-lo em disco para posterior

recuperação. Após a fase de digitação, este modelo será analisado para verificar possíveis inconsistências em sua sintaxe. Estando o modelo correto do ponto de vista sintático o sistema encarrega-se de guiar o usuário passo a passo, de uma forma interativa, desde a colocação do modelo na forma padrão até a obtenção de uma solução ótima, ou a indicação de sua inexistência.

O sistema permite a resolução de problemas com até noventa e nove variáveis e noventa e nove restrições. Além dessas restrições de tamanho, o limite para cada problema, também é dado pela quantidade de memória disponível na máquina, pois todos os quadros SIMPLEX, gerados a cada iteração, são armazenados na memória principal para permitir maior rapidez de acesso durante a execução do algoritmo.

No Capítulo 2, é apresentado o algoritmo SIMPLEX sem muito detalhamento, porém com a profundidade necessária ao desenvolvimento da teoria empregada em sua implementação computacional. Neste capítulo também é discutido o problema dos erros de arredondamento, que se acumulam durante a execução do algoritmo, podendo afetar seriamente a solução final, e técnicas para tratá-los.

No Capítulo 3, temos um enfoque eminentemente computacional. Nele são apresentados detalhes de implementação do sistema.

Finalmente, no Capítulo 4, apresentamos um tutorial de todo o sistema onde são explicados detalhes de todos os comandos.

CAPÍTULO 2

PROGRAMAÇÃO LINEAR

Uma da áreas de maior importância no campo da Pesquisa Operacional é sem dúvida a Programação Linear e nesta, o algoritmo SIMPLEX, apesar da existência de novos algoritmos teoricamente superiores, ainda é o mais popular.

Neste capítulo apresentaremos um pequeno resumo da Programação Linear dando ênfase ao uso do SIMPLEX. Na Seção 2.2 é feita uma breve descrição do mesmo.

Como veremos, o uso da aritmética de números de representação finita em computadores digitais, pode ocasionar acumulação de erros que comprometem a solução de problemas, resolvidos, através do SIMPLEX. Apresentaremos na Seção 2.3 a natureza destes erros, bem como técnicas que visam a recuperação dos problemas causadas por eles.

2.1 INTRODUÇÃO

O desenvolvimento da Programação Linear é considerado por muitos um dos grandes avanços científicos do meio do século. Isto foi possível graças ao trabalho de George Dantzig que desenvolveu em 1947 o método SIMPLEX para a resolução

de problemas lineares (Veja história do desenvolvimento do SIMPLEX em Dantzig[1982]). Um segundo fator teve também grande influência para o aprimoramento e popularização da Programação Linear, foi o rápido desenvolvimento e disseminação dos computadores eletrônicos.

Um número elevado de cálculos é necessário para a solução de problemas grandes e complexos, usualmente tratados em Programação Linear. Efetuá-los manualmente é inviável. Por isso o desenvolvimento dos computadores eletrônicos foi crucial.Atualmente problemas antes inimagináveis, por sua complexidade, são resolvidos em simples microcomputadores portáteis propiciando uma economia de milhares de dólares em companhias localizadas em várias partes do mundo.

Um dos tipos de aplicações mais comuns em Programação Linear é caracterizado pela solução de um problema de alocação de recursos limitados, entre atividades que por eles competem , da maneira mais eficiente possível. Este problema é geralmente representado por um modelo composto por expressões lineares.

O modelo do problema de Programaçao Linear é normalmente composto de uma função linear chamada FUNÇÃO OBJETIVO, que deve ser maximizada ou minimizada. As variáveis desta função, devem satisfazer um sistema de equações e (ou) inequações lineares que recebem o nome de RESTRIÇÕES DO PROBLEMA. Estas restrições referem-se normalmente às quantidades dos recursos disponíveis, ou a outras exigências que devem ser cumpridas pelo problema. A solução do problema, que será chamada SOLUÇÃO ÓTIMA, se existir, será aquela que maximizada ou minimiza a função objetivo.

Na resolução de um problema de Programação Linear são necessárias ao menos duas etapas:

- A modelagem do problema de acordo com a forma descrita anteriormente (veja Bregalda[1988], Hillier[1986] e Bazaraa[1990]).
- Emprego de um método para a resolução deste modelo. Apresentaremos a seguir o método SIMPLEX.

2.2 MÉTODO SIMPLEX

O método SIMPLEX tem provado ser bastante eficiente na resolução de problemas de Programação Linear (PPL) e, apesar de terem sido desenvolvidos novos algoritmos, ver Khachian[1979], Karmarkar[1984] e Gonzaga[1987], teoricamente superiores a este, ainda é o mais utilizado. No entanto para que o problema possa ser resolvido pelo SIMPLEX devemos colocá-lo numa determinada forma denominada FORMA-PADRÃO.

2.2.1 FORMA-PADRÃO DE UM PPL

DEFINIÇÃO 2.1:

Um PPL encontra-se na FORMA-PADRÃO quando o seu modelo estiver na seguinte forma:

$$\begin{cases} \text{MIN } Q(x) = \sum_{j=1}^{m} c_j x_j & (1) \\ \sum_{j=1}^{n} a_{ij} x_j = b_i \quad \text{onde } b_i \geq 0 \ (i = 1,2,\cdots,m) & (2) \\ x_j \geq 0 \qquad (j = 1,2,\cdots,n) & (3) \end{cases} \qquad (2.1)$$

Onde:

(1) Representa o função objetivo (que deve ser minimizada).

(2) São as restrições do problema.

(3) Condições de não negatividade.

O modelo (2.1) pode ser apresentado usando notação matricial da seguinte forma:

$$\begin{cases} \text{MIN } Q(x) = c^T x & (1) \\ Ax = b \quad \text{onde } b \geq 0 & (2) \\ x \geq 0 & (3) \end{cases} \qquad (2.2)$$

Onde:

$A \rightarrow$ matriz $m \times n$ constituída pelos elementos a_{ij} $i=1,2,\cdots,m$ e j=1,2,$\cdots$,n

$b \rightarrow$ vetor $m \times 1$ constituído pelos elementos b_i $i=1,2,\cdots,m$

$c^T \rightarrow$ vetor $1 \times n$ constituído pelos elementos c_j $j=1,2,\cdots,n$

$x \rightarrow$ vetor $n \times 1$ constituído pelos elementos x_j $j=1,2,\cdots,n$

REDUÇÃO DE UM PPL A FORMA-PADRÃO

Ao modelarmos um determinado PPL este pode não se encontrar inicialmente na forma (2.1). Apresentaremos a seguir as diversas formas que contrariam (2.1) e mostraremos uma transformação que contorna este problema.

1. Existência de variável x_i não positiva

   Basta substituir a variável $x_i \leq 0$ por sua simétrica, ou seja $x_i' = -x_i$ onde $x_i' \geq 0$, e substituir as ocorrências de x_i no problema por $-x_i'$.

2. Existência de variável x_i livre

   A variável x_i será dita livre se esta pode assumir um valor, positivo, negativo ou nulo. Quando isto acontecer podemos substituir as ocorrências de x_i no problema por duas variáveis x_i' e x_i'' ambas não negativas, tais que $x_i = x_i' - x_i''$.

3. A função objetivo é de maximização

   Basta substituirmos a função objetivo do problema por sua simétrica que deverá ser minimizada.

Suponhamos a função objetivo do problema

$$\text{MAX } Q(x) = c^T x$$

é só fazer

$$\text{MIN } Q'(x) = -c^T x$$

4. Existência de $b_i < 0$

Basta multiplicarmos a restrição i por -1

5. Existência de desigualdades

Estas podem ser transformadas facilmente em igualdades pela adição ou subtração de uma variável não negativa. Esta variável é chamada VARIÁVEL DE FOLGA.

a) Seja a inequação linear i: $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$

Basta somarmos a variável de folga x_{n+i} na inequação e teremos:

$$\sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = b_i$$

b) Seja a inequação linear i: $\sum_{j=1}^{n} a_{ij} x_j \geq b_i$

Basta subtrairmos variável de folga x_{n+i} na inequação e teremos:

$$\sum_{j=1}^{n} a_{ij} x_j - x_{n+i} = b_i$$

EXEMPLO 2.1:

Reduzir o seguinte PPL à forma padrão.

$$\begin{cases} \text{MAX } Q(x) = 3x_1 - 1x_2 + 2x_3 + 1x_4 - 2x_5 \\ \quad 1x_1 + 3x_2 - 2x_3 \quad - 1x_5 \geq 5 \\ \quad 2x_1 + 2x_2 \quad + 4x_5 \leq 2 \\ \quad + 1x_3 + 2x_4 + 1x_5 \geq -7 \\ \quad 1x_1 \quad - 3x_3 + 1x_4 \quad = 4 \\ x_1 \geq 0, x_2 \geq 0, x_3 \leq 0, x_4 \geq 0, x_5 \text{ Livre.} \end{cases}$$

Verificamos a existência de uma variável não positiva x_3 e outra livre x_5. Assim sendo devemos fazer $x_3 = -x_3'$ com $x_3' \geq 0$ e $x_5 = x_5' - x_5''$ onde $x_5' \geq 0$ e $x_5'' \geq 0$. Efetuada esta substituição teremos:

$$\begin{cases} \text{MAX } Q(x) = 3x_1 - 1x_2 - 2x_3' + 1x_4 - 2x_5' + 2x_5'' \\ \quad 1x_1 + 3x_2 + 2x_3' \quad - 1x_5' + 1x_5'' \geq 5 \\ \quad 2x_1 + 2x_2 \quad + 4x_5' - 4x_5'' \leq 2 \\ \quad - 1x_3' + 2x_4 + 1x_5' - 1x_5'' \geq -7 \\ \quad 1x_1 \quad + 3x_3' + 1x_4 \quad = 4 \\ x_1 \geq 0, x_2 \geq 0, x_3' \geq 0, x_4 \geq 0, x_5' \geq 0, x_5'' \geq 0. \end{cases}$$

Devemos agora transformar a função de maximizar em minimizar, ou seja Q(x) = -Q'(x), multiplicar a terceira restrição que apresenta $b_i < 0$ por -1, e introduzir as variáveis de folga necessárias à primeira, segunda e terceira restrições. Feito isto teremos o problema na forma-padrão:

$$\begin{cases} \text{MIN } Q'(x) = -3x_1 + 1x_2 + 2x_3' - 1x_4 + 2x_5' - 2x_5'' \\ \quad 1x_1 + 3x_2 + 2x_3' \quad - 1x_5' + 1x_5'' - F01 \quad = 5 \\ \quad 2x_1 + 2x_2 \quad + 4x_5' - 4x_5'' \quad + F02 \quad = 2 \\ \quad + 1x_3' - 2x_4 - 1x_5' + 1x_5'' \quad + F03 = 7 \\ \quad 1x_1 \quad + 3x_3' + 1x_4 \quad = 4 \\ x_1 \geq 0, x_2 \geq 0, x_3' \geq 0, x_4 \geq 0, x_5' \geq 0, x_5'' \geq 0, F01 \geq 0, F02 \geq 0, F03 \geq 0. \end{cases}$$

Colocando o problema acima na forma matricial (2.2) teremos:

$$A = \begin{vmatrix} 1 & 3 & 2 & 0 & -1 & 1 & -1 & 0 & 0 \\ 2 & 2 & 0 & 0 & 4 & -4 & 0 & 1 & 0 \\ 0 & 0 & 1 & -2 & -1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 3 & 1 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

$$x = \begin{vmatrix} x_1 \\ x_2 \\ x_3' \\ x_4 \\ x_5' \\ x_5'' \\ F01 \\ F02 \\ F03 \end{vmatrix} \qquad c = \begin{vmatrix} -3 \\ 1 \\ 2 \\ -4 \\ 2 \\ -2 \\ 0 \\ 0 \\ 0 \end{vmatrix} \qquad b = \begin{vmatrix} 5 \\ 2 \\ 7 \\ 4 \end{vmatrix}$$

2.2.2 ALGORITMO SIMPLEX

O SIMPLEX é um algoritmo que a partir de uma solução básica inicial (ver Definição 2.2 abaixo), gera a cada iteração uma nova solução básica cada vez melhor (se existir será aquela que decrementa a função objetivo), até que se obtenha uma que não pode ser melhorada.

DEFINIÇÃO 2.2:

Considere o sistema formado por pelas equações (2) e (3) de (2.2). Suponha que posto (A,b) = posto (A) = m. Depois de permutar as colunas da matriz A, seja $A=[B,N]$ onde B é uma matriz inversível.

A solução $x = \begin{bmatrix} x_B \\ x_N \end{bmatrix}$ para o conjunto de equações $Ax=b$, onde:

$$x_B = B^{-1}b \qquad \text{e} \qquad x_N = 0$$

é chamada solução básica do sistema. Se $x_B \geq 0$, então x é chamada solução básica viável do sistema.

Considerando o sistema (2.2) e supondo que seja possível determinarmos uma solução básica viável $\hat{x}$ podemos reescrevê-lo da seguinte forma:

$$\begin{cases} \text{MIN } Q(x) = c_N^T x_N + c_B^T x_B & (1) \\ Nx_N + Bx_B = b & (2) \\ x_B \geq 0, x_N \geq 0 & (3) \end{cases} \qquad (2.3)$$

Onde:

$I = \{j_1, \cdots, j_m\} \rightarrow$ conjunto de todos os índices das variáveis básicas (VB).

$J = \{j_{m+1}, \cdots, j_n\} \rightarrow$conjunto de todos os índices das variáveis não básicas (VNB).

$B = (a_1 \cdots, a_m) \rightarrow$matriz $m \times m$ constituída pelos vetores coluna de A referentes às VB.

$N = (a_{m+1}, \cdots, a_n) \rightarrow$matriz $m \times (n-m)$ constituída pelos vetores coluna de A referentes às VNB.

$c^B = \begin{vmatrix} c_1 \\ \vdots \\ c_m \end{vmatrix} \rightarrow$vetor $m \times 1$ constituído pelos coeficientes das VB na função objetivo.

$c^N = \begin{vmatrix} c_{m+1} \\ \vdots \\ c_n \end{vmatrix} \rightarrow$vetor $(n-m) \times 1$ constituído pelos coeficientes das VNB na função objetivo.

$x^B = \begin{vmatrix} x_1 \\ \vdots \\ x_m \end{vmatrix} \rightarrow$vetor $m \times 1$ constituído pelas VB.

$x^N = \begin{vmatrix} x_{m+1} \\ \vdots \\ x_n \end{vmatrix} \rightarrow$vetor $(n-m) \times 1$ constituído pelas VNB.

FORMA CANÔNICA DE UM PPL

Para facilitar a verificação das alterações introduzidas quando passamos de uma determinada solução básica viável $\hat{x}$ a outra colocamos nosso PPL em uma forma denominada FORMA CANÔNICA.

Se multiplicarmos a equação (2) de (2.3) por B^{-1} teremos

$$B^{-1}Nx_N + x_B = B^{-1}b \qquad (2.4)$$

A solução básica viável $\hat{x}$ é obtida pela anulação das VNB, isto é, fazendo $x_N = 0$. Assim $\hat{x}_B = B^{-1}b$. Fazendo $\hat{Y} = B^{-1}N$ e substituindo em (2.4) temos:

$$\hat{Y}x_N + x_B = \sum_{j \in J} \hat{y}_j x_j + x_B = \hat{x}_B \qquad (2.5)$$

Substituindo a equação (2.5) na função objetivo do modelo (2.3) temos:

$$\begin{aligned} Q(x) &= (c_N^T x_N) + c_B^T \hat{x}_B - c_B^T \hat{Y} x_N \\ &= c_B^T \hat{x}_B + ((c_N^T - c_B^T \hat{Y}))x_N \end{aligned} \qquad (2.6)$$

Façamos também

$$Q(\hat{x}) = c_B^T \hat{x}_B$$

e

$$\hat{z} = (c_B^T)\hat{Y} \Leftrightarrow \hat{z}_j = c_B^T \hat{y}_j \ \forall \ j \in J$$

onde

$$\hat{z} = (\hat{z}_{j_{m+1}}, \cdots, \hat{z}_{j_{m+n}})$$

Substituindo estes valores em (2.6) temos:

$$Q(x) - Q(\hat{x}) = (c_N^T - \hat{z})x_N = \sum_{j \in J} (c_j - \hat{z}_j)x_j \quad (2.7).$$

Dizemos que (2.5) e (2.7) compõem a forma canônica de um PPL para uma dada solução $\hat{x}$.

Outra forma de representação de um PPL é dada através do QUADRO SIMPLEX, apresentado a seguir.

	x_N^T	x_B^T	
x_B	N	B	b
	c_N^T	c_B^T	$Q(x)$

Quadro (2.1)

Ao multiplicarmos membro a membro as três últimas colunas da segunda linha, do Quadro (2.1), por B^{-1} e as subtrairmos das correspondentes na terceira linha teremos o seguinte quadro;

	x_N^T	x_B^T	
x_B	$\hat{Y}$	I	$\hat{x}_B$
	$c_N^T - \hat{z}$	O	$Q(x) - Q(\hat{x})$

Quadro (2.2)

Devemos observar que para aplicarmos o algoritmo SIMPLEX, não é necessário passarmos sempre de (2.3) para a forma canônica (2.5) e (2.7), pois esta, para uma solução básica viável x será sempre obtida a partir de uma solução $\hat{x}$ que já se encontra na forma canônica.

Mostraremos a seguir que é sempre possível obtermos uma solução básica viável inicial na forma canônica.

Se o PPL for constituído somente de desigualdades do tipo:

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i \text{ onde } b_i \geq 0$$

para gerarmos uma solução básica inicial na forma canônica, necessitamos somente adicionar variáveis de folga tornando-as VB e anular as restantes, que passam a ser as VNB do nosso problema. Entretanto, se o PPL possui restrições do tipo:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \quad \text{ou} \quad \sum_{j=1}^{n} a_{ij}x_j \geq b_i \quad \text{onde} \quad b_i \geq 0$$

devemos, após reduzí-lo à forma padrão, acrescentar variáveis artificiais positivas tornando-as, juntamente com as variáveis de folga das restrições que possuem sinal de menor ou igual "≤", as variáveis básicas iniciais do nosso problema, gerando assim uma solução básica artificial. Esta é chamada de Artificial pois não pertence ao conjunto de soluções viáveis do PPL original.

Teremos então o seguinte problema, que poderá ser solucionado com o auxilio do próprio SIMPLEX:

$$\begin{cases} \text{MIN } Q^a(x) = \sum_{i=1}^{m} x_i^a \\ Ax + x^a = b \\ x \geq 0 \\ x^a \geq 0 \end{cases}$$

Onde x^a e $Q^a(x)$ são respectivamente as variáveis artificiais e a função objetivo artificial.

Fazemos $x_i^a = 0$ para todas as restrições "≤" do PPL original.

Ao aplicarmos o algoritmo SIMPLEX para a solução do problema acima teremos ao final uma solução $\hat{x}^*$ que minimiza a função objetivo artificial. Dependendo do valor de $\hat{x}^*$ podemos ter:

1. $Q^a(x^*) = \sum_i x_i^{*a} = 0$

Isto implica em que todas as variáveis artificiais foram anuladas e que é possível determinar uma solução básica viável para o PPL original. Podemos encontrar os seguintes casos:

a)Todas as variáveis artificiais são VNB

Neste caso podemos eliminá-las assim como a função objetivo artificial

b)Existe variável artificial que é VB

Neste caso após eliminarmos todas as variáveis artificiais que são não básicas nos resta alguma variável x_j^a que é VB. Com isto podemos ter:

b.1)Todos os elementos y_{jk} da linha de x_j^a são nulos. Neste caso podemos eliminar x_j^a bem como a equação onde ela se encontra.

b.2)Existe algum y_{jk} da linha de x_j^a não nulo. Neste caso tornamos variável x_k VB e eliminamos x_j^a.

2. $Q^a(x^*) = \sum_i x_i^{*a} \geq 0$

Isto significa que o problema original não possui solução.

Já vimos que o algoritmo SIMPLEX parte de uma solução básica $\hat{x}$ e prossegue gerando novas soluções cada vez melhores. Só nos resta ver de que maneira é feita a escolha da variável que se tornará VNB e da que será a nossa nova VB, bem como as condições que nos indicam quando parar.

ESCOLHA DA NOVA VB:

Suponhamos $\hat{x}$ uma solução básica viável. Seja ainda $c_u - \hat{z}_u < 0$ para algum u $\in J$, tal que exista algum $\hat{y}_{iu} > 0$ com $i \in I$. Tomemos:

$$\frac{\hat{x}_t}{\hat{y}_{tu}} = mín\left(\frac{\hat{x}_i}{\hat{y}_{iu}}\right) \text{ tal que } \hat{y}_{iu} > 0$$

Assim $x_u = \hat{x}_t / \hat{y}_{tu}$ será a nossa nova VB, anulamos x_t que se tornará a nova VNB. Após realizarmos esta operação nosso problema não mais estará na forma canônica, por isso devemos efetuar uma nova operação denominada PIVOTEAMENTO, que visa a restauração da forma canônica. A operação de pivoteamento constitui-se em:

1. Dividir a linha pivô "t" pelo pivô $\hat{y}_{tu}$.
2. Subtrair de cada linha l_j a nova linha pivô multiplicada por y_{ju}.
3. Subtrair da linha correspondente à função objetivo a nova linha pivô multiplicada por $(c_u - \hat{z}_u)$.

CONDIÇÕES DE PARADA:

1. Se tivermos $c_u - \hat{z}_u < 0$ e $\hat{y}_u \leq 0$ para algum $u \in J$ a solução do PPL é impossível, seu valor pode ser tornado arbitrariamente pequeno..
2. Se tivermos $c_j - \hat{z}_j \geq 0$ para todo $j \in J$ a solução atual é ótima.

2.3- MÉTODO SIMPLEX E DECOMPOSIÇÃO LU

A medida que o método Simplex é aplicado e passamos de uma base a outra, freqüentemente ocorrem erros nos cálculos efetuados que podem afetar gravemente nossa solução final. Veremos a seguir a natureza destes erros e como minimizá-los, maiores detalhes podem ser encontrados em Gill[1990] e Press[1989].

2.3.1- ERROS NUMÉRICOS

Os computadores por sua própria natureza armazenam números não inteiros com uma precisão finita, isto é, muito provavelmente um número real para ser armazenado em um computador deve ser aproximado.

Os números reais são armazenados internamente na máquina em uma forma chamada de ponto flutuante. Dado um inteiro $b \geq 0$ podemos representar internamente qualquer escalar ξ da seguinte forma:

$$\xi = s \bullet M \bullet b^z$$

onde:

$s \rightarrow$ bit de sinal (+/-)

$M \rightarrow$ número real não negativo, chamado de mantissa (só possui um dígito à esquerda da vírgula)

$b \rightarrow$ base da representação (normalmente, $b=2$ e algumas vezes $b=16$)

$z \rightarrow$ número inteiro com sinal, chamado de expoente.

Além dos erros de arredondamento muitas operações aritméticas entre números com representação de ponto flutuante não são exatas, mesmo que os operadores estejam representados de forma exata.

Por exemplo, a soma de dois números em ponto flutuante, em uma máquina de base 2, é feita dividindo-se iterativamente por dois a mantissa do menor entre os dois (em magnitude) e incrementando-se seu expoente até que os dois operandos tenham o mesmo expoente.

Se os dois operandos diferem bastante em magnitude então o menor será substituído por zero, já que a mantissa M só consegue reter um número finito de dígitos. O Exemplo 02 em Pascal mostra este tipo de erro.

EXEMPLO 02:

```
var
  x,y:real;
begin
   x:=1.0;
   y:=0.000000000001;
   x:=x + y;
   writeln('x: ',x);
end.
```

Teremos como resultado; x:1.0000000000E+00

O menor número (em magnitude) na representação de ponto flutuante que quando somado ao número 1.0 (ponto flutuante), produz um resultado diferente de 1.0 é chamado de precisão da máquina ξ_M. O Exemplo 03 mostra como calcular ξ_M

EXEMPLO 03:

```
var
  x,y:real;
begin
   y:=1.0;
   repeat
     y:=y / 2;
     x:=1 + y;
   until x = 1;
   writeln('Precisao: ',y);
end.
```

O valor da precisão estará na variável "y".

Devemos observar que ξ_M não é o menor número em ponto flutuante que pode ser representado em uma determinada máquina, este número depende da quantidade de dígitos (bits) que dispomos para representar o expoente, enquanto ξ_M depende do número de dígitos da mantissa.

2.3.2- INVERSÃO DE MATRIZES USANDO DECOMPOSIÇÃO LU

Suponha que a matriz A $(m \times n)$ pode ser escrita como o produto de duas matrizes, da seguinte forma:

$$A = L \bullet U \qquad (2.8)$$

onde L é uma matriz triangular inferior (os elementos acima da diagonal são nulos) e U é uma matriz triangular superior (os elementos abaixo da diagonal são nulos). Expandindo a equação (2.8) temos:

$$\begin{vmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & \cdots & & a_{nn} \end{vmatrix} = \begin{vmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ l_{n1} & \cdots & & l_{nn} \end{vmatrix} \bullet \begin{vmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \ddots & u_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{nn} \end{vmatrix}$$

Para encontrarmos L e U mostraremos primeiramente como encontrar o elemento a_{ij} que será o produto da linha l_i pela coluna u_j de L e U respectivamente. Desta forma:

$$a_{ij} = l_{i1}u_{1j} + l_{i2}u_{2j} + \cdots + l_{ii}u_{ij} \text{ para } i < j \qquad (2.10)$$

$$a_{ij} = l_{i1}u_{1j} + l_{i2}u_{2j} + \cdots + l_{ii}u_{jj} \text{ para } i = j \qquad (2.11)$$

$$a_{ij} = l_{i1}u_{1j} + l_{i2}u_{2j} + \cdots + l_{ij}u_{jj} \text{ para } i > j \qquad (2.12)$$

O algoritmo de Crout soluciona eficientemente o sistema de equações definidos por (2.10), (2.11) e (2.12):

ALGORITMO DE CROUT:

Faça $l_{ii} = 1$, $i = 1,\cdots,n$

Para cada $j = 1,2,\cdots n$ faça

Para cada $i = 1,2,\cdots, j$ faça

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{ki} \qquad (2.13)$$

(Quando $i=1$ o somatório assume o valor zero)

Para cada $i = j+1, j+2,\cdots, n$ faça

$$l_{ij} = \frac{1}{u_{jj}}(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}) \qquad (2.14)$$

Podemos observar que os l's e u's que aparecem do lado direito das equações (2.13) e (2.14) já estão calculados quando precisamos deles, e que todo a_{ij} só é utilizado uma única vez. Isto significa que o l_{ij} ou u_{ij} correspondente pode ser armazenado em um local ocupado por a_{ij}. No final o algoritmo de Crout gera a matriz:

$$\begin{vmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ l_{21} & u_{22} & & u_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ l_{n1} & \cdots & l_{nn-1} & u_{nn} \end{vmatrix}$$

Lembramos que $l_{ii} = 1$, $i = 1,\cdots,n$

Um fator essencial para o algoritmo descrito acima é a existência de um pivô (u_{ij} na equação 2.14) não nulo em cada iteração. Vale ressaltar que a ocorrência de um pivô zero não implica na singularidade da matriz. Por exemplo a matriz B abaixo possui um pivô zero na primeira linha e não é singular

$$B = \begin{vmatrix} 0 & 2 & 2 \\ 1 & 1 & 2 \\ 8 & 10 & 12 \end{vmatrix}$$

Quando isto ocorre uma maneira natural de transpormos esta dificuldade é fazer uma permuta entre a linha que possui pivô nulo por outra cujo elemento da mesma coluna seja não nulo. Devemos no entanto, manter armazenada a ordem das permutações.

Outro problema que surge se deve a instabilidade numérica caso tenhamos pivôs de pequenas magnitudes. Isto pode ser evitado através de PIVOTEAMENTO PARCIAL.

O pivoteamento parcial consiste em selecionarmos na iteração k o elemento de maior magnitude, x_{tk} onde $t \geq k$, da coluna k para ser o nosso pivô. Se t for diferente de k então permutamos a linha t com a linha k . Em seguida dividimos a linha k pelo pivô x_{tk}.

EXEMPLO 04:

Decompor a matriz A abaixo como o produto de duas matrizes L (triangular inferior) e U (triangular superior).

$$A = \begin{vmatrix} 2 & 1 & -1 \\ 3 & -1 & 1 \\ 1 & 2 & 1 \end{vmatrix}$$

Iteração 1

Cálculo da coluna 1

$$L \bullet U = \begin{vmatrix} 2 & \\ 3 & \\ 1 & \end{vmatrix}$$

Pivoteamento parcial

$$L \bullet U = \begin{vmatrix} 3 & \\ 2/3 & \\ 1/3 & \end{vmatrix}$$

Iteração 2

Cálculo da coluna 2

$$L \bullet U = \begin{vmatrix} 3 & -1 \\ 2/3 & 5/3 \\ 1/3 & 7/3 \end{vmatrix}$$

Pivoteamento parcial

$$L \bullet U = \begin{vmatrix} 3 & -1 \\ 1/3 & 7/3 \\ 2/3 & 5/7 \end{vmatrix}$$

Iteração 3

Cálculo da coluna 3

$$L \bullet U = \begin{vmatrix} 3 & -1 & 1 \\ 1/3 & 7/3 & 2/3 \\ 2/3 & 5/7 & -15/7 \end{vmatrix}$$

Assim temos

$$L = \begin{vmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & 5/7 & 1 \end{vmatrix} \quad \text{e} \quad U = \begin{vmatrix} 3 & -1 & 1 \\ 0 & 7/3 & 2/3 \\ 0 & 0 & -15/7 \end{vmatrix}$$

CÁLCULO DA INVERSA

Para calcularmos a inversa de uma matriz A ($n \times n$ não singular) basta resolvermos o sistema da forma:

$$LUx_j = e_j \quad \text{j} = 1, \cdots, \text{n} \qquad (2.15)$$

onde e_j é um vetor unitário e a solução x_j é a j-ésima coluna de A^{-1}(inversa da matriz A). Assim A^{-1} é constituída pela união das colunas x_j :

$$A^{-1} = |x_1, x_2, \cdots, x_n|$$

Devemos lembrar que se houve permutação durante a decomposição LU, teremos que permutar da mesma forma os vetores e_j.

Podemos reescrever o sistema (2.15) da seguinte forma:

$$LUx_j = L(Ux_j) = e_j \qquad (2.16)$$

Solucionamos primeiramente para um vetor y_j tal que

$$Ly_j = e_j \qquad (2.17)$$

e depois resolvemos

$$Ux_j = y_j \qquad (2.18)$$

A vantagem em dividirmos o sistema (2.15) em (2.17) e (2.18) é que a solução de um sistema de equações triangulares é bastante trivial. Assim a equação (2.17) pode ser resolvida por substituição "FORWARD" da seguinte maneira

$$\begin{cases} y_{j_1} = \dfrac{e_{j_1}}{l_{11}} \\ \\ y_{j_i} = \dfrac{1}{l_{ii}}\left[e_{j_i} - \sum_{k=1}^{i-1} l_{ik} y_k \right] \quad i = 2,3,\cdots,n \end{cases} \qquad (2.19)$$

A equação (2.18) pode por sua vez ser solucionada por substituição "BACKWARD":

$$\begin{cases} x_{j_n} = \dfrac{y_{j_n}}{u_{nn}} \\ \\ x_{j_i} = \dfrac{1}{u_{ii}}\left[y_{j_i} - \sum_{k=i+1}^{n} u_{ik} x_{j_k} \right] \quad i = n-1, n-2, \cdots, 1 \end{cases} \qquad (2.20)$$

EXEMPLO 05:

Calcular a inversa da matriz A do Exemplo 04 a partir de sua decomposição LU.

Obs: Faremos somente para a primeira coluna da A, as demais serão calculadas de forma idêntica.

Usando (2.17) temos:

$$\overbrace{\begin{vmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ 2/3 & 5/7 & 1 \end{vmatrix}}^{L} \bullet \overbrace{\begin{vmatrix} y_{1_1} \\ y_{1_2} \\ y_{1_3} \end{vmatrix}}^{y_1} = \overbrace{\begin{vmatrix} 0 \\ 0 \\ 1 \end{vmatrix}}^{e_3}$$

* Foi usado e_3 do lado direito da igualdade pois durante a decomposição LU a primeira linha foi permutada com a terceira.

Usando (2.19) para o cálculo de Y_1 temos:

$$Y_1 = \begin{vmatrix} 0 \\ 0 \\ 1 \end{vmatrix}$$

Aplicando (2.18) temos que:

$$\overbrace{\begin{vmatrix} 3 & -1 & 1 \\ 0 & 7/3 & 2/3 \\ 0 & 0 & -15/7 \end{vmatrix}}^{U} \bullet \overbrace{\begin{vmatrix} x_{1_1} \\ x_{1_2} \\ x_{1_3} \end{vmatrix}}^{x_1} = \overbrace{\begin{vmatrix} 0 \\ 0 \\ 1 \end{vmatrix}}^{y_1}$$

Usando (2.20) para o cálculo de x_1 temos:

$$x_1 = \begin{vmatrix} 1/5 \\ 2/15 \\ -7/15 \end{vmatrix}$$

Após calcularmos x_2 e x_3 da forma descrita acima seus valores serão:

$$x_2 = \begin{vmatrix} 1/5 \\ -1/5 \\ 1/5 \end{vmatrix} \qquad x_3 = \begin{vmatrix} 0 \\ 1/3 \\ 1/3 \end{vmatrix}$$

Após a junção das colunas x_1, x_2 e x_3 temos a matriz inversa de A:

$$A^{-1} = |x_1 \quad x_2 \quad x_3| = \begin{vmatrix} 1/5 & 1/5 & 0 \\ 2/5 & -1/5 & 1/3 \\ -7/15 & 1/5 & 1/3 \end{vmatrix}$$

2.3.3 REINVERSÃO

Como vimos na seção (2.3.1) a aritmética de ponto flutuante pode introduzir erros a cada operação. Após um certo número de iterações do algoritmo SIMPLEX a matriz B^{-1} armazenada pode conter erros que afetam seriamente nossa solução. Devemos pois, recalculá-la bem como os outros valores do nosso quadro SIMPLEX.

Reescrevendo o quadro (2.2) utilizando a matriz B^{-1} nos fornece o seguinte quadro:

	x_N^T	x_B^T	
x_B	$B^{-1}N$	I	$B^{-1}b$
	$c_N^T - c_B^T B^{-1} N$	O	$Q(x) - c_B^T B^{-1} b$

Quadro (2.3)

Vemos pelo quadro (2.3) que é possível recalcular todos os valores da iteração k, bastando para isto termos armazenados os valores da iteração inicial. Mostraremos no exemplo abaixo como recalcular os valores armazenados pelo SIMPLEX em uma determinada iteração k.

EXEMPLO 06:

Seja o PPL dado pelo modelo:

$$\begin{aligned} Max\ Q'(x) &= 2x1 + 1x2 \\ & -1x1 + 1x2 \leq 1 \\ & -2x1 + 1x2 > -4 \\ & x1 \geq 0, x2 \geq 0 \end{aligned}$$

Colocando o modelo acima na forma padrão e acrescentando as variáveis de folga necessárias temos:

$$\begin{aligned} Min\ Q(x) &= -2x1 - 1x2 \\ & -1x1 + 1x2 + F1 = 1 \\ & +2x1 - 1x2 + F2 = 4 \\ & x1 \geq 0, x2 \geq 0, F1 \geq 0, F2 \geq 0 \end{aligned}$$

Colocando o problema acima na forma tabular do SIMPLEX temos o seguinte quadro :

Iteração 0

	x1	x2	F1	F2	
F1	*-1*	*1*	*1*	*0*	*1*
F2	*2*	*-1*	*0*	*1*	*4*
	-2	*-1*	*0*	*0*	*Q(x)*

Quadro (2.4)

Após a efetuarmos a primeira iteração do algoritmo SIMPLEX teremos o seguinte quadro:

	F2	x2	F1	x1	
F1	*1/2*	*1/2*	*1*	*0*	*2*
x1	*1/2*	*-1/2*	*0*	*1*	*3*
	1	*-2*	*0*	*0*	*Q(x)+16*

Quadro (2.5)

Para recalcularmos o quadro (2.5) devemos identificar:

$$x_B = \begin{vmatrix} F1 \\ x1 \end{vmatrix} \qquad x_N = \begin{vmatrix} F2 \\ x2 \end{vmatrix}$$

$$B = \begin{vmatrix} 1 & -1 \\ 0 & 2 \end{vmatrix} \qquad N = \begin{vmatrix} 0 & 1 \\ 1 & -1 \end{vmatrix}$$

$$c_B = \begin{vmatrix} 0 \\ -2 \end{vmatrix} \qquad c_N = \begin{vmatrix} 0 \\ -1 \end{vmatrix} \qquad b = \begin{vmatrix} 1 \\ 4 \end{vmatrix}$$

Calculando a inversa da matriz B temos:

$$B^{-1} = \begin{vmatrix} 1 & 1/2 \\ 0 & 1/2 \end{vmatrix}$$

Desta forma:

$$B^{-1}N = \begin{vmatrix} 1 & 1/2 \\ 0 & 1/2 \end{vmatrix} \bullet \begin{vmatrix} 0 & 1 \\ 1 & -1 \end{vmatrix} = \begin{vmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \end{vmatrix}$$

$$B^{-1}b = \begin{vmatrix} 1 & 1/2 \\ 0 & 1/2 \end{vmatrix} \bullet \begin{vmatrix} 1 \\ 4 \end{vmatrix} = \begin{vmatrix} 3 \\ 2 \end{vmatrix}$$

$$c_N^T - c_B^T B^{-1} N = \begin{vmatrix} 0 & -1 \end{vmatrix} - \begin{vmatrix} 0 & -2 \end{vmatrix} \bullet \begin{vmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \end{vmatrix}$$

$$= \begin{vmatrix} 0 & -1 \end{vmatrix} - \begin{vmatrix} -1 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 1 & -2 \end{vmatrix}$$

De fato obtivemos, como esperavamos, os mesmos valores do quadro (2.5).

CAPÍTULO 3

AMBIENTE SIAMPLEX

Neste capítulo é descrito detalhadamente o ambiente SIAMPLEX. São apresentados detalhes de implementação do programa principal e dos módulos que o compõem. Aqui algumas diferenças introduzidas na implementação do algoritmo SIMPLEX são mostradas e seus motivos discutidos.

3.1 INTRODUÇÃO

O SIAMPLEX (**S**istema **I**nterativo para **A**prendizagem do **M**étodo Sim**plex**) foi desenvolvido para ser usado com uma finalidade eminentemente didática e visa ensinar ao usuário de uma maneira interativa, através do uso de um microcomputador, a parte mecânica do algoritmo Simplex.

Fazemos questão de frisar que embora o SIAMPLEX permita ao usuário com conhecimento ínfimo de programação linear, particularmente do algoritmo Simplex, utilizá-lo e obter a solução de um determinado problema, esta não é sua finalidade. Para aqueles que estão somente interessados na solução final de um problema existem pacotes bastante eficientes e confiáveis no mercado.

O SIAMPLEX deve ser utilizado como uma ferramenta de apoio no aprendizado da mecânica do Simplex, livrando o usuário das tediosas operações aritméticas e indicando ações inválidas por ele tentadas. Por isso é fortemente aconselhado ao usuário que antes de utilizá-lo esteja seguro do que é o algoritmo Simplex, ou seja, domine a teoria do que existe por trás desta parte mecânica, isto implica em saber por que e quando se deve acrescentar variáveis de folga ou artificiais, o que significa introduzir uma variável na base e os motivos que levaram a sua escolha, etc.

Sem o auxilio de um computador, alguém que deseje solucionar um determinado PPL necessita colocar o modelo manualmente na forma padrão (Seção 2.2.1) e dispô-lo no quadro Simplex (Seção 2.2.2). A partir do quadro Simplex inicial o usuário, dependendo do problema, geralmente deve realizar uma grande quantidade de operações mecânicas (contas de somar, subtrair, multiplicar e dividir) para passar de um quadro ao próximo. Essas operações embora simples são desistimulantes e tornam-se uma fonte de muitos erros. Não raramente descuidos comprometem a solução final o que requer o recálculo de grande parte dos quadros. Essas dificuldades desestimulam o estudante a executar exercícios numéricos importantes para o entendimento e fixação dos detalhes do algoritmo SIMPLEX.

O SIAMPLEX encaixa-se justamente nesta fase posterior à modelagem do PPL. O sistema dispõe de um editor de texto onde o usuário digita o seu PPL podendo salvá-lo em disco para posterior recuperação. Após a fase de digitação este modelo será analisado para verificar possíveis inconsistências na sintaxe do mesmo. Estando o modelo correto do ponto de vista sintático (a modelagem do problema como não poderia deixar de ser, é atribuição exclusiva do usuário, que deve valer-se de sua experiência) o sistema encarrega-se de guiar o usuário passo a passo, de uma forma interativa, desde a colocação do modelo na forma padrão até a solução final, que poderá existir ou não.

A interação do sistema com o usuário é feita através de paineis de opções de fácil operação, que simulam paulatinamente todas as etapas que este executaria manualmente para a resolução de um determinado PPL. Além de guiar o usuário até a solução final o sistema monitora as escolhas de todas as opções a cada etapa. Caso esta não seja a correta em uma determinada situação, o sistema através de janelas de mensagens indica a causa do erro e não permite a passagem para uma etapa posterior sem que o erro seja corrigido, evitando assim que este só seja

detectado tardiamente pelo usuário, forçando-o a recomeçar todo o trabalho. Como função adicional o sistema monitora todos os erros cometidos a cada etapa e no fim da resolução do PPL apresenta separadamente uma estatística dos mesmos permitindo ao usuário uma avaliação do seu desempenho.

3.2 O SIAMPLEX

O SIAMPLEX foi implementado na linguagem Turbo Pascal versão 6.0 para ser executado em microcomputadores compatíveis com o IBM-PC. O sistema pode ser utilizado em máquinas equipadas com monitores CGA, HERCULES, VGA e SVGA sem problemas e requer ao menos uma unidade de disco flexível com 360 K-bytes de capacidade. Caso não sejam executados modelos que requeiram muitas iterações, o sistema executa em microcomputadores com 256 K-bytes de memória principal. Posteriormente serão apresentadas as formas de armazenamento do mesmo, que permitirão ao usuário estimar a quantidade de memória principal requerida para a execução de um determinado problema.

O uso da linguagem Pascal (utilizaremos Pascal deste ponto em diante para nos referirmos à Turbo Pascal versão 6.0) permitiu que o SIAMPLEX fosse implementado usando instrumentos da mesma que reforçam o conceito de programação estruturada. Entre estes está o uso de "units".

As units são a base da programação modular em Pascal. Elas são usadas para a criação de bibliotecas que podem ser incluídas em vários programas, sem a necessidade de incluir o código fonte e permitem dividir programas grandes (o SIAMPLEX possui mais de cinco mil linhas de código fonte) em módulos relacionados logicamente. Além de units, o sistema faz uso de elementos padrão da biblioteca de objetos do Pascal que serão descritos posteriormente.

O SIAMPLEX é composto de um programa principal e das units:

- UTelas (unit responsável pelas operações de leitura de dados, obtidos via teclado, e escrita na tela).

- UGlobal (unit que contém e inicializa variáveis tipos de dados e procedimentos que podem ser acessados por todas as units do sistema).

- UMensagem (unit responsável pelas interfaces de mensagens entre o sistema e o usuário, contém todas as mensagens de erro).

- UArquivo (unit responsável pelas operações de leitura e gravação de arquivos).

- UEditor (nesta unit é implementado um editor de texto que pode também ser utilizado para permitir apenas a leitura do texto).

- UAnalisa (nesta unit foi implementado um analisador sintático responsável pela verificação da sintaxe do PPL).

- UForPadr (unit responsável pela colocação do modelo original do problema na forma padrão).

- USimplex (unit responsável pela implementação do algoritmo SIMPLEX).

3.2.1 PROGRAMA PRINCIPAL

O programa principal utiliza as seguintes units: objects, dos, crt, UTelas, UGlobal, UArquivo, UEditor, UAnalisa, UForPadr e USimplex. Suas funções principais são a inicialização de variáveis e a coordenação das chamadas às units do sistema.

Podemos notar com isto a facilidade com que o uso de units nos permite modularizar uma grande quantidade de código. Se todo o sistema tivesse sido desenvolvido como um único bloco de código o esforço necessário ao seu desenvolvimento seria muito maior e as futuras alterações muito mais difíceis.

3.2.2 UNIT UTELAS

A unit UTelas é responsável por quase todas as operações que fazem uso da tela para enviar mensagens ao usuário, podendo capturar dados digitados pelo mesmo.

Devido ao grande número de operações que fazem uso da tela, principalmente pela unit UEditor, fomos levados a acessar diretamente a memória de vídeo, pois caso mostrássemos os caracteres através da função padrão "write" do Pascal, seria muito demorado e deselegante. Por exemplo, ao fazermos a tela rolar para cima no editor de texto veríamos a primeira linha sendo apagada e a segunda sendo colocada em seu lugar, este processo continuaria até a última linha da tela do editor.

Para que mantivéssemos a portabilidade do sistema, no que diz respeito ao tipo de monitor utilizado (color ou mono), fizemos uso da interrupção $10 do BIOS para identificar o tipo do vídeo. Se este tipo for "color" a memória de vídeo inicia-se no endereço $B800, se for do tipo "mono" inicia-se no endereço $B000.

Descreveremos a seguir os procedimentos mais importantes da unit UTelas que são freqüentemente usados pelas outras units do sistema.

Procedure SalvaTela(var BuffTela ; X1, Y1, X2, Y2 : integer);

Este procedimento move o conteúdo da memória de vídeo que corresponde à janela definida por X1 (coluna superior esquerda), Y1 (linha superior) e X2 (coluna inferior direita), Y2 (linha inferior) para a variável BuffTela. Para cada posição da tela, guardamos na variável BuffTela o caracter que se encontrava nesta posição bem como seus atributos de cor.

O objetivo principal deste procedimento é guardar as informações que estão em um determinado local da tela para posterior recuperação. Isto é bastante útil quando sobrepomos janelas em um mesmo local físico da tela, pois, ao mostrarmos uma mensagem de advertência ou painel de opções, o que havia naquele local será perdido. Por isso guardamos as informações e, quando a janela sobreposta for desativada as recuperamos sem a necessidade de as gerarmos novamente.

Procedure RestauraTela(var BuffTela; X1, Y1, X2, Y2 : integer);

Este procedimento move o conteúdo da variável BuffTela diretamente para a memória de vídeo. Como vimos anteriormente o conteúdo da variável BuffTela é uma janela de tela que foi armazenada pelo procedimento **SalvaTela**. Da mesma forma que este procedimento, X1, Y1, X2 e Y2, representam as coordenadas da janela que serão mostradas na tela.

Procedure MostraNCarAtrib(S : string80; NumHi:, HiAtrib, X , Y : integer);

Este procedimento tem por objetivo mostrar na posição X (coluna), Y(linha) da tela a variável S, com os NumHi primeiros caracteres na cor HiAtrib e os demais na cor corrente em uso.

O procedimento **MostraNCarAtrib** é útil para destacar em um painel de opções uma letra que poderá ser utilizada pelo usuário para ativar uma determinada opção sem que necessite deslocar-se até a mesma. Tal letra é denominada tecla de atalho.

Procedure DesenhaMoldura(X, Y, Larg, Alt :integer; Tit : string80; AtribBorda, AtribTit : integer; TipoMold : char);

Este procedimento desenha uma moldura com Larg caracteres de largura e Alt linhas de altura, cujo canto superior esquerdo está na coluna X e na linha Y da tela. Na parte superior da moldura será colocado um título dado pelo valor da variável Tit. As variáveis AtribBorda e AtribTit indicam respectivamente os atributos de cor da borda da moldura e do título da mesma. A variável TipoMold indica o tipo de cercadura da moldura: Se o valor de TipoMold for "D" a moldura será formada por traços duplos, caso contrário, terá traços simples.

Function UsaPainel(TipoHV : char; QualPainel : TipoPainel; EspacoHor : byte; ConjSai : Conj; NumItens, LargBarra, CorBarra : integer; EscolhaPainel : string80) : char;

Está função é uma das principais do sistema. Ela é que coordena a colocação dos paineis de opções e monitora a seleção das mesmas. A função retorna um caracter que indica a ordem da opção selecionada dentro do painel.

A variável TipoHV indica se o painel será disposto na posição horizontal "H", (neste caso a variável EspacoHor contém o valor do espaçamento entre as opções do painel), ou vertical "V". A variável QualPainel indica qual painel de opções será mostrado. A variável ConjSai indica quais permitem ao usuário deixar um determinado painel de opções. A variável NumItens informa o número de itens a serem exibidos em um painel de opções. As variáveis LargBarra e CorBarra indicam respectivamente a largura da barra que aparecerá em vídeo reverso e sua cor. Esta barra tem como função indicar qual a opção do painel será selecionada. A variável EscolhaPainel contém um conjunto de letras que são as opções de atalho do painel de opções.

Se o painel for disposto horizontalmente o usuário poderá mover a barra de seleção através das setas "→" (para a direita) e "←" (para a esquerda). Caso esteja disposto verticalmente a barra de seleção será movida através das setas ,"↓" (para baixo) e "↑" (para cima). A escolha de uma determinada opção pode ser feita de duas formas: Colocando a barra de seleção na opção desejada e teclando "ENTER" ou através da letra de atalho (quando esta existir estará destacada das demais e deverá ser única em um mesmo painel).

3.2.3 UNIT UMENSAGEM

A unit UMensagem é responsável por toda a interface de mensagens entre o sistema e o usuário. Existem basicamente dois tipos de mensagens para o usuário, são elas:

Comunicação de erro → indica que o usuário cometeu algum erro e explica sua causa.

Solicitação de ação → mostra uma mensagem e solicita alguma ação a ser tomada pelo usuário.

A interface da unit UMensagen com as demais units do sistema é feita através de três procedimentos, a saber:

♦ As mensagens estão aqui da forma que aparecem na tela.

Procedure MensErro(Errom :byte);

Este procedimento é utilizado pela unit UAnalisa e indica qual erro ocorreu durante a análise sintática do modelo. A variável Errom indica o número do erro ocorrido, e dependendo deste é mostrada uma mensagem. Os números de erros possíveis e suas respectivas mensagens são:

01:' Caracter invalido';

02:' Max/Min esperado';

03:' Numero Invalido';

04:' Operador Invalido';

05:' Operador Esperado';

06:' Operador ou numero esperado';

07:' Variavel grande demais';

08:' Variavel repetida';

09:' Variavel reservada';

10:' Caracter invalido na funcao objetivo';

11:' Ja existe comentario aberto';

12:' Fim de arquivo inesperado';

13:' Comentario nao aberto';

14:' RHS invalido';

15:' Res ou Operador esperado';

16:' Palavra Reservada';

17:' Fim ou Operador Esperado';

18:' Numero esperado';

19:' Numero de Variaveis grande demais';

20:' Problema Correto';

21:' Divisao por Zero';

22:' Caracter invalido apos RES ou RHS';

23:' Problema nao possui restricao';

24:' Numero de Restricoes Grande Demais';

Procedure PrintSubTela(Col, Lin, Larg, Alt : byte; Tit : string80; CodMens : byte; **var** RetMens : char);

Este procedimento é utilizado por todas as outras units do sistema e pode indicar um erro ocorrido ou uma ação que deve ser tomada pelo usuário.

As variáveis Col e Lin indicam a coluna e a linha da tela onde será, colocado o canto superior esquerdo de uma janela que possui Larg colunas de largura e Alt linhas de altura. Esta janela apresenta em sua linha superior um título, dado pelo valor da variável Tit. A variável RetMens retorna um valor "S" (Sim) ou "N" (Não) caso a variável CodMens seja maior do que 150 e menor do que 200, ou, "S", "N" ou "ESC" se CodMens for maior do que 200.

Quando da colocação das mensagens, aquelas que apresentarem códigos de 100 a 150 serão acompanhas de um sinal sonoro (BELL). A mensagem, ou ação a ser tomada pelo usuário dependem do valor de CodMens. Estes valores e suas respectivas mensagens são dados por:

1:'MODELO CORRETO';

2:'Variavel ' + Mens + ' deve deixar a base';

100:'Nao existe arquivo de edicao aberto.';

101:'Linha Grande Demais';

102:'Variavel de Folga Deve Ser Positiva.';

103:'Variavel de Folga Deve Ser Negativa.';

104:'Restricao nao precisa de Variavel de Folga.';

105:'Funcao deve ser transformada em Min.';

106:'Funcao ja esta na Forma Padrao.';

107:'Restricao deve se multiplicada por -1.';

108:'RHS da restricao ja e positivo.';

109:'Restricao nao necessita de Variavel Artificial';

110:'Restricao necessita de Variavel Artificial positiva';

111:'Variavel '+ Mens + ' ja e basica.';

112:'Variavel '+ Mens + ' nao pode entrar na base.';

113:'Variavel '+ Mens + ' deve sair da base.';

114:'Variavel '+ Mens + ' nao pode sair da base.';

115:'Problema deve ser colocado na Forma Padrao antes de ser Executado';

116:'Numero de variaveis grande demais. PROGRAMA SERA TERMINADO.';

117:'Problema deve iniciado pela FASE 1';

118:'Problema nao necessita da FASE 1';

119:'FASE 1 completada';

120:'FASE 1 nao completada.';

121:'FASE 2 completada';

122:'FASE 2 nao completada.';

123:'Fim da FASE 1. Solucao Ilimitada.';

124:'Fim da FASE 2. Solucao Ilimitada.';

125:' FASE 1 nao completada. Funcao Artificial <> 0.';

126:'FASE 1 nao completada. Existem Variaveis Artificiais na base.';

130:'Numero invalido';

131:'Arquivo inexistente';

151:'Ja existe arquivo de edicao aberto. Abre novo?';

152:'Modelo Correto. Colocar na Forma Padrao?';

153:'Todas as variaveis sao >= 0 ?';

154:'Inicia pela FASE 1 ?';

160:'Deseja Impressao dos quadros ?';

161:'Imprime Quadros ?';

201:'Fim da FASE 1 ?';

202:'Fim da FASE 2 ?';

- A variável "MENS" das mensagens de erro 02, 111, 112, 113 e 114 contém o nome da variável que ocasionou o erro.

Procedure PrintRetMens(Col, Lin, Larg, Alt : byte; Tit : string80; CodMens : byte; **var** RetMens : string12);

É semelhante ao procedimento **PrintSubTela** visto anteriormente. A diferença entre os dois é que o procedimento **PrintRetMens** retorna o nome de um arquivo na variável RetMens. É usado pela unit UArquivo.

3.2.4 UNIT UARQUIVO

A unit UArquivo é responsável pelas operações de leitura e gravação dos arquivos manipulados pela unit UEditor. Nela são também selecionados os arquivos **.PLX** (arquivos padrão dos problemas digitados no SIAMPLEX) do diretório corrente.

Para tornar mais fácil e rápida a seleção dos arquivos que contém os problemas do usuário, estes devem apresentar a terminação **.PLX**. Assim quando for solicitada a abertura de algum arquivo, os nomes de todos aqueles localizados no diretório corrente, que possuírem esta terminação serão mostrados em uma janela. Caso o usuário deseje abrir um arquivo que não possua esta terminação, basta selecionar na janela de nomes a opção "..\", e digitar o nome desejado.

Os arquivos **.PLX** estão gravados no formato texto, assim é possível gerar um arquivo **.PLX** em outro editor de texto que não seja o editor do sistema.

Na unit UArquivo é feita a conversão do arquivo de entrada para que este possa ser armazenado na variável PColecLin da unit UEditor (ver Seção 3.2.5). Da mesma forma os dados de PColecLin devem ser convertidos para o formato de arquivo "text" do Pascal. Esta é a principal função da unit UArquivo.

3.2.5 UNIT UEDITOR

A unit UEditor é uma das mais importantes do SIAMPLEX. Nela é implementado um editor de texto no qual o usuário digita o seu PPL. Este editor é usado em todas as outras fases para permitir a visualização das inclusões de variáveis de folga e artificiais feitas pela unit UForPadr, bem como a visualização dos quadros Simplex calculados na unit USimplex..

Quando o modelo está sendo analisado e a unit UAnalisa encontra algum erro sintático no mesmo, o controle é retornado à unit UEditor. Esta por sua vez posiciona o cursor do erro e apresenta uma mensagem explicando o motivo do mesmo.

Como vimos a unit UEditor além de ser um editor de texto é utilizada para que possamos navegar em um determinado texto. O editor permite que o arquivo de texto possua no máximo 255 colunas e 1500 linhas.

COMANDOS DO EDITOR:

Os comandos básicos do editor de texto são:

Enter Insere uma nova linha e move o cursor da posição corrente (na parte inferior da tela do editor existem dois números separados por ":". O primeiro indica a linha onde encontra-se o cursor e o segundo sua coluna), para uma linha abaixo e o desloca para a primeira coluna. Se houver algum texto, na mesma linha, após a posição onde encontra-se o cursor a linha será separada neste ponto e o texto irá para a nova linha.

BackSpace Apaga o caracter que antecede àquele onde encontra-se o cursor . Se houver texto na mesma linha após a coluna na qual encontra-se o cursor, este será deslocado para a esquerda uma posição. No final o cursor estará posicionado uma casa a esquerda de sua posição original.

Del Apaga o caracter que encontra-se na posição do cursor. Se houver texto na mesma linha após a coluna na qual encontra-se o cursor, este será deslocado para a esquerda uma posição.

→ Move o cursor uma coluna para a direita da posição atual. Se este estiver na margem direita da janela e a coluna do texto for menor do que 254 , todo o texto será movido para a esquerda.

← Move o cursor uma coluna para a esquerda da posição atual. Se este estiver na margem esquerda da janela e a coluna do texto for maior do que 1, todo o texto será movido para a direita

↑ Move o cursor 1 linha para cima da posição atual. Se este estiver no topo da janela e a linha corrente for maior do que 1 todo o texto será movido 1 linha para baixo.

↓ Move o cursor 1 linha para baixo da posição atual. Se este estiver no parte inferior da janela e não estivermos no fim do texto todo o texto será movido 1 linha para cima.

Todos os caracteres digitados no editor, quando este está em modo de edição, que correspondem a ordem de 34 a 254 na tabela de códigos ASCII serão inseridos na posição atual do cursor.

A interface da unit UEditor é feita através do procedimento **Editor** descrito a seguir.

Procedure Editor(var PColecLin: PCollection; **var** ColJanEdit, LinJanEdit, ColArqEdit, LinArqEdit :integer; ErroSintatico ,SoVeTexto : boolean; ITH, TTH : byte; Tit : string80 ; TipoMoldura : char);

A variável PColecLin é a base de todo o procedimento Editor. Ela é do tipo PCollection, que é um ponteiro para um objeto TCollection (objeto da biblioteca do Pascal).

O objeto TCollection é um tipo que nos permite implementar qualquer coleção de itens, incluindo outros objetos. O conceito do TCollection é mais geral do que os tradicionais tipos de dados **vetor**, **conjunto**, e **lista**. O objeto TCollection pode mudar de tamanho dinamicamente em tempo de execução. Este foi um dos fatores que mais nos influenciou na sua escolha para a implementação da unit UEditor pois não há possibilidade de saber com antecedência ao editar um determinado texto (problema) qual será o seu tamanho.

No nosso caso o item que usamos para ser armazenado na variável PColecLin foi um ponteiro para uma "string" de 256 caracteres, este ponteiro é automaticamente alocado através do objeto Pstring. Cada "string" armazenada em PColecLin representa uma linha do texto.

Para uma melhor compreensão do procedimento **Editor** descreveremos rapidamente alguns campos e métodos do objeto TCollection. Maiores detalhes podem ser obtidos em Turbo Pascal[1990].

OBJETO TCOLLECTION

Campos:

Count : integer;

Indica o número corrente de elementos armazenados na "collection".

Delta : integer

Indica o número de itens que devemos acrescentar à lista **Items** todas as vezes que ela estiver cheia. Se **Delta** for zero, a "collection" não pode crescer além do tamanho dado por **Limit**.

Items : PItemList

É um ponteiro para um vetor de ponteiros.

Limit : integer

Indica o número de elementos alocados na lista de **Items**".

Métodos:

Constructor Init(Alimit, Adelta : integer);

Cria uma "collection" com **Limit** dado por **Alimit** e **Delta** dado por **Adelta**.

Procedure AtInsert(Index : integer; Item : Pointer);

Insere **Item** na posição dada por **Index** e move os itens seguintes para baixo uma posição. Se **Index** for menor do que zero ou maior do que **Count** ocorre um erro. Se **Count** for igual a **Limit** antes da chamada do procedimento **AtInsert**, o tamanho alocado da "collection" é expandido de **Delta** itens. Se isto não for possível ocorre um erro.

Procedure AtPut(Index : integer; Item : pointer);

Substitui o item que se encontra na posição dada por **Index** pelo item dado por **Item**. Se o valor de **Index** for menor do que zero ou maior ou igual a **Count** ocorre um erro.

Procedure AtDelete(Index : integer);

Exclui o item que está na posição dada por **Index** e move todos os outros itens que o sucedem uma posição para cima. A variável **Count** é decrementada de um. Se **Index** for menor do que zero ou maior ou igual a **Count** ocorre um erro.

Procedure DeleteAll;

Apaga todos os itens da "collection".

Voltando a **Procedure Editor**, as variáveis ColJanEdit e LinJanEdit indicam respectivamente a coluna e a linha onde está ou será posicionado o cursor na janela de edição. A posição do cursor é relativa às coordenadas do canto superior esquerdo da janela do editor, que poderá estar localizada em diversas partes da tela.

As variáveis ColArqEdit e LinArqEdit indicam respectivamente a coluna e a linha que um determinado caracter ocupa no arquivo de problema. O valor destas variáveis, nesta mesma ordem, é mostrado na parte inferior esquerda da janela de edição.

A variável ITH indica a linha da tela onde será colocada a linha superior da moldura da janela de edição e a variável TTH indica o número de linhas da janela de edição. A linha inferior da moldura da janela de edição ficará posicionada na linha ITH + TTH + 1 da tela.

A variável ErroSintatico é utilizada para indicar se ocorreu algum erro sintático durante a fase de análise do modelo. Se ocorreu erro o sistema retorna à tela de edição do mesmo, coloca o cursor na posição onde aquele ocorreu e mostra uma mensagem explicando o seu motivo.

A variável SoVeTexto indica que o procedimento **Editar** será utilizado para permitir ao usuário navegar horizontalmente e verticalmente pelo texto com as setas, como descrito na Seção 3.2.5. Se a variável SoVeTexto for verdadeira não será permitida nenhuma modificação no texto mostrado na janela. A variável Tit contém o título que será colocado no topo da janela de edição.

Sempre que o usuário desejar sair da janela de edição ou visualização deverá teclar **F10**. Quando for permitida a saída, uma mensagem será mostrada na parte inferior da tela.

3.2.6 UNIT UANALISA

A unit UAnalisa é responsável pela analise sintática do modelo digitado pelo usuário. O modelo só poderá ser colocado na forma padrão, como descrito na Seção 2.2.1, após ser analisado e se estiver sintaticamente correto de acordo com regras que serão descritas abaixo.

Uma das grandes dificuldades encontradas no desenvolvimento do SIAMPLEX foi como permitir que o modelo digitado pelo usuário estivesse de acordo com o que descrevemos (Seção 2.2) como sendo a forma de um modelo de programação linear. Para tal tivemos que desenvolver uma pequena linguagem, com regras sintáticas simples, que nos permitisse esta verificação.

Apresentaremos a seguir a gramática da linguagem utilizada pelo SIAMPLEX, para a formulação de modelos válidos, através de diagramas sintáticos (Ver Aho[1972] e Wirth[1985]). Estes são utilizados a fim de estabelecer as diferentes construções sintáticas da linguagem.. Para a compreensão dos diagramas sintáticos basta acompanhar as seguintes regras:

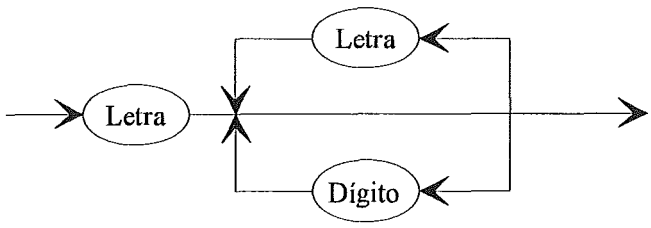
- Qualquer seqüência de símbolos obtida por um caminho que seja o sentido da seta constitui exemplo válido da "fifo" da cadeia.
- Os caracteres ou símbolos dentro de elipses ou círculos são elementos da linguagem.
- As palavras contidas em retângulos são nomes ou expressões definidas pelo usuário.

DEFINIÇÃO DA LINGUAGEM:

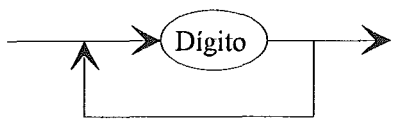
A, ... ,Z	Letras
0,1,2, ... ,9	Dígitos
+ –	Operadores tipo 1
/	Operador tipo 2
< = >	Operadores relacionais
{ }	Chaves de comentário
NL	Separador de nova linha
MAX MIN RES FIM	Palavras reservadas

VARIÁVEL

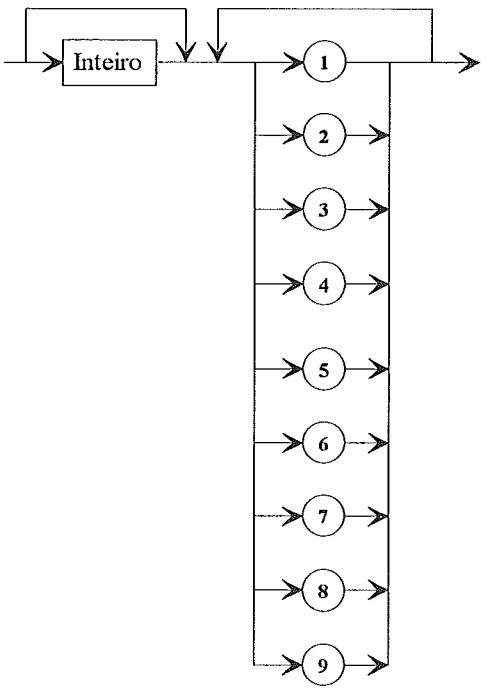
- (Uma variável válida possui no máximo 6 letras ou dígitos devendo ser iniciada obrigatoriamente por uma letra.)



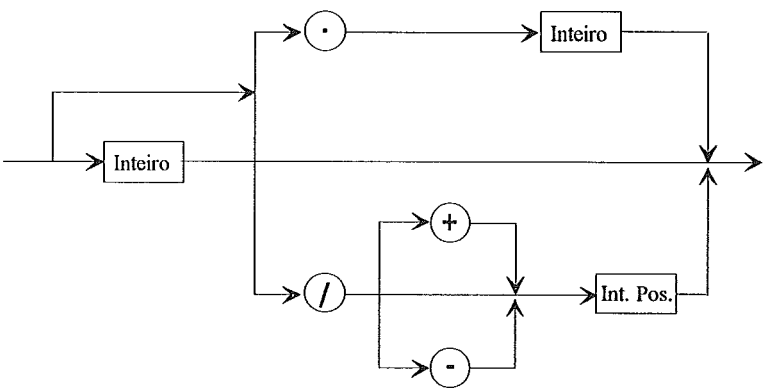
INTEIRO



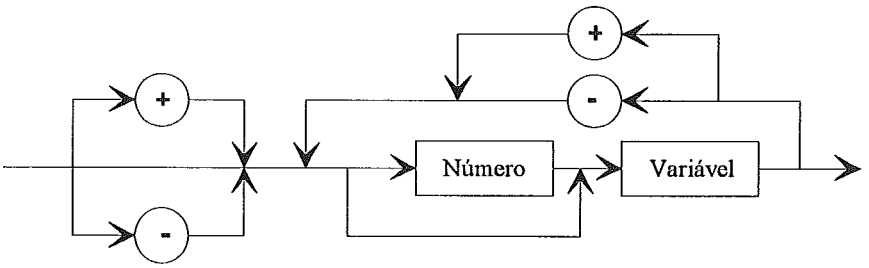
INTEIRO POSITIVO (Int. Pos.)



NÚMERO

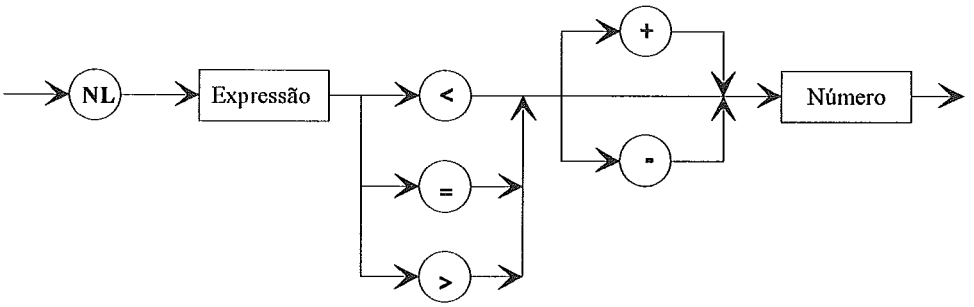


EXPRESSÃO

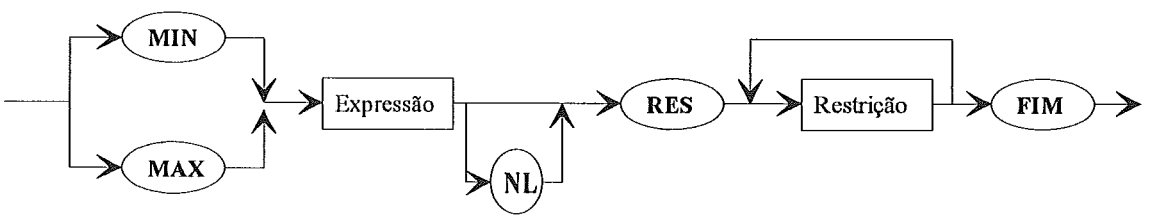


RESTRIÇÃO

♦ (Uma restrição não pode conter variável de nome repetido)



PROBLEMA



Observações:

- As letras maiúsculas e minúsculas são consideradas como iguais quando da formação do nome de uma variável. Assim a variável de nome "var01" é considerada pelo analisador como sendo igual a "VaR01", e se ambas aparecerem na mesma restrição ocorrerá erro de variável repetida.

- Os símbolos de comentário devem aparecer em pares "{ }" e o que estiver contido entre eles não será alvo de análise sintática. A principal utilidade dos comentários é permitir ao usuário fazer anotações a respeito do modelo e armazená-las junto ao mesmo.

  A inserção de comentários é permitida entre qualquer elemento de uma EXPRESSÃO, RESTRIÇÃO OU PROBLEMA. Um comentário pode se estender por várias linhas.

- Os elementos de uma mesma EXPRESSÃO ou RESTRIÇÃO podem estar colocados em diversas linhas. Com isto o usuário pode dispor o problema no formato que mais lhe agradar.

- Qualquer caracter que aparecer após a palavra reservada **FIM** será tratado pelo analisador sintático como se fosse um comentário.

3.2.7 UNIT UFORPADR

Após o modelo ser analisado e estar sintaticamente correto a unit UForPadr guiará o usuário passo a passo até que o modelo seja reduzido à forma padrão. O sistema seguirá como descrito na Seção 2.2.1.

Logo no início o sistema perguntará ao usuário sobre o tipo de cada variável do problema. Caso alguma variável seja negativa esta será automaticamente substituída por outra, positiva, que terá o mesmo nome daquela acrescido de uma aspa simples (').Os coeficientes da variável negativa serão substituídos por seus simétricos. Caso alguma variável seja livre está será substituída pela diferença de duas variáveis positivas que terão como nomes respectivamente, o nome da variável

livre acrescido de aspa simples (') e aspa dupla ("). Os coeficientes das novas variáveis serão o da variável livre e seu simétrico respectivamente.

Para uma melhor visualização ao adicionarmos uma variável de folga e/ou artificial em uma determinada restrição esta terá o nome FOLGA ou ARTIF seguido do número da restrição correspondente. Assim se acrescentarmos uma variável de folga e outra artificial na segunda restrição estas terão respectivamente os nomes, FOLGA02 e ARTIF02.

3.2.8 UNIT USIMPLEX

Na unit USIMPLEX é implementado o algoritmo SIMPLEX, conforme descrito na Seção 2.3.2, com uma pequena diferença que será apresentada posteriormente. Aqui também são implementadas, a inversão de matrizes usando decomposição LU (Seção 2.3.2) e reinversão (Seção 2.3.3). Esta unit é a responsável pelo monitoramento de todas as etapas do SIMPLEX.

Quando o usuário instrui o sistema para executar o seu modelo, que já deve ter sido previamente colocado na forma padrão, o controle é passado para a unit USIMPLEX.

Caso o modelo possua variáveis artificiais, o sistema inclui no quadro SIMPLEX inicial uma linha com a função artificial e seus coeficientes. Esta função será a sua FUNÇÃO OBJETIVO até que a Fase 1 seja concluída.

Durante a Fase 1, toda vez que o usuário retira uma variável artificial da base, o sistema exclui a coluna do quadro SIMPLEX correspondente a esta. Isto é feito sem nenhum prejuízo para a solução final do problema. Pois quando uma variável artificial é retirada da base o SIMPLEX não permite o seu retorno, isto poderia ocasionar o aumento do valor da função objetivo artificial, que deve ser minimizada.

A retirada da coluna correspondente à variável artificial que sai da base tem por objetivo diminuir a quantidade de informações, desnecessárias, a ser analisada pelo usuário. Dependendo do tamanho do modelo um quadro SIMPLEX pode não

caber nas 255 colunas do editor e deve ser dividido em subquadros para permitir a sua visualização.

A partir do momento que não existirem mais variáveis artificiais e a função artificial houver sido anulada, o sistema a exclui do quadro SIMPLEX e dá inicio a Fase 2.

Antes de selecionar qual variável deve entrar na base, o usuário deve verificar no quadro os itens abaixo:

- Se existe alguma coluna que não possua, pelo menos um coeficiente positivo. Isto indica que o problema possui solução ilimitada.

- Se todos os coeficientes da função objetivo sendo minimizada são não negativos (na Fase 1 será a função artificial). Isto indica que a solução encontrada é ótima.

- Se o valor da função artificial é zero e não existem variáveis artificiais básicas. Isto indica que a Fase 1 já foi concluída.

- Se todos os coeficientes da função artificial são não negativos e seu valor é diferente de zero. Isto indica que o problema não possui solução inicial básica viável.

Caso ocorra algum dos itens acima o usuário deve indicar que a fase correspondente está terminada.

O algoritmo implementado na unit USIMPLEX só difere do descrito na Seção 2.2.2 no que se refere a ESCOLHA DA NOVA VB. Ao invés de requerer que a variável a se tornar básica seja a mais negativa (negativa de maior valor absoluto), o sistema permite a escolha de qualquer variável, desde que esta seja negativa.

TOLERÂNCIA DE ERROS

Apesar do sistema recalcular todo o quadro após um determinado número de iterações (o usuário é quem controla este número), o uso da aritmética de ponto flutuante ocasiona erros, o que nos força a usar tolerâncias de erros ao invés de testarmos se determinados valores são exatamente iguais a zero. Na unit USIMPLEX empregamos os seguintes parâmetros para tolerância:

TOLCJ (10^{-5}) É o valor da tolerância usada para os coeficientes (de custo relativo) da função objetivo. Se $(c_j - z_j)$ for maior do que - TOLCJ para todas as variáveis não básicas a fase correspondente (Fase 1 ou 2) está terminada e a solução encontrada é considerada ótima.

TOLAIJ (10^{-8}) É o valor da tolerância usada para os coeficientes da matriz A. Se algum coeficiente a_{ij} tiver valor absoluto inferior a TOLAIJ é considerado como zero.

De acordo com Murtagh[1981] os números mostrados entre parênteses são valores típicos usados quando se trabalha com palavras de ponto flutuante com comprimento de 60 **bits**. A diferença de magnitude entre os mesmos reflete a importância relativa do erro permitido para cada um.

CAPÍTULO 4

OPERANDO O SIAMPLEX

Apresentamos neste capítulo como o usuário deve operar o SIAMPLEX. Aqui são mostradas todas as etapas desde a entrada no sistema até a resposta final, obtida após a execução do modelo. São também descritas com detalhes as telas do sistema bem como o significado dos seus campos.

Com certeza, após a leitura deste capítulo, o usuário que não conseguiu operar corretamente o sistema utilizando as telas de auxílio do mesmo, estará apto a fazê-lo.

Além de servir de um tutorial do sistema apresentaremos um exemplo onde fica demonstrada a importância da reinversão para a resolução de modelos de maior porte.

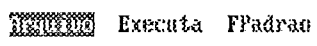
4.1 DESCRIÇÃO DO SIAMPLEX

Para iniciar a execução do SIAMPLEX, o usuário deve na linha de comando do "DOS", digitar "SIAMPLEX". No início da execução do sistema a primeira tela que aparece é mostrada na Figura 01. Podemos observar na parte superior desta figura, no painel principal de opções, que a palavra "Arquivo" aparece em vídeo

reverso, isto indica que o item "Arquivo" será selecionado caso o usuário tecle **ENTER**. Se este deseja outro item, deve mover a barra de seleção para a direita com a tecla "→", para a esquerda com "←", ou através das letras de atalho: **A** para "Arquivo", **E** para "Executa" ou **F** para "FPadrão".

Na parte inferior da tela, "**F1**" em destaque ao lado da palavra "Ajuda", indica que se o usuário pressionar esta tecla será mostrada uma tela de ajuda contendo explicações sobre o funcionamento do sistema.

O usuário pode cancelar qualquer seleção do painel principal através da tecla "**ESC**". Se este houver selecionado o item "Arquivo", mostrado na Figura 02, ao teclar "**ESC**", a tela retorna imediatamente ao estado inicial, Figura 01.



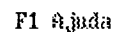


Figura 01

A Figura 02 apresenta uma nova tela do sistema que é ativada ao ser selecionado item "Arquivo" do painel principal de opções. Esta tela mostra um novo painel de opções com quatro itens. A seleção destes pode ser efetuada de duas maneiras: movendo a barra de seleção para baixo com a tecla "↓", para cima com "↑" e teclando "**ENTER**", ou através das letras de atalho **E** (Edita), **A** (Abre), **S**, (Salva) e **F** (Fim).

Com o objetivo de evitar repetições, a menos que seja mencionado o contrário, só serão aceitos como comandos em um painel de opções:

- As teclas de atalho.
- As teclas "→" e "←" para um painel disposto horizontalmente.
- As teclas "↓" e "↑" para um painel disposto verticalmente.
- Teclas de função mostradas na parte inferior da tela.
- Tecla "ENTER".

Ao ser selecionado o item "Edita" na Figura 02, o sistema invoca o editor de texto (apresentado na Seção 3.2.5).

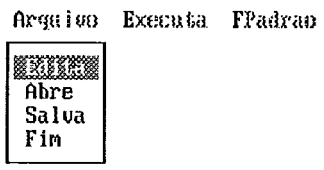


Figura 02

Podemos notar na Figura 03, uma moldura formada por traços duplos. Na parte superior desta, aparece o nome do arquivo sendo editado. No nosso caso, como não foi aberto nenhum arquivo, através do item "Abre", o sistema atribui temporariamente o nome "SEMNOME.PLX'. Na parte inferior do lado esquerdo da moldura, aparecem dois números que indicam respectivamente a linha e a coluna onde se encontra o cursor. Na última linha da tela, a tecla de função **F10** indica que ao ser precionada o sistema abandona o modo de edição e retorna ao painel principal.

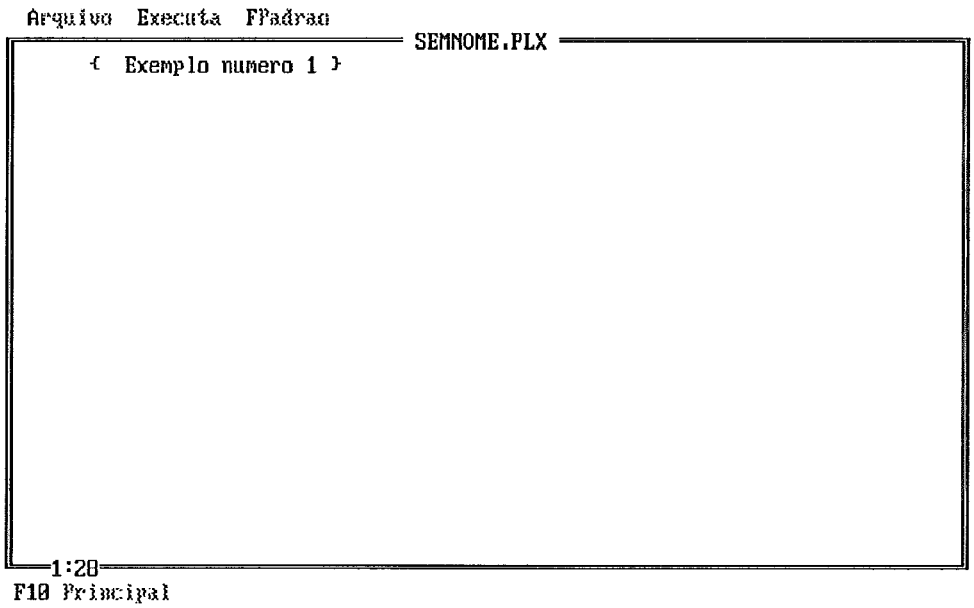


Figura 03

Quando o usuário ativa o item "Abre" do painel mostrado na Figura 02, o controle do sistema é passado para a unit UARQUIVO, esta seleciona todos os arquivos do diretório corrente contendo a terminação **.PLX**, e os coloca na janela que aparece no canto superior esquerdo da Figura 04. Aqui o usuário através das teclas "↓" e "↑", move a barra de seleção para o arquivo desejado.

A janela "Diretorio", contendo o nome dos arquivos tem tamanho variável de 1 a 15 linhas. Se o diretório corrente contém mais do que 15 arquivos com terminação **.PLX** o usuário deve mover a barra de seleção para baixo, quando esta chegar na linha inferior da janela os nomes começarão a rolar para cima e aqueles que não puderam ser mostrados anteriormente aparecerão. O contrário acontece quando a barra está no topo, neste caso se o usuário houver rolado os nomes para cima, estes voltarão à posição inicial bastando teclar "↑".

Se o usuário desejar editar um arquivo com terminação diferente, na última linha da janela "Diretorio" existe uma linha com os caracteres "..\", ao selecioná-la o sistema apresenta uma nova janela onde solicita o nome do arquivo.

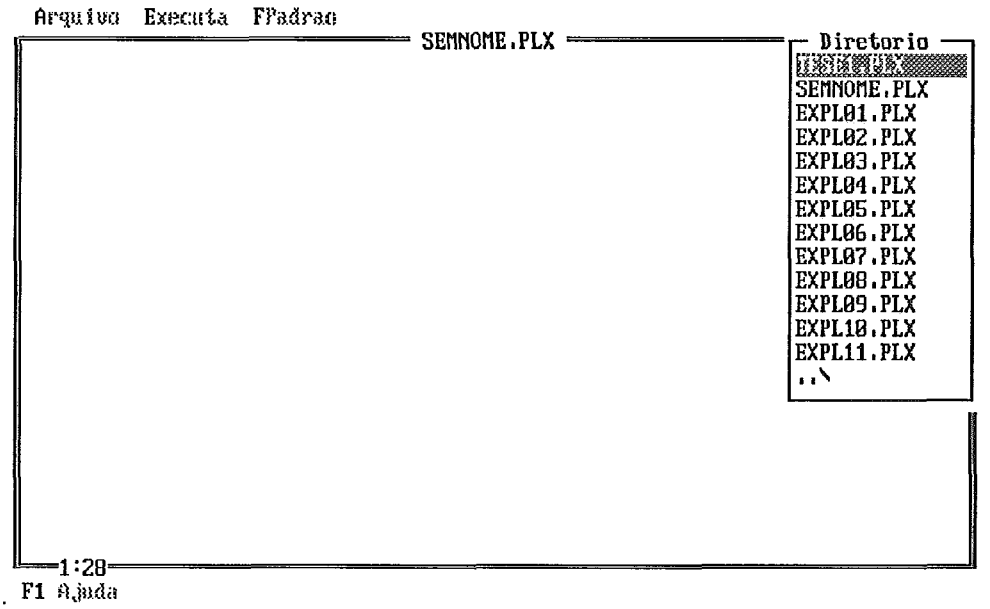


Figura 04

Ao selecionar o nome do arquivo desejado, a unit UARQUIVO inclui o seu conteúdo no editor de texto e o torna disponível para edição. Veja na Figura 05 o conteúdo do arquivo "TESE1.PLX" selecionado na Figura 04. Note no topo da moldura o nome do arquivo.

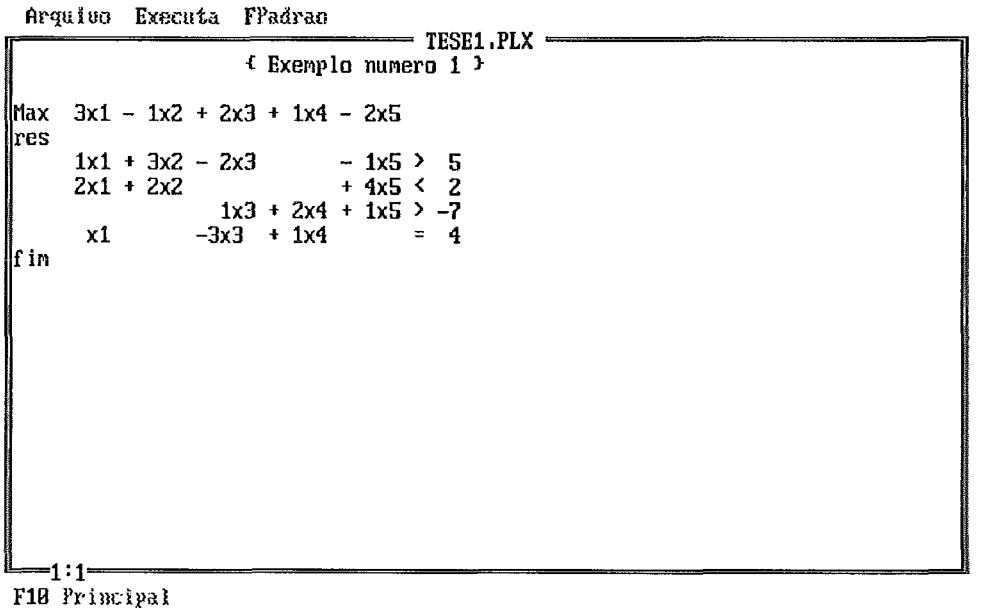


Figura 05

Na Figura 06 temos uma janela no centro da tela que solicita o nome com o qual o arquivo deve ser salvo em disco. Esta janela é ativada ao ser selecionada a opção "Salva" da Figura 02.

Quando a janela "Salvar Arquivo" é mostrada o nome do arquivo corrente em edição é automaticamente colocado após "NOME: ", caso o usuário deseje salvá-lo com outro nome basta apagá-lo com "BACKSPACE" e digitar o novo nome, como feito na Figura 06.

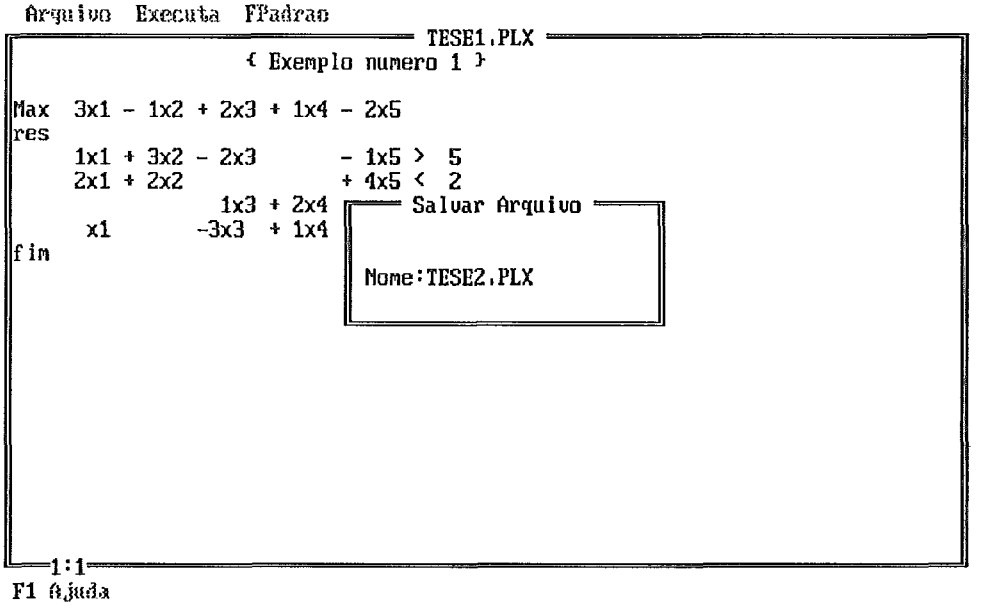


Figura 06

Ao selecionar o item "Fim" da Figura 02, o usuário encerra a execução do sistema e retorna a linha de "PROMPT" do DOS.

A fim de que o usuário possa colocar um determinado modelo na forma padrão, este deve estar no arquivo atual da janela de edição.

A colocação do modelo na forma padrão é feita através do item "FPadrao" do painel principal, como mostrado na Figura 07.

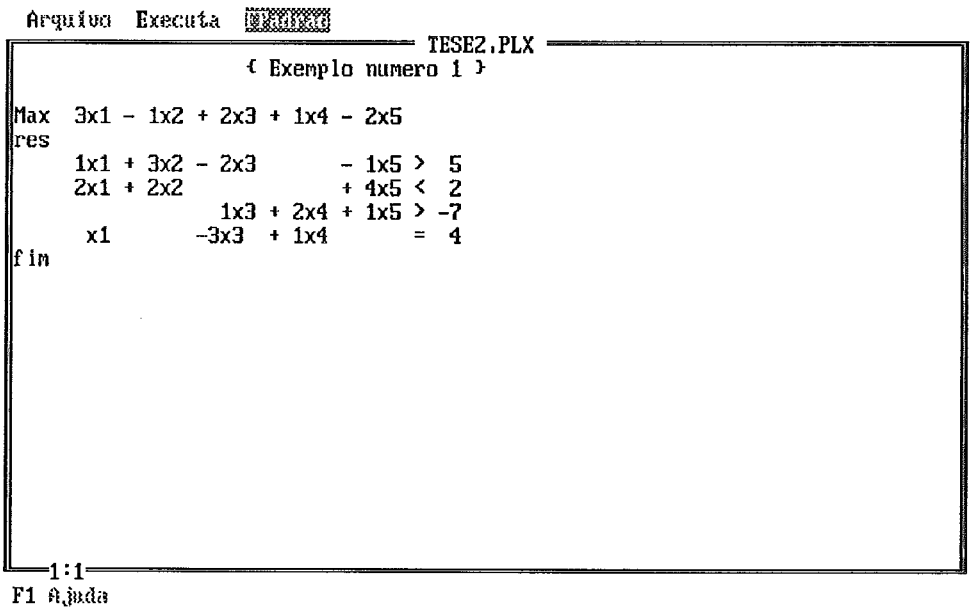


Figura 07

O primeiro passo para a colocação do modelo na forma padrão é a análise sintática do mesmo feita pela unit UANALISA (apresentada na Seção 3.2.6). Caso seja encontrado algum erro na sintaxe do modelo, uma mensagem de advertência é colocada no canto superior esquerdo da moldura, como mostra a Figura 08 (nesta figura foi retirada, propositalmente, a chave '{' , que indica início de comentário).

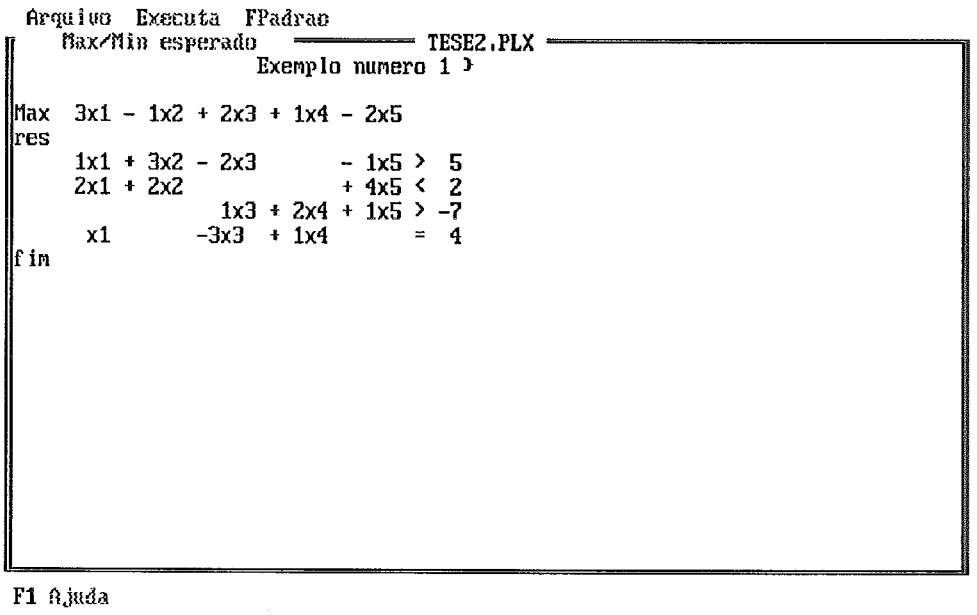


Figura 08

Quando o modelo estiver sintaticamente correto, uma janela é mostrada no centro da tela, como mostra a Figura 09, com uma mensagem que solicita ao usuário, se este deseja ou não colocar o modelo na forma padrão.

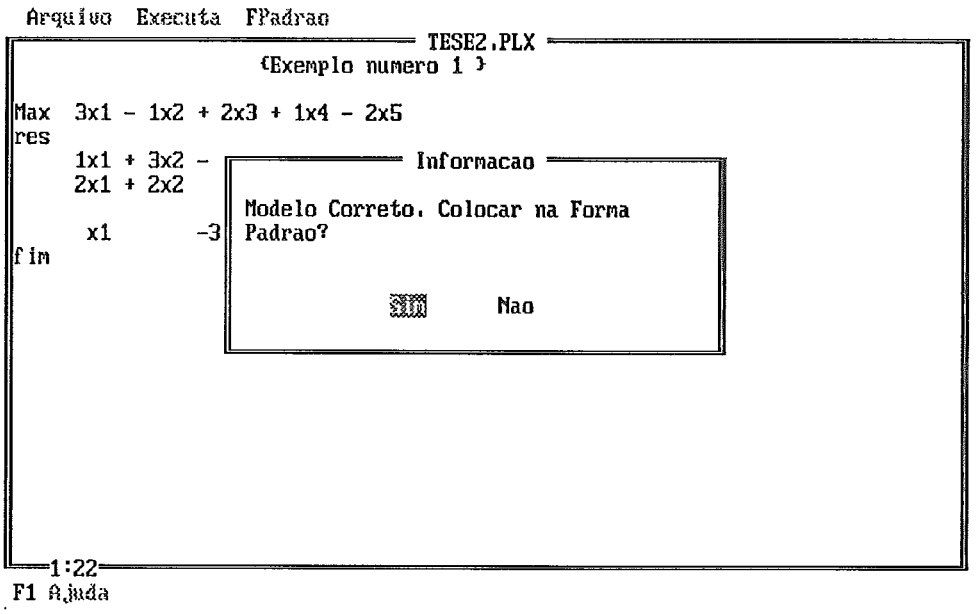


Figura 09

Ao responder afirmativamente na janela da Figura 09, uma nova janela, Figura 10, solicita informações sobre o tipo das variáveis do modelo.

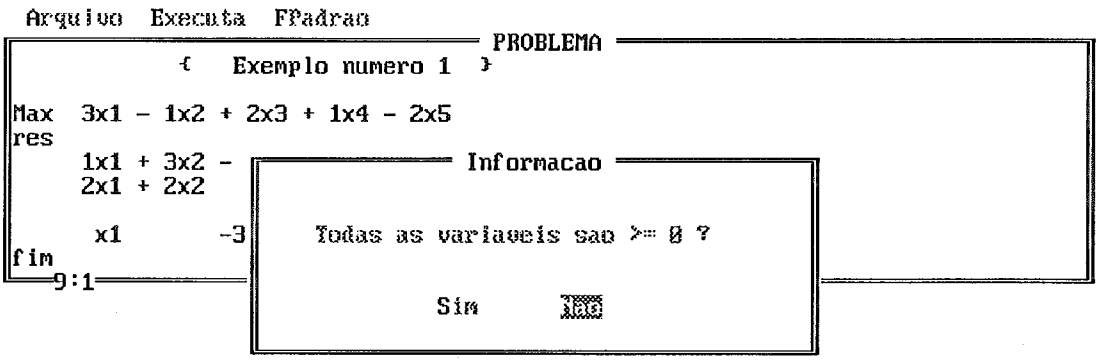


Figura 10

Se houver alguma variável não positiva (menor ou igual a zero) no modelo, o usuário deve indicar na janela a opção "Nao". Neste caso aparecerá uma nova tela, Figura 11, onde este deve informar o tipo de cada variável.

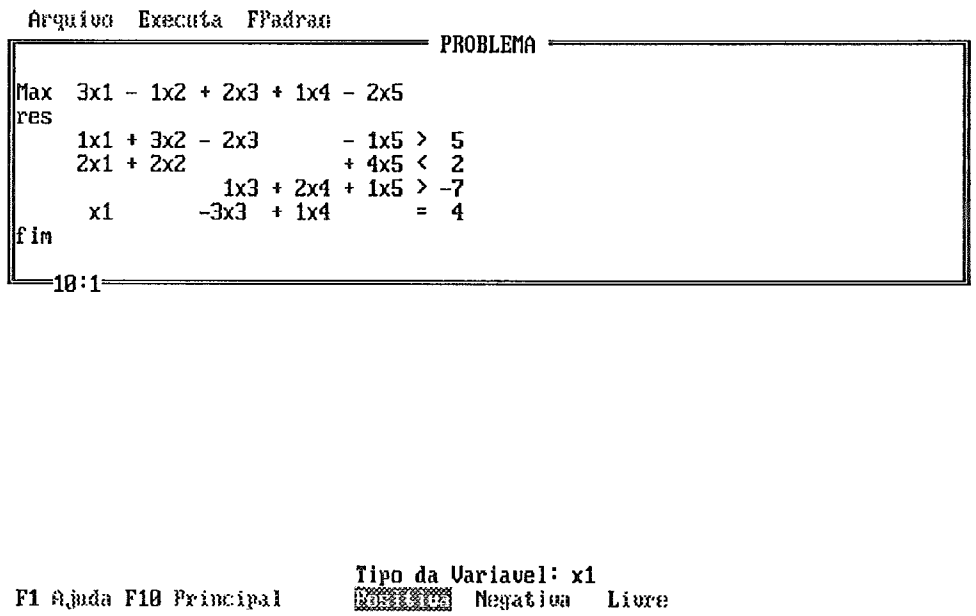


Figura 11

Após informar o tipo de todas as variáveis, o usuário inicia propriamente a colocação do modelo na forma padrão de acordo com a Seção 2.2.

A Figura 12, mostra como o usuário deve colocar a função objetivo na forma padrão. Se esta for de minimização ("MIN"), deverá selecionar o item "Funcao Correta", caso contrário "Multiplica(-1)". O item "Ver problema" permite ao usuário visualizar no editor o modelo. Isto é útil quando este, por seu tamanho, não pode ser mostrado por completo na janela da Figura 12.

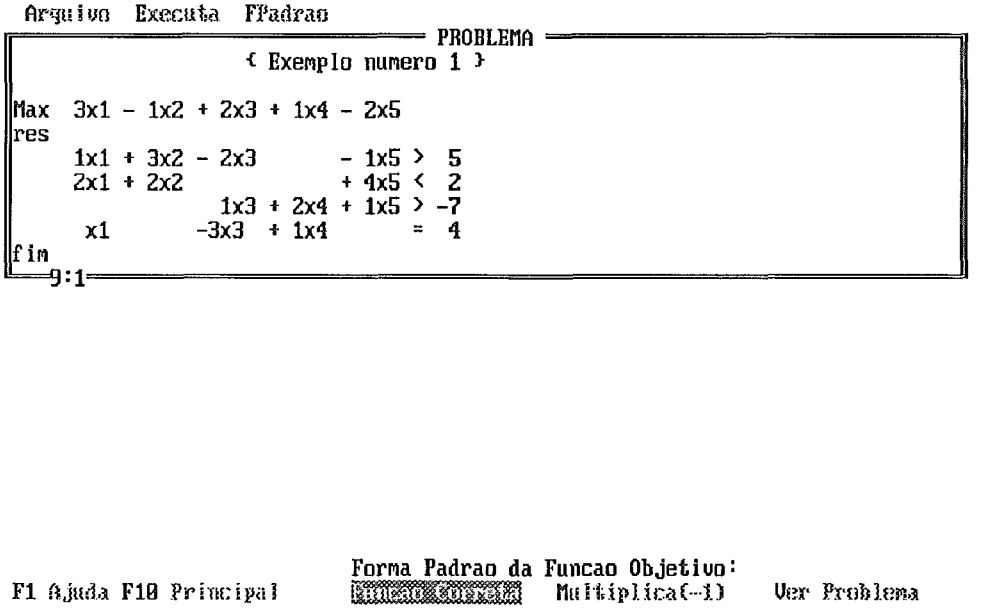


Figura 12

Após a colocação da função objetivo na forma padrão o usuário passa à colocação dos RHS's das restrições do modelo nesta forma. Caso ocorra algum erro aparecerá uma mensagem explicando o motivo.

Na Figura 13, selecionamos propositalmente o item "RHS Correto" para a terceira restrição, isto ocasionou o erro pois por apresentar RHS negativo todos os coeficientes da restrição e o RHS deveriam ser multiplicados por -1 (item "Multiplica(-1)).

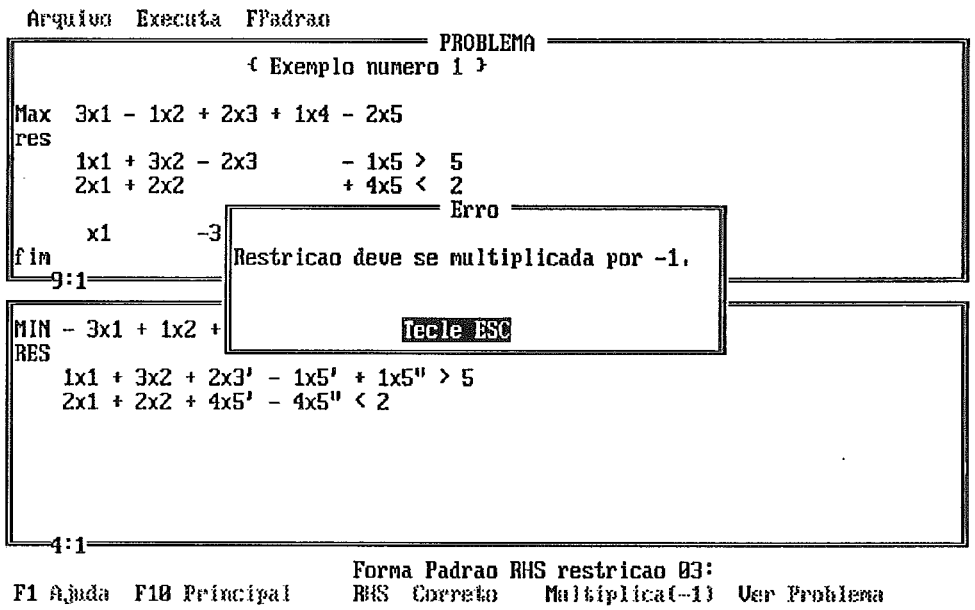


Figura 13

O próximo passo será a adição das variáveis de folga, se necessárias, em cada restrição como mostra a Figura 14.

Se a restrição necessita de uma variável de folga positiva, o usuário deve selecionar o item "+Folga**", se negativa "-Folga**" (os asteriscos indicam o número da restrição), caso não necessite de variável de folga "Sem Folga".

O item "Janelas" ao ser selecionado coloca uma janela no canto superior direito da moldura, como mostra a Figura 15. Esta permite ao usuário visualizar na parte superior da tela o modelo na forma original ou este com as alterações efetuadas até o momento.

A janela da Figura 15 permite duas opções:

"Problema" → permite visualizar na parte superior da tela o problema em sua forma original.

"FuncRHS" → permite visualizar na parte superior da tela as alterações feitas nas variáveis, na função objetivo e no RHS do problema original.

Após a conclusão de cada etapa para a colocação do problema na forma padrão, será acrescentada uma nova opção nesta janela, o que permite o acompanhamento das alterações efetuadas..

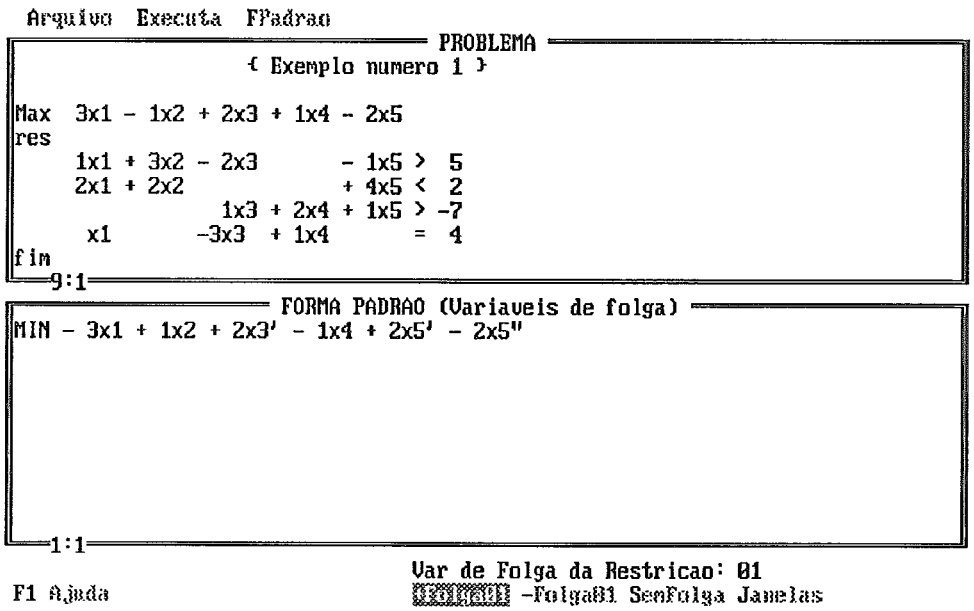


Figura 14

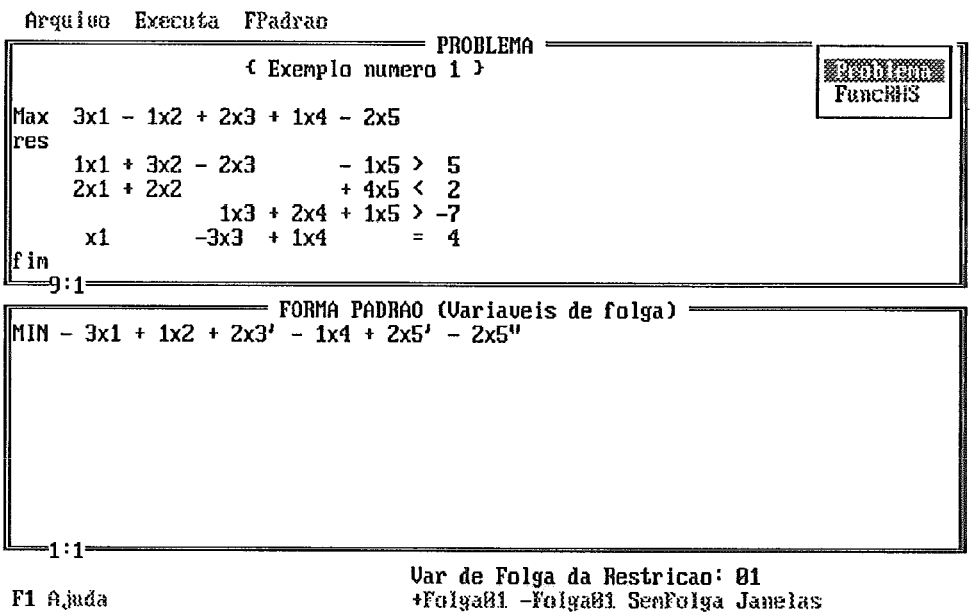


Figura 15

Na janela superior da Figura 16, vemos o modelo com as variáveis, a Função objetivo e os RHS's das restrições na forma padrão, isto indica que o usuário selecionou o item "FuncRHS" da Figura 15. Nesta janela o usuário pode mover o

texto, isto é útil quando o modelo se estende além da linha 9 ou da coluna 78 e a parte que interessa não está visível.

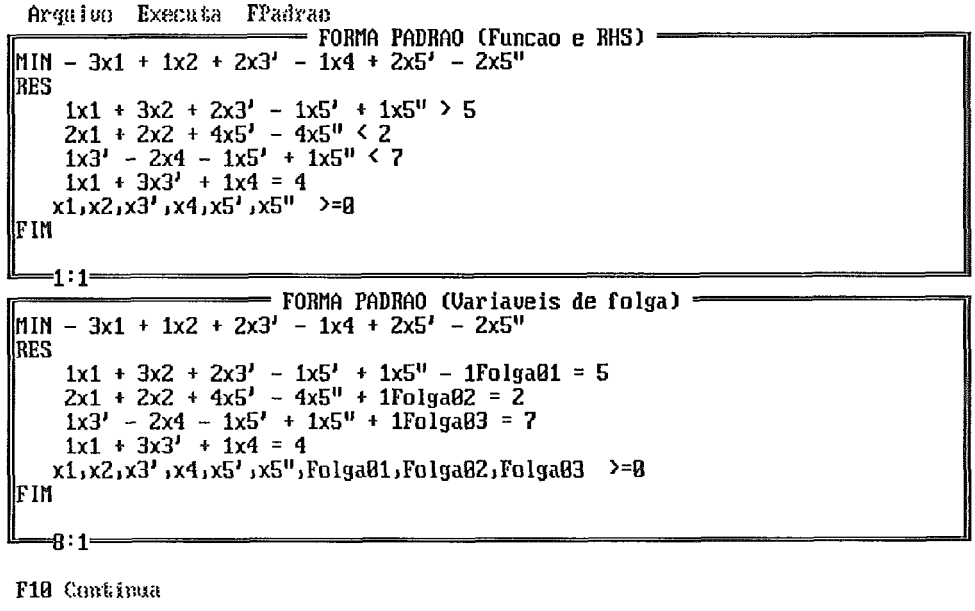


Figura 16

A Figura 17 mostra como o usuário deve fazer as inclusões das variáveis artificiais, se necessárias, em cada restrição.

O item "Janelas" ao ser selecionado abrirá uma janela no canto superior direito, como mostra a Figura 15, porém acrescenta o item "VarFolg" aos dois existentes, este permite visualizar o modelo já com as variáveis de folga.

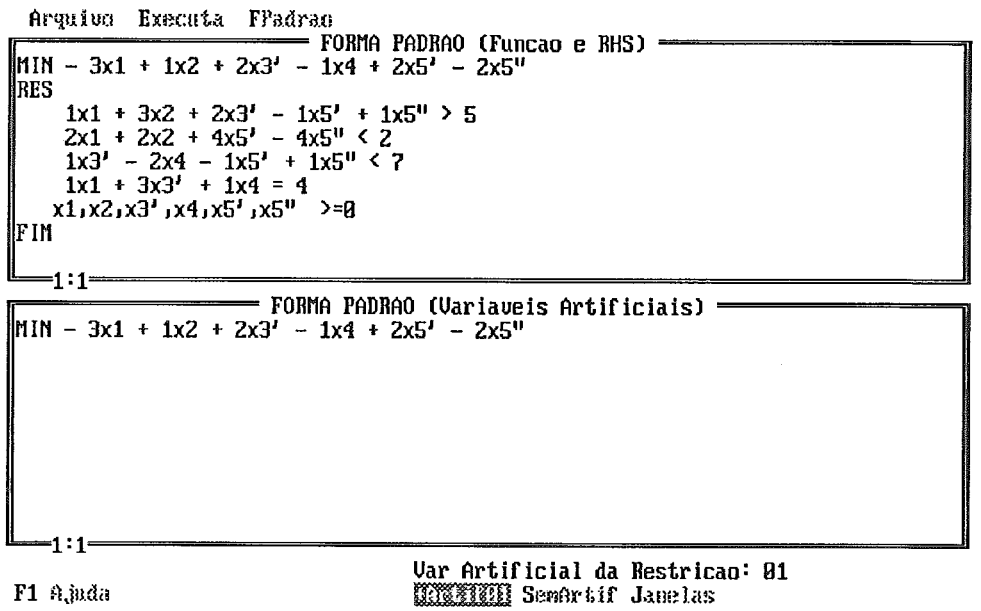


Figura 17

Na janela inferior da Figura 18, vemos o modelo na forma padrão final. A partir deste momento ele já pode ser executado.

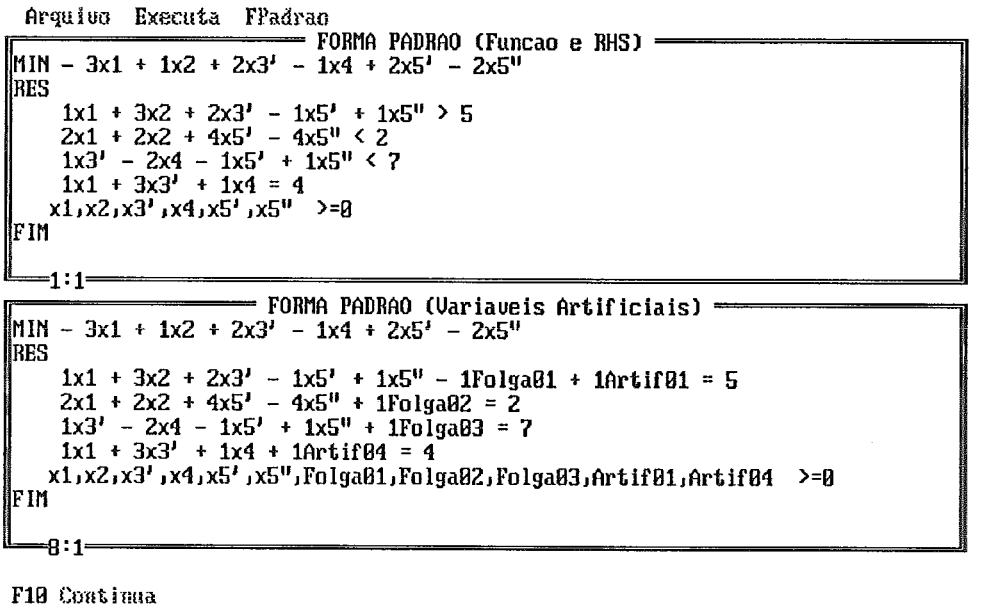


Figura 18

Quando o usuário finaliza a colocação do modelo na forma padrão o sistema retorna ao painel principal e mostra o problema original. Somente neste momento o usuário pode selecionar o item "Executa", como mostra a Figura 19.

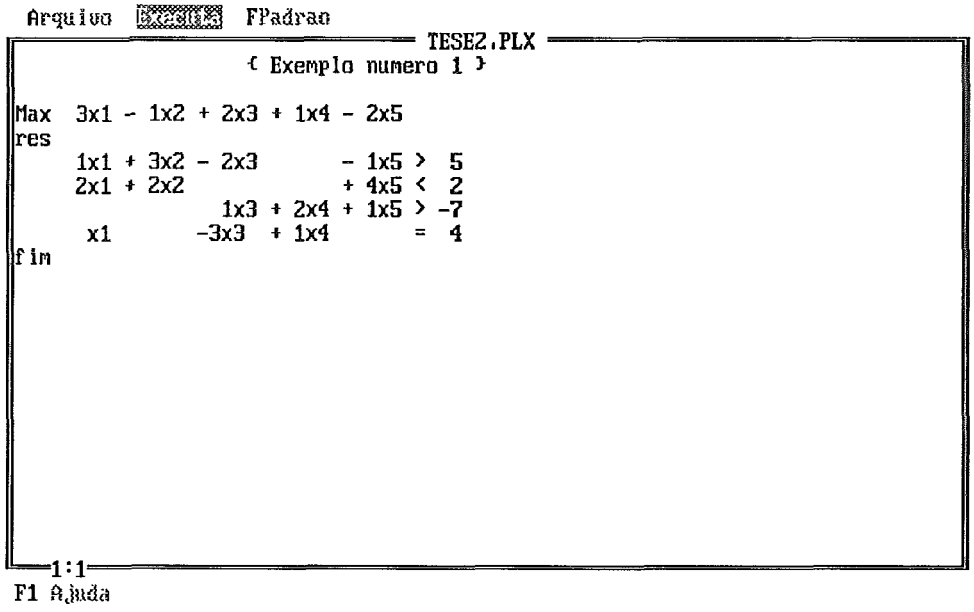


Figura 19

Como primeiro passo para a execução do modelo, o sistema solicita ao usuário o número de iterações que o SIMPLEX deve realizar, Figura 20, antes da reinversão da matriz **B** e de recalcular todo o quadro (Seção 2.3).

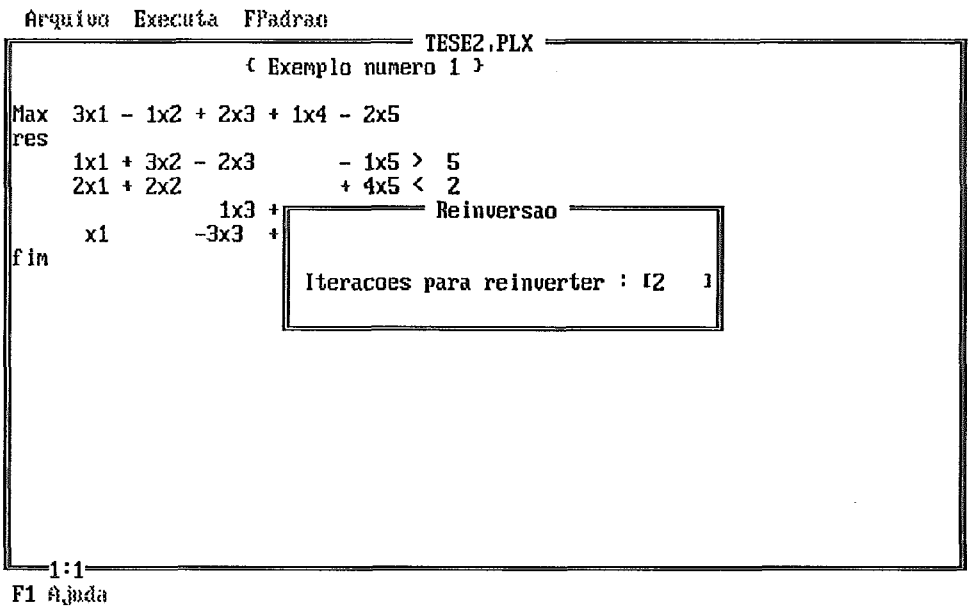


Figura 20

Prosseguindo o sistema solicita ao usuário, se este deseja a impressão dos quadros SIMPLEX, como mostra a Figura 21. Em caso afirmativo, estes não serão apresentados nas 255 colunas do editor, mas em 78 colunas, a fim de permitir a impressão completa dos mesmos (caso fossem dispostos nas 255 colunas teríamos muitas dificuldades para formatá-los novamente a fim de imprimí-los).

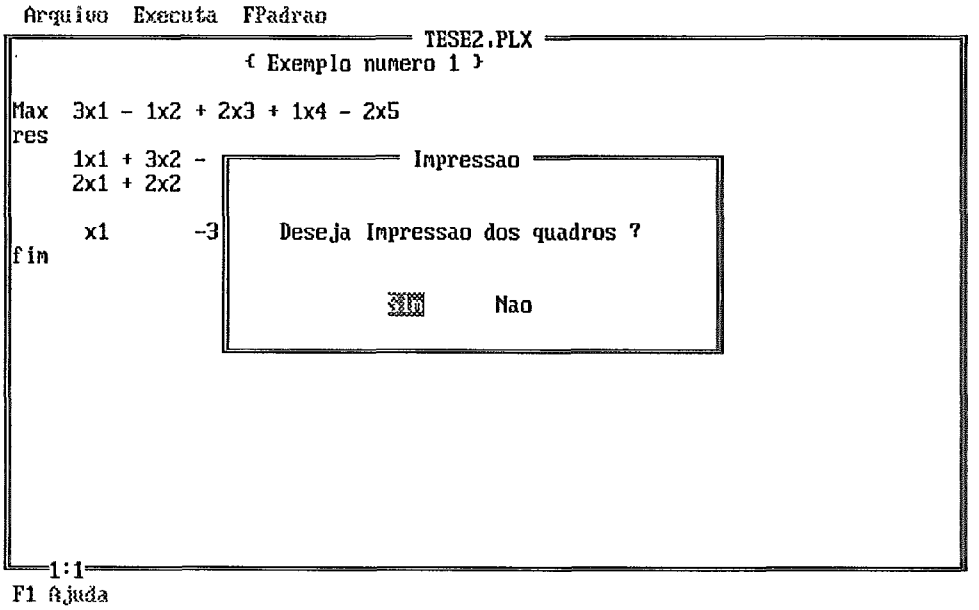


Figura 21

A Figura 22 mostra o início da execução do SIMPLEX. Esta apresenta uma janela no centro da tela que pergunta ou usuário se o SIMPLEX deve iniciar pela fase 1 (se o modelo contém variáveis artificiais) ou não (caso contrário).

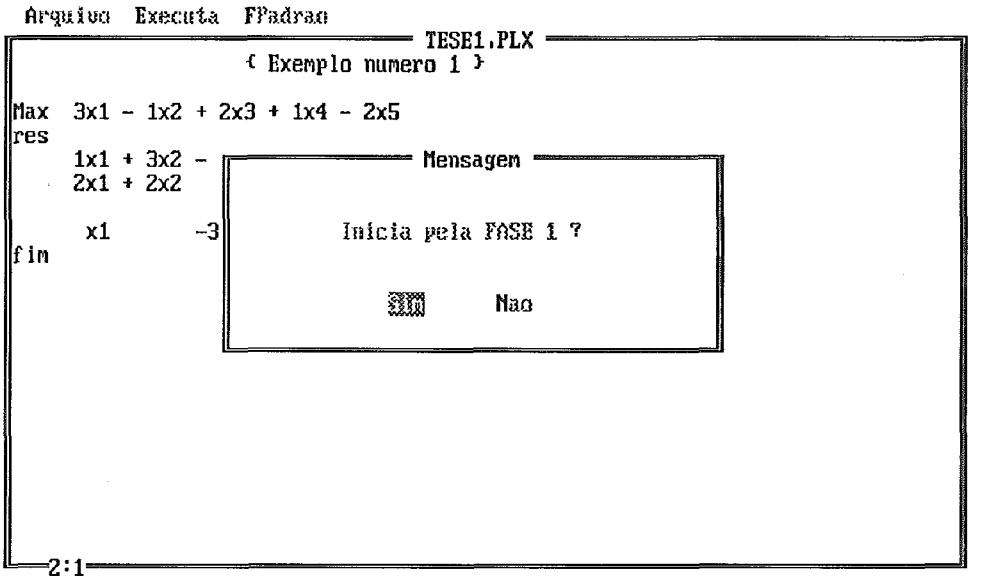


Figura 22

Na Figura 23 vemos o quadro inicial do SIMPLEX. Podemos notar por esta figura que quando o tamanho do quadro SIMPLEX for maior do que a janela de edição (78 ou 255 colunas), estes serão particionados.

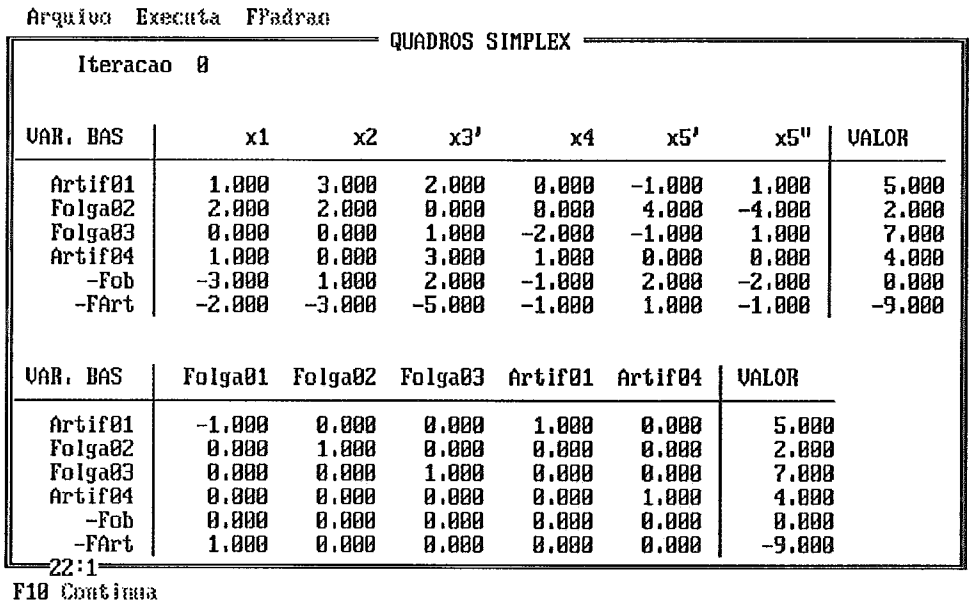


Figura 23

A cada iteração o sistema solicita ao usuário se a fase corrente (Fase 1 ou 2) está concluída, Figura 24. Para que este possa voltar ao quadro e examiná-lo, antes de tomar a decisão, basta teclar "ESC".

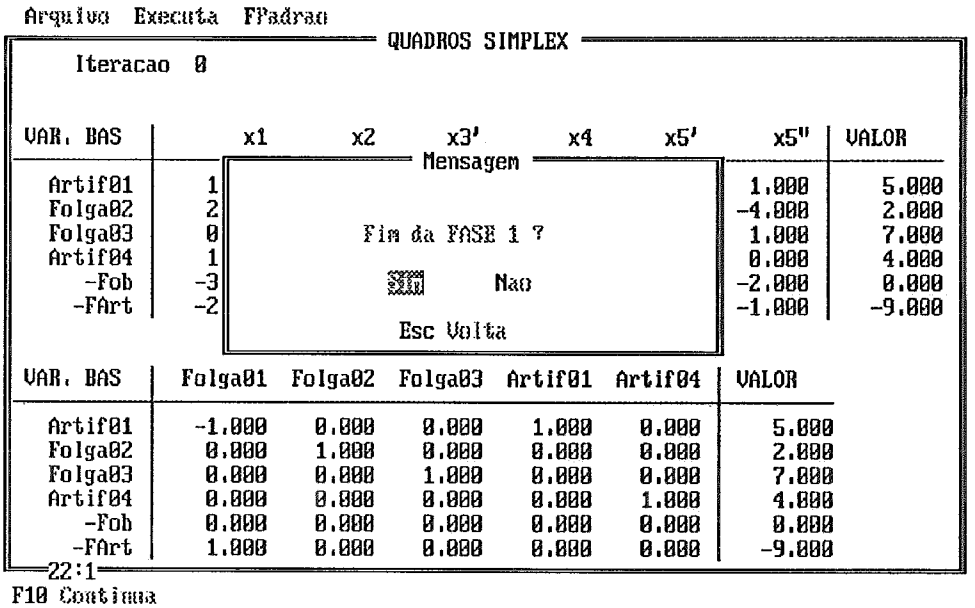


Figura 24

Escolhemos propositalmente o item "Sim", na Figura 24, a fim de mostrar o tipo de mensagem mostrada caso o usuário cometa algum erro na seleção, ver Figura 25.

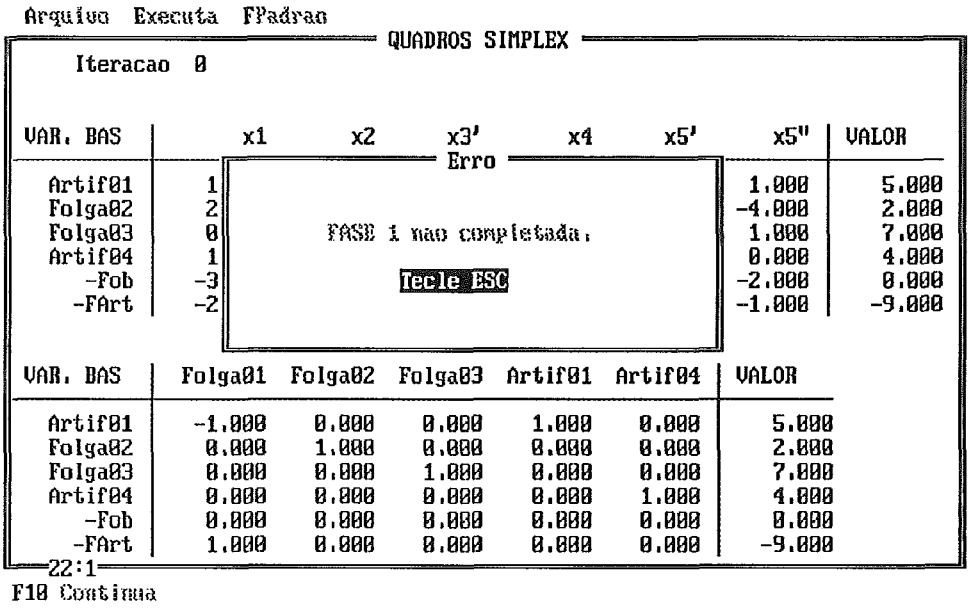


Figura 25

Neste ponto, Figura 26, o usuário de acordo com as regras do SIMPLEX (Seção 2.2), deve escolher a variável a se tornar básica.

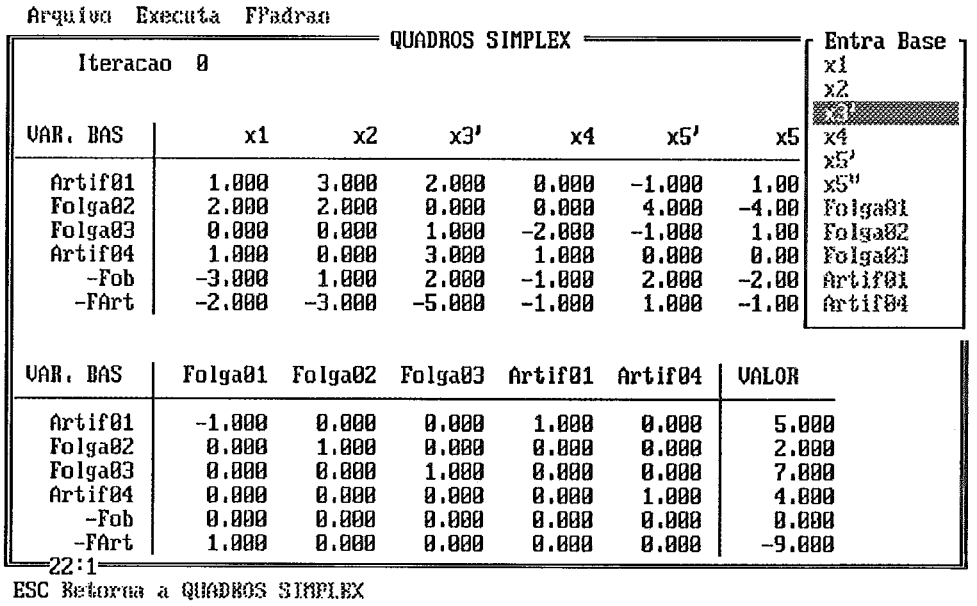


Figura 26

Após escolher a variável que deve entrar na base o sistema apresenta uma tela, Figura 27, onde são mostrados os resultados da divisão do RHS de cada restrição pelo A[I,J] correspondente. Aqui o usuário seleciona a variável que deve sair da base.

Note que no resultado da segunda linha temos "INFINITO", isto se deve ao fato de que o valor correspondente a A[I,J] é zero, ver Figura 26. Neste ponto o usuário pode sair da janela "Sai da Base" para visualizar os resultados, antes de efetuar sua escolha, através da tecla "**ESC**".

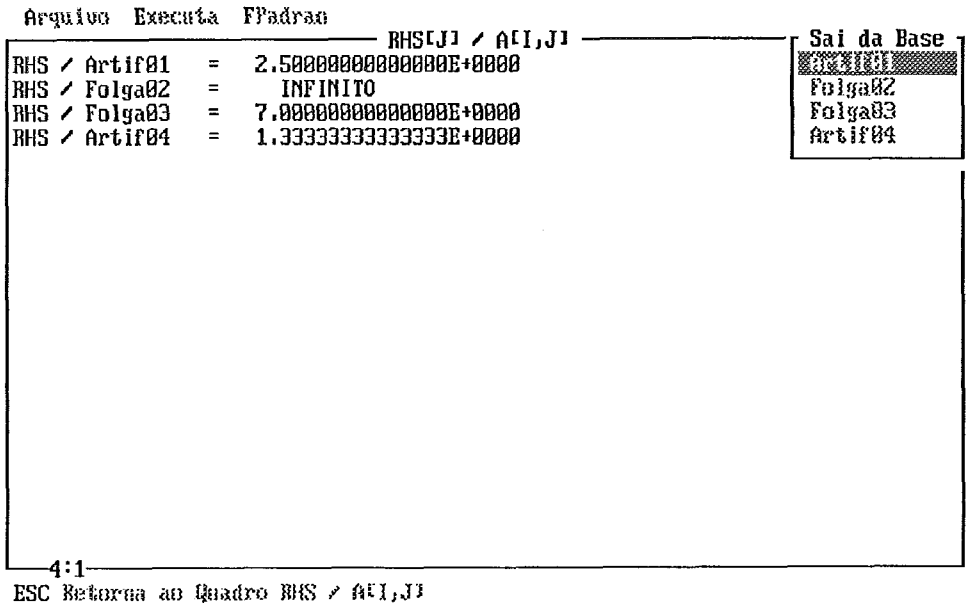


Figura 27

Pelas regras do SIMPLEX a variável que deve deixar a base, Figura 26, é **ARTIF04**, porém escolhemos **ARTIF01**, isto ocasiona o erro mostrado na Figura 28.

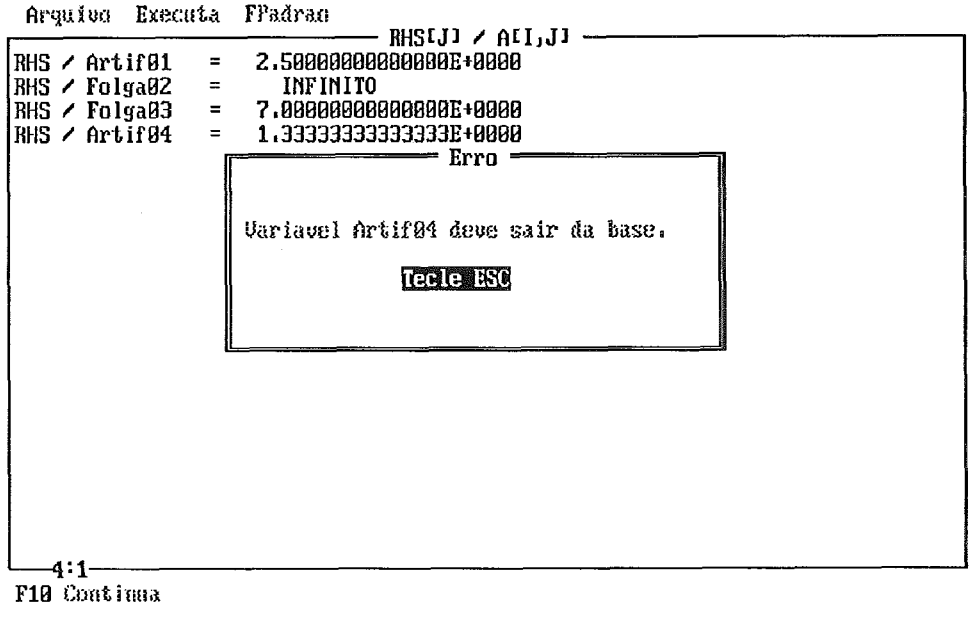


Figura 28

A Figura 29 mostra o quadro SIMPLEX após a primeira iteração.

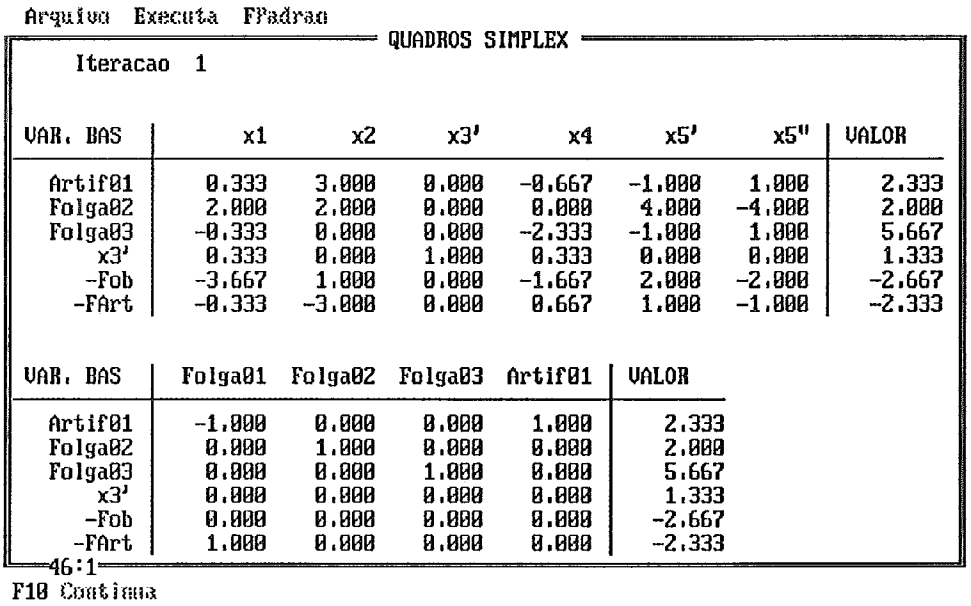


Figura 29

Escolhemos **x3'**, como a próxima variável a entrar na base na iteração 2, Figura 30, com o objetivo de mostrar uma escolha inválida e sua correspondente mensagem de erro, Figura 31.

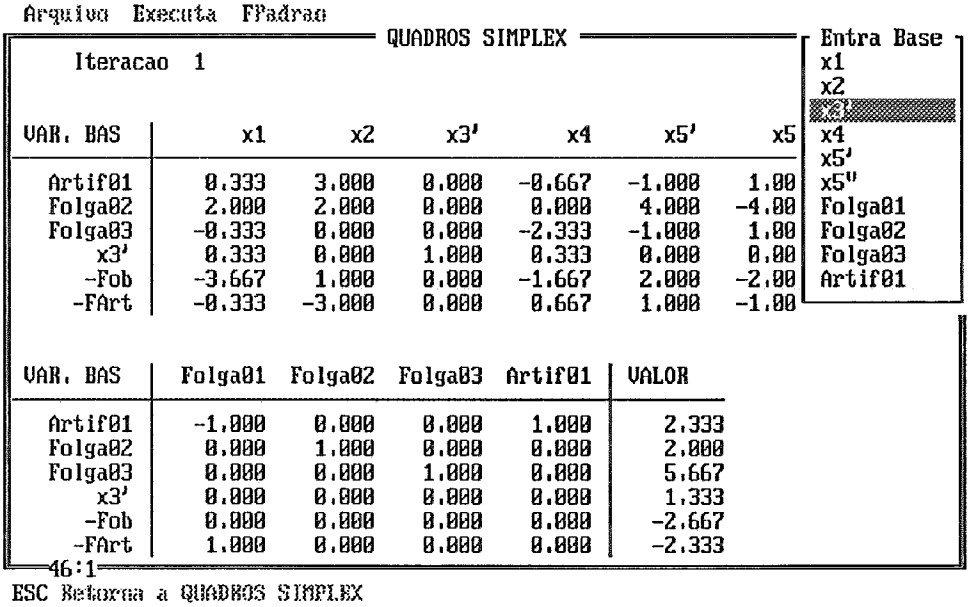


Figura 30

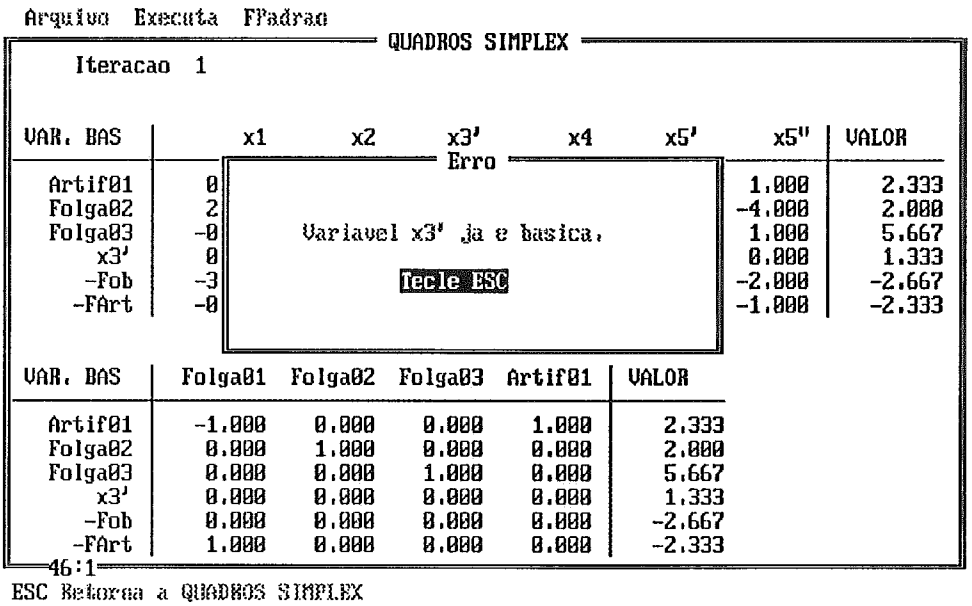


Figura 31

Vemos pela Figura 31, que o sistema não permite a escolha de **x3'**, e não continuará até que uma variável escolhida possa entrar na base.

De acordo com as regras do SIMPLEX a variável escolhida para entrar na base deve ser a "mais negativa", a negativa de maior valor absoluto. Esta regra de maior negatividade é apenas uma heurística que tem por base o maior decréscimo da função objetivo. Porém em alguns casos, especificamente no nosso exemplo, a escolha por esta regra pode levar a um maior número de iterações até a solução final. Para dar maior flexibilidade ao usuário qualquer variável negativa pode entrar na base.

Na Figura 32, escolhemos **x5"** para entrar na base.

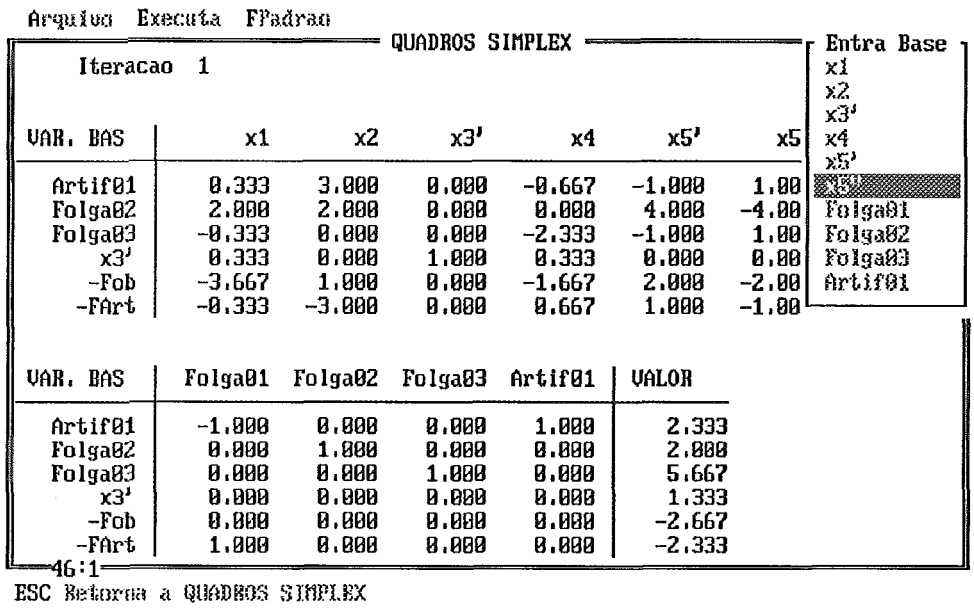


Figura 32

Devemos agora, Figura 33, escolher a variável a deixar a base. Observe também nesta figura um valor negativo na segunda linha, e "INFINITO" na quarta.

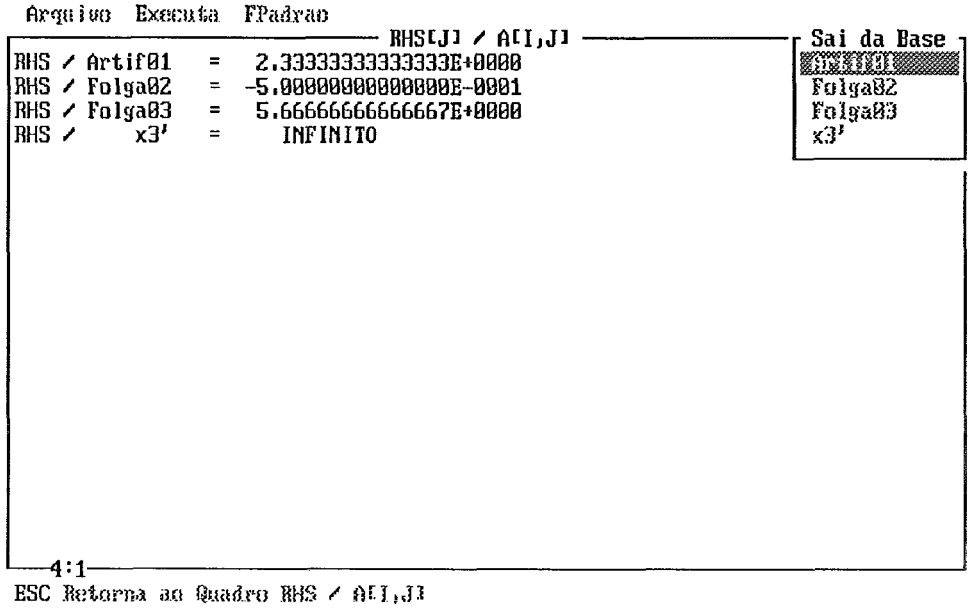


Figura 33

A Figura 34, mostra o quadro após a segunda iteração. Neste quadro não existem mais variáveis artificiais básicas e a função artificial foi zerada. Assim o

usuário deve responder afirmativamente à pergunta (ver Figura 24) que é feita antes da indicação de qual variável deve tornar-se básica.

Propositalmente respondemos negativamente, o que ocasiona o erro mostrado na Figura 35.

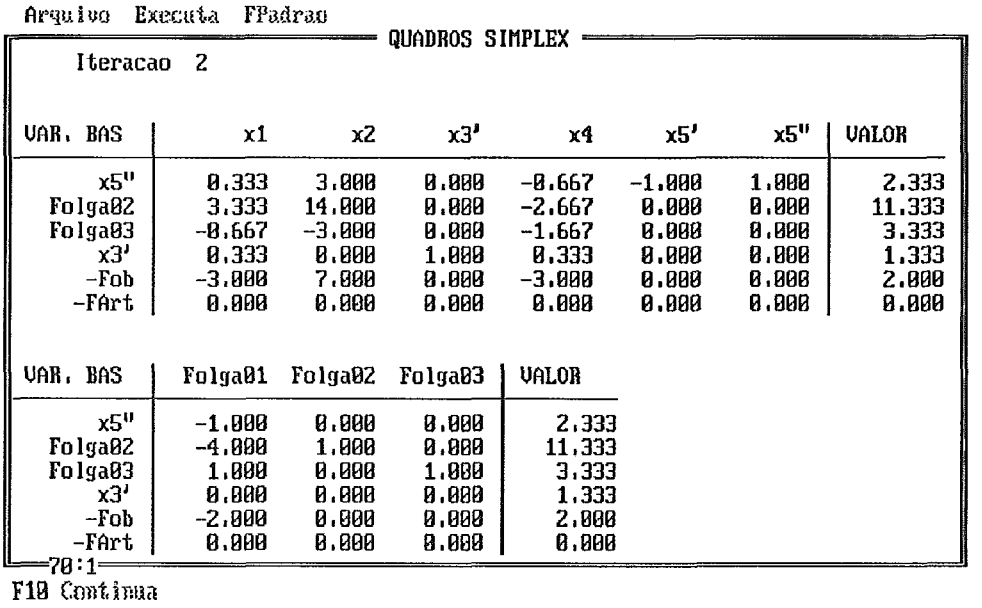


Figura 34

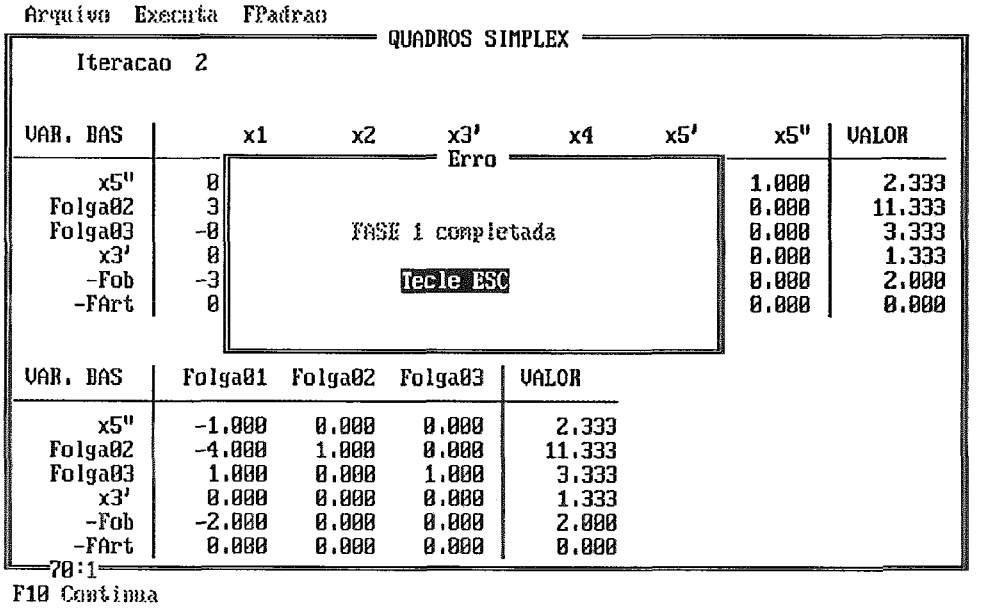
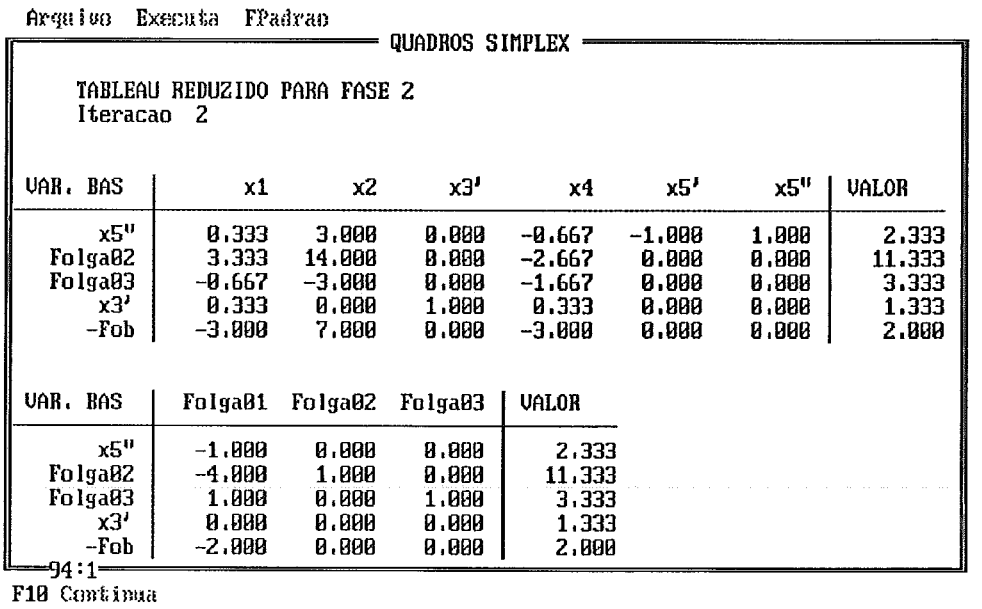


Figura 35

Após a eliminação das variáveis artificiais e da função artificial, teremos o início da Fase 2. A Figura 36 mostra o quadro SIMPLEX reduzido para a Fase 2.



VAR. BAS	x1	x2	x3'	x4	x5'	x5"	VALOR
x5"	0.333	3.000	0.000	-0.667	-1.000	1.000	2.333
Folga02	3.333	14.000	0.000	-2.667	0.000	0.000	11.333
Folga03	-0.667	-3.000	0.000	-1.667	0.000	0.000	3.333
x3'	0.333	0.000	1.000	0.333	0.000	0.000	1.333
-Fob	-3.000	7.000	0.000	-3.000	0.000	0.000	2.000

VAR. BAS	Folga01	Folga02	Folga03	VALOR
x5"	-1.000	0.000	0.000	2.333
Folga02	-4.000	1.000	0.000	11.333
Folga03	1.000	0.000	1.000	3.333
x3'	0.000	0.000	0.000	1.333
-Fob	-2.000	0.000	0.000	2.000

Figura 36

Como mostrado na Figura 24, antes de solicitar o nome da variável a se tornar básica o sistema pergunta se já foi terminada a Fase 1 ou 2. A Figura 37 mostra esta mensagem para a Fase 2, pois não mais existem variáveis ou função artificial no quadro SIMPLEX .

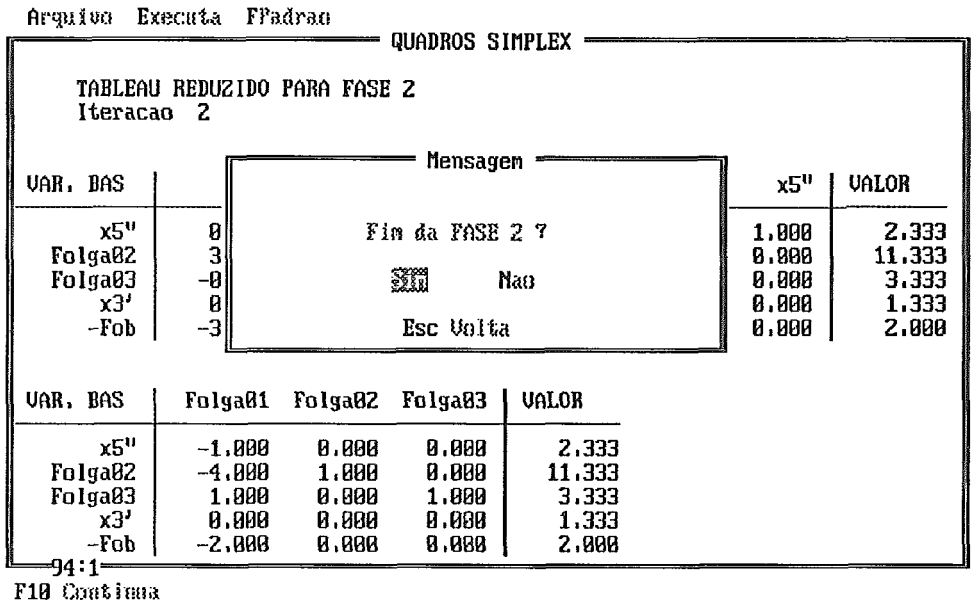


Figura 37

As Figuras de números 38 a 49 mostram a execução da terceira à sexta iteração do SIMPLEX.

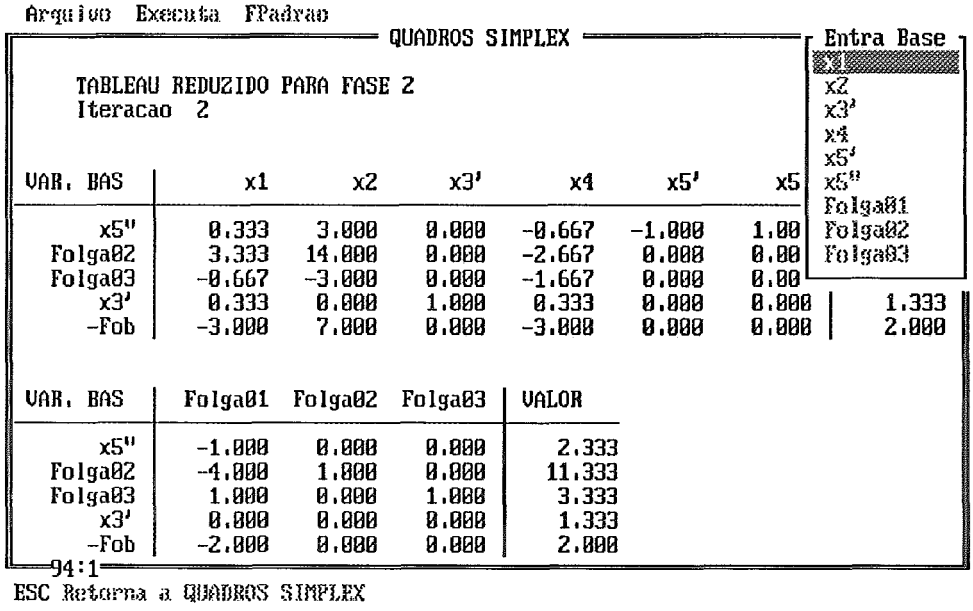


Figura 38

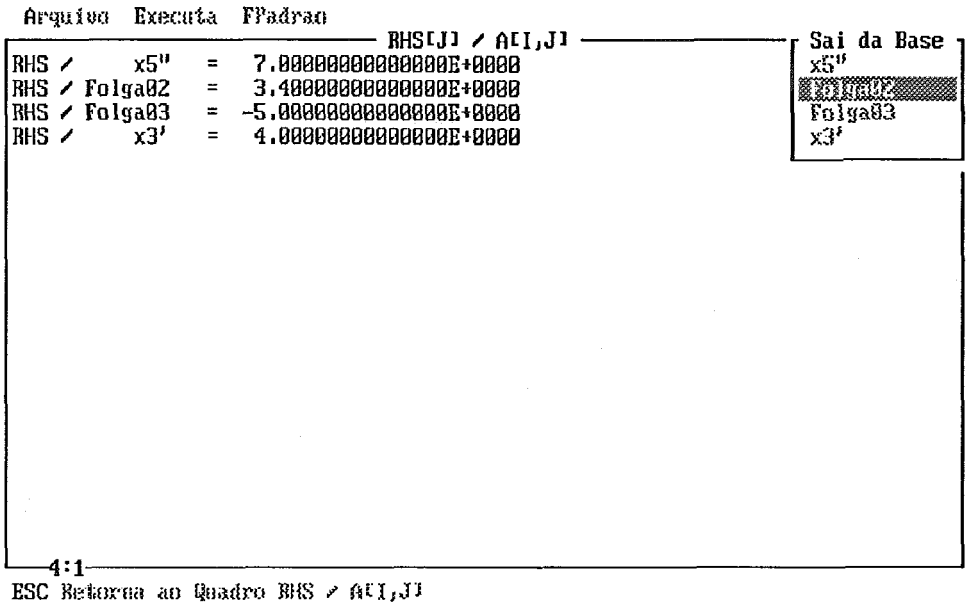


Figura 39

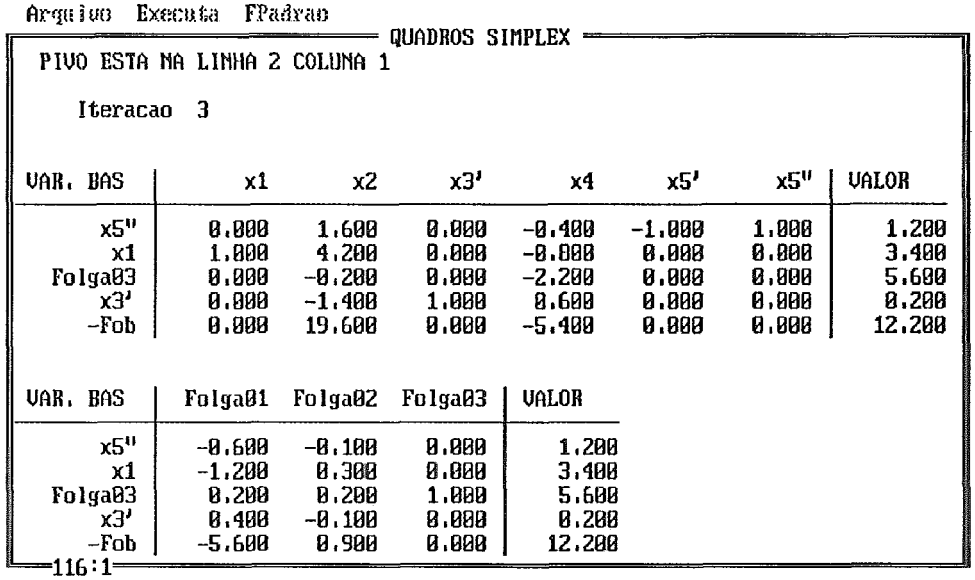


Figura 40

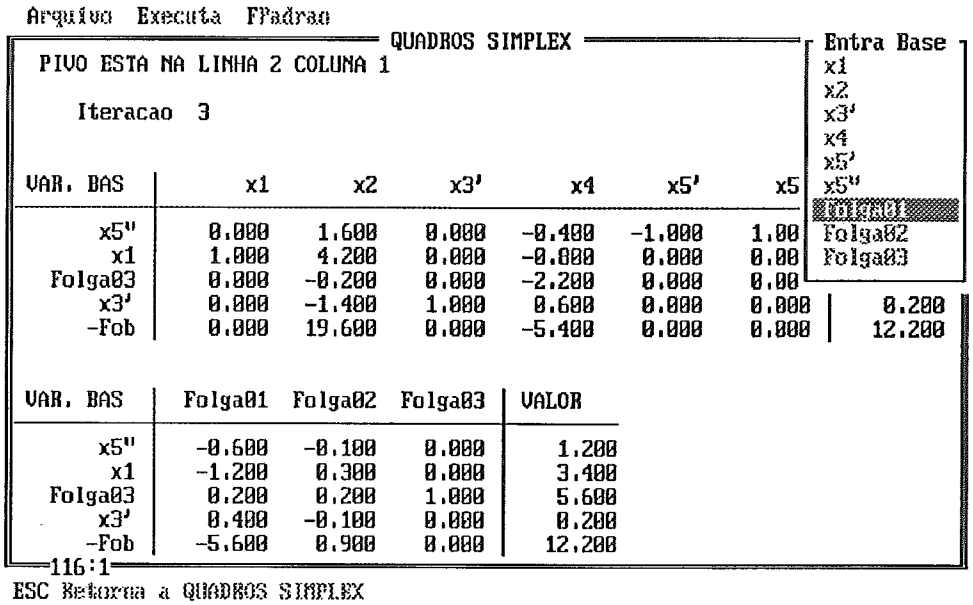


Figura 41

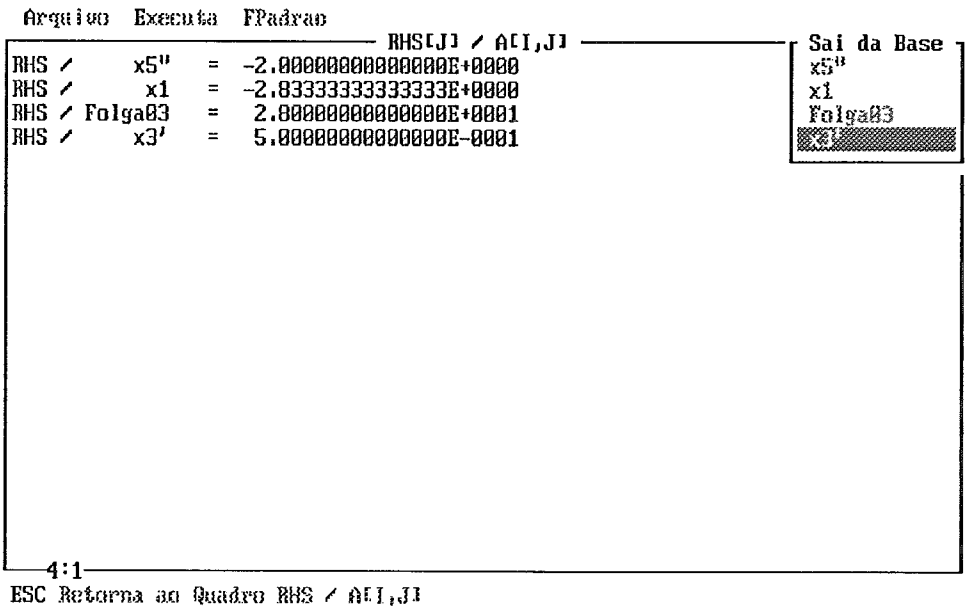


Figura 42

Arquivo Executa FPadrao

QUADROS SIMPLEX

PIVO ESTA NA LINHA 4 COLUNA 7

Iteracao 4

VAR. BAS	x1	x2	x3'	x4	x5'	x5"	VALOR
x5"	0.000	-0.500	1.500	0.500	-1.000	1.000	1.500
x1	1.000	0.000	3.000	1.000	0.000	0.000	4.000
Folga03	0.000	0.500	-0.500	-2.500	0.000	0.000	5.500
Folga01	0.000	-3.500	2.500	1.500	0.000	0.000	0.500
-Fob	0.000	0.000	14.000	3.000	0.000	0.000	15.000

VAR. BAS	Folga01	Folga02	Folga03	VALOR
x5"	0.000	-0.250	0.000	1.500
x1	0.000	0.000	0.000	4.000
Folga03	0.000	0.250	1.000	5.500
Folga01	1.000	-0.250	0.000	0.500
-Fob	0.000	-0.500	0.000	15.000

138:1

F10 Continua

Figura 43

Arquivo Executa FPadrao

QUADROS SIMPLEX

PIVO ESTA NA LINHA 4 COLUNA 7

Iteracao 4

VAR. BAS	x1	x2	x3'	x4	x5'	x5	VALOR
x5"	0.000	-0.500	1.500	0.500	-1.000	1.00	
x1	1.000	0.000	3.000	1.000	0.000	0.00	
Folga03	0.000	0.500	-0.500	-2.500	0.000	0.00	
Folga01	0.000	-3.500	2.500	1.500	0.000	0.000	0.500
-Fob	0.000	0.000	14.000	3.000	0.000	0.000	15.000

Entra Base
x1
x2
x3'
x4
x5'
x5"
Folga01
Folga02
Folga03

VAR. BAS	Folga01	Folga02	Folga03	VALOR
x5"	0.000	-0.250	0.000	1.500
x1	0.000	0.000	0.000	4.000
Folga03	0.000	0.250	1.000	5.500
Folga01	1.000	-0.250	0.000	0.500
-Fob	0.000	-0.500	0.000	15.000

138:1

ESC Retorna a QUADROS SIMPLEX

Figura 44

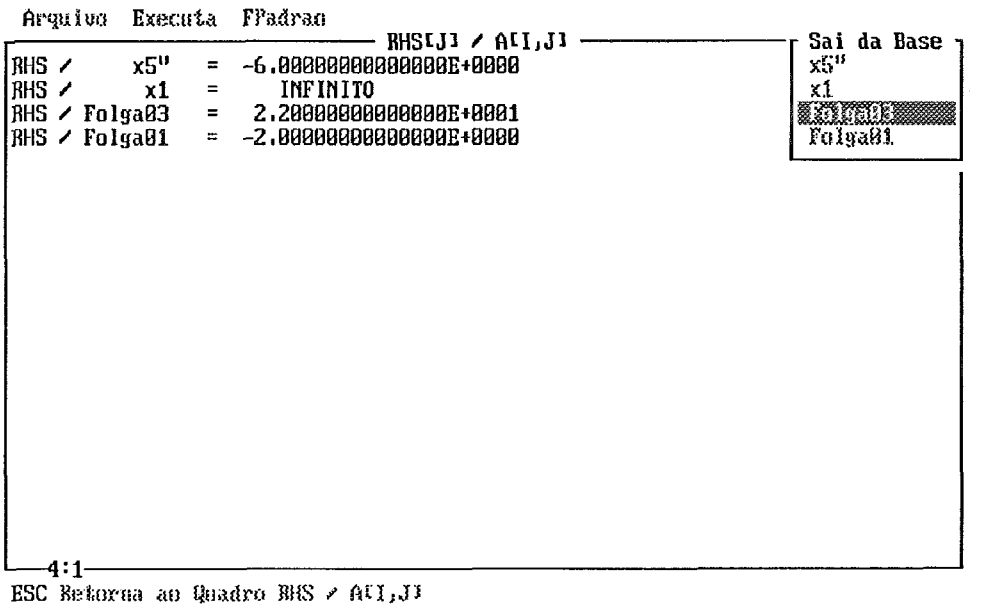
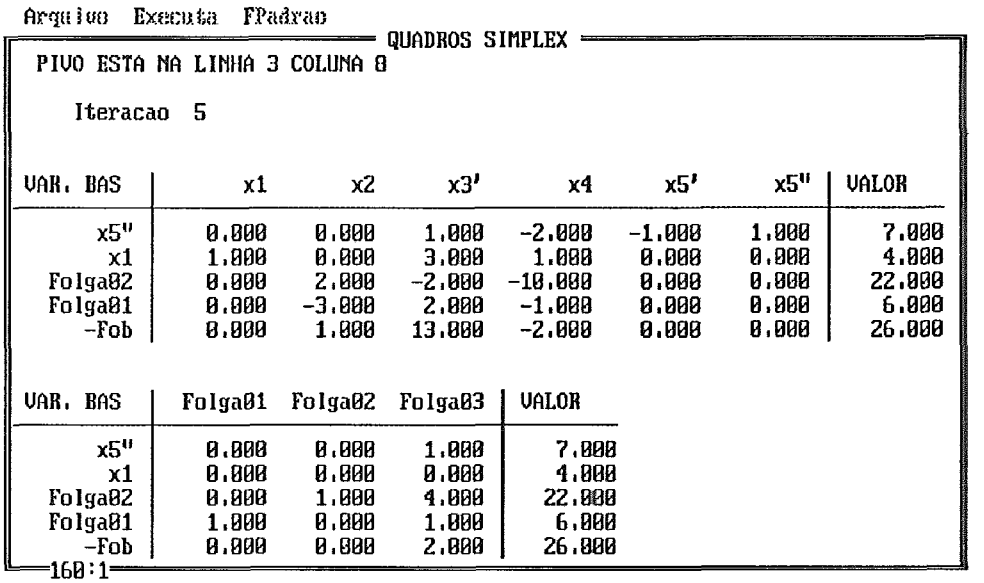


Figura 45



VAR. BAS	x1	x2	x3'	x4	x5'	x5"	VALOR
x5"	0,000	0,000	1,000	-2,000	-1,000	1,000	7,000
x1	1,000	0,000	3,000	1,000	0,000	0,000	4,000
Folga02	0,000	2,000	-2,000	-10,000	0,000	0,000	22,000
Folga01	0,000	-3,000	2,000	-1,000	0,000	0,000	6,000
-Fob	0,000	1,000	13,000	-2,000	0,000	0,000	26,000

VAR. BAS	Folga01	Folga02	Folga03	VALOR
x5"	0,000	0,000	1,000	7,000
x1	0,000	0,000	0,000	4,000
Folga02	0,000	1,000	4,000	22,000
Folga01	1,000	0,000	1,000	6,000
-Fob	0,000	0,000	2,000	26,000

Figura 46

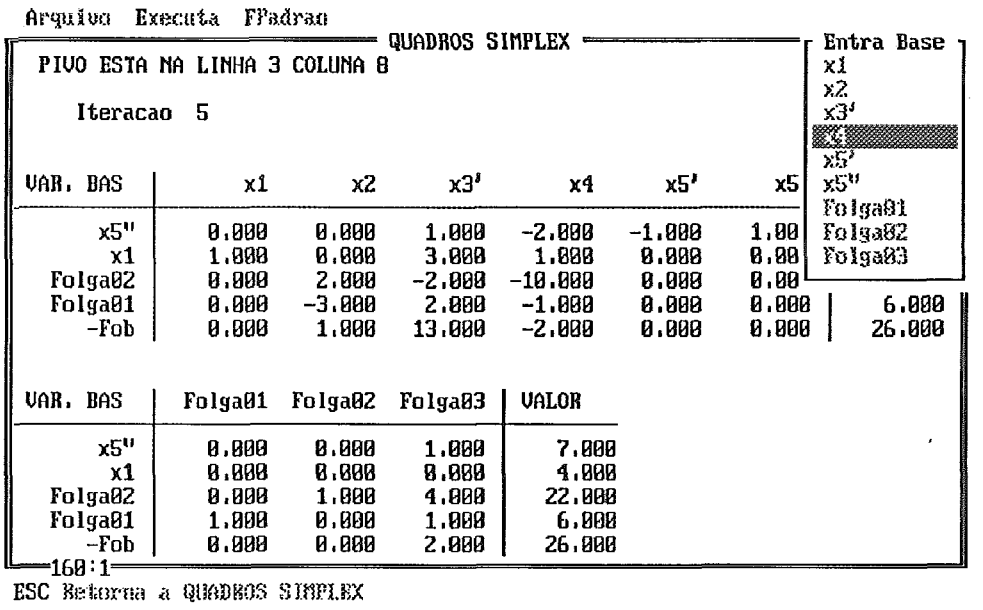


Figura 47

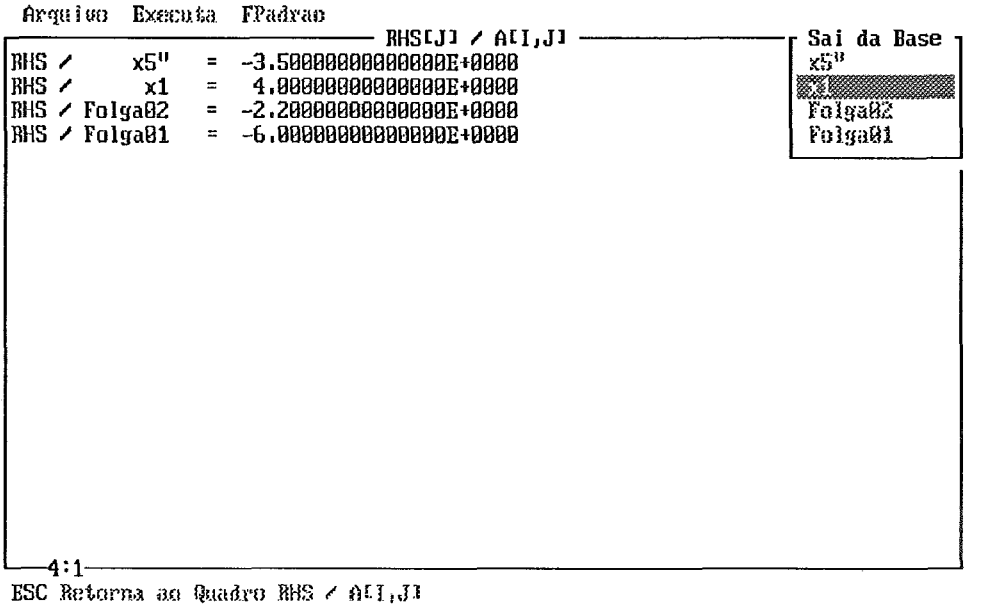


Figura 48

Observe na Figura 49, que todos os coeficientes da função objetivo do quadro SIMPLEX são positivos (maiores ou iguais a zero), isto indica que o algoritmo chegou a uma solução ótima. Assim sendo o usuário deve responder afirmativamente à pergunta apresentada na Figura 50.

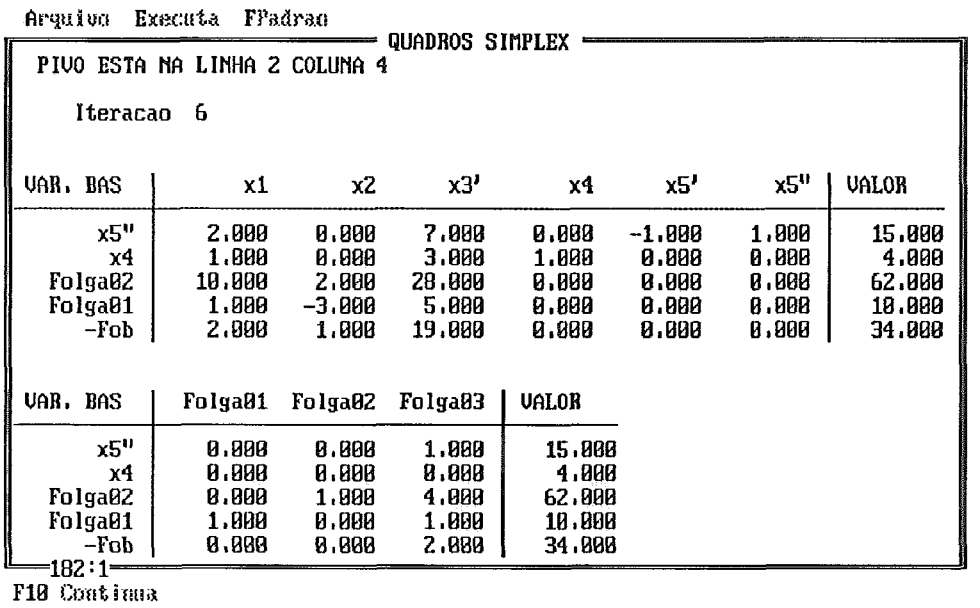


Figura 49

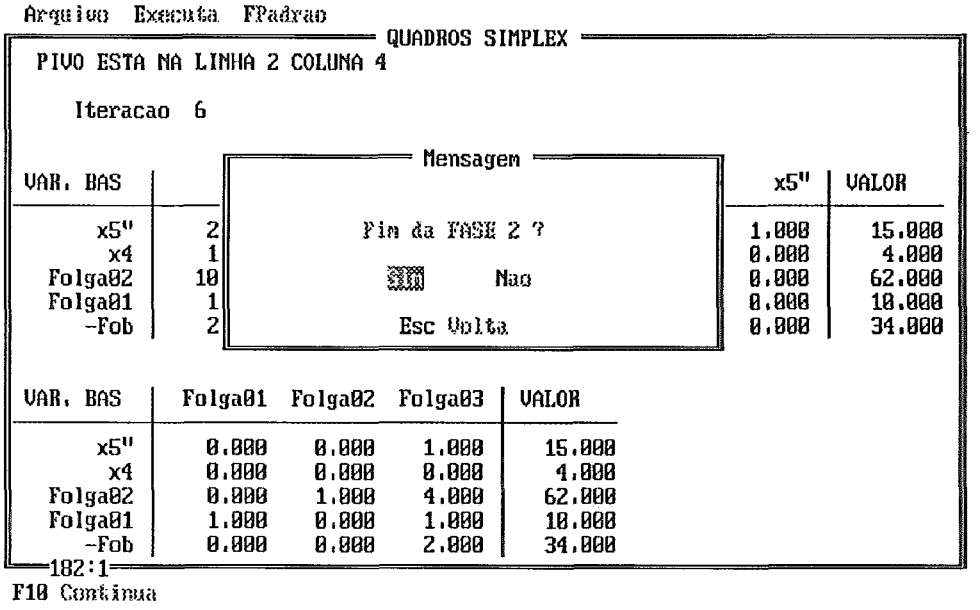


Figura 50

Após concluída a execução do SIMPLEX para um determinado problema, o sistema apresenta o valor da função objetivo e das variáveis básicas bem como uma estatística dos erros cometidos pelo usuário em todas as etapas, como mostra a Figura 51.

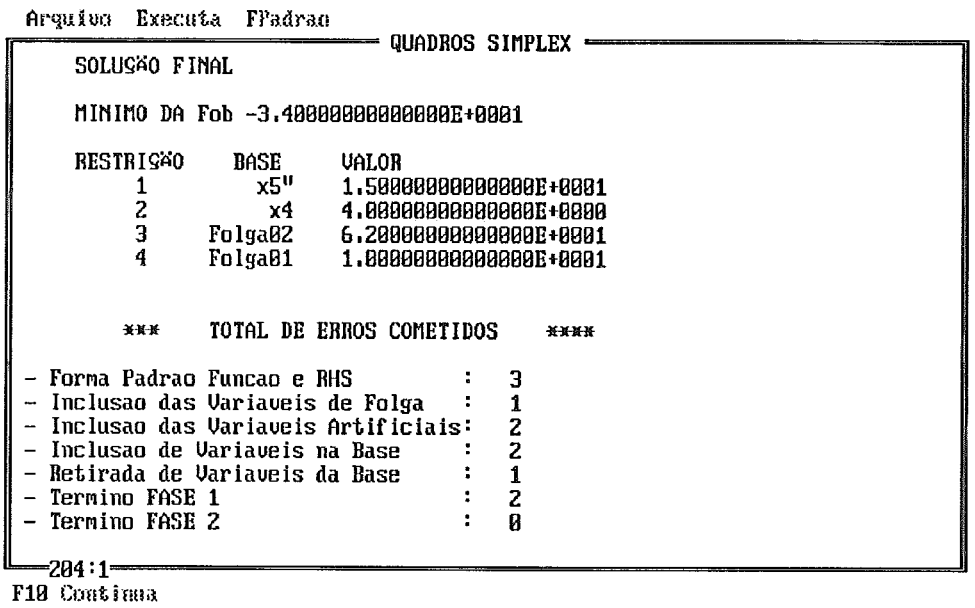


Figura 51

Finalmente se o usuário instruiu o sistema a organizar os quadros para impressão, Figura 21, este solicitará se deve ou não iniciar a impressão dos mesmos, Figura 52. Esta mensagem tem por objetivo permitir ao mesmo preparar a impressora, dar início ou cancelar a impressão.

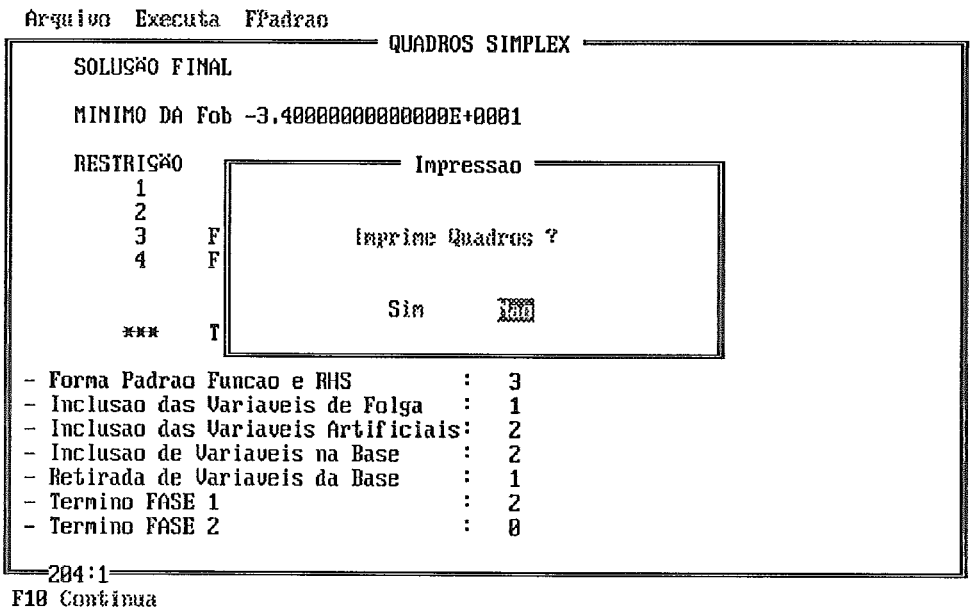


Figura 52

CAPÍTULO 5

CONCLUSÕES

Foram realizados vários testes com problemas de diversas dimensões e seus resultados comparados com os obtidos através do uso do LINDO. Para problemas de maior porte (trinta variáveis e vinte restrições) verificamos a necessidade do uso das rotinas para tratamento dos erros de arredondamento descritas no Capítulo 3. Para problemas com poucas variáveis e restrições a diferença entre os resultados não foi verificada.

Com relação aos tempos de execução de cada iteração do algoritmo SIMPLEX obtidos em um micro computador AT-386 DX-40, são ínfimos para problemas com até dez variáveis e dez restrições (menos de um segundo), e bastante rápidos para problemas maiores, com até trinta variáveis e vinte restrições (cerca de dois segundos).

Embora não disponha da eficiência e confiabilidade de um pacote comercial para a resolução de problemas de programação linear, o SIAMPLEX pode ter sua utilidade na resolução de problemas de pequeno porte. O seu principal valor, no entanto, é didático, podendo ser uma ferramenta útil no aprendizado do método SIMPLEX.

Para finalizar, destacamos que o uso das técnicas de programação utilizadas no desenvolvimento do ambiente SIAMPLEX, permitem que este seja facilmente estendido. Posteriormente pode ser feita a inclusão do algoritmo Dual-Simplex e técnicas para análise de sensibilidade dos resultados do problema.

REFERÊNCIAS BIBLIOGRÁFICAS

Aho, Alfred V. [1972] "Theory of Parsing, Translation and Compiling", Englewood Cliffs, Prentice-Hall, N. J.

Andrade, E. L. [1990] "Introdução à Pesquisa Operacional: métodos e modelos para análise de decisão", LTC, Rio de Janeiro.

Bazaraa, M. S., J. J. Jarvis, and H. D. Sherali [1990] "Linear Programming and Network Flows", 2nd ed., John Willey and Sons, Inc., New York.

Bazaraa, M. S., and C. M. Shetty. [1976] "Nonlinear Programming: Theory and Algorithms", John Willey and Sons, Inc., New York.

Bunday, B. D. and Garside, G. R. [1987] "Linear Programming in Pascal", Edward Arnold Publishers Ltd, London.

Bregalda P. F., A. F. de Oliveira, Cláudio T. Bornstein. [1988] "Introdução a Programação Linear", 3. ed., Campos, Rio de Janeiro.

Chvátal, V. [1983] "Linear Programming", W. H. Freeman and Company.

Dantzig, G. B. [1963] "Linear Programming and Extensions", Princeton University Press, Princeton, N.J.

Dantzig, G. B. [1982] "Reminiscences About the Origins of Linear Programming", Operations Research Letters, 1 (2), pp. 43-48, April.

Gill, P. E., Murray, W. and Wright, M. H. [1991] "Numerical Linear Algebra and Optimization", Addison-Wesley Publishing Company,, CA.

Gonzaga , C. D.[1987] "An Algorithm for Solving Linear Programming Problems in $O(n^3L)$ Operations" , Memo No. UCBL/ERL M87/10, Eletronics Research Laboratory, College of Engineering, University of California, Berkley, CA 94720.

Hadley, G.[1982] "Programação Linear", Editora Guanabara Dois, RJ.

Hillier, F. S., and G. J. Lieberman.[1986] Introduction to Operations Research, 4th ed., Holden-Day, Inc. , San Francisco.

Karmarkar, N. [1984] "A New Polynomial-Time Algorithm for Linear Programming" Combinatorica, 4, pp. 373-395.

Khachian, L. G. [1979] "A Polynomial Algorithm in Linear Programming" USSR Computational Mathematics and Mathematical Physics, 20, pp. 53-72.

Lasdon, L. S. [1970], "Optimization Theory for Large Systems, Macmillan, New York.

Lindo [1991] "User's Manual", Linus Schrage ed. The Scientific Press.

Luenberger, D. G. [1984] "Linear and Nonlinear Programming", Addison_Wesley Publishing Company, Inc.

Minus [1983] "User's Guide" Bruce A. Murtagh and Michael A. Sanders, Technical Report, Dec 1983.

Murtagh, B. A. [1981] "Advanced Linear Programming", McGraw-Hill, New York.

Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling W. T. [1989] "Numerical Recipes in FORTRAN", Cambridge University Press, London.

Simonnard, M. A. [1976] "Linear Programming", Prentice-Hall, Englewood Cliffs, N.J.

Turbo Pascal [1990] "Turbo Vision Guide" e "User's Manual", Borland International, Inc.

Wirth Niklaus [1985] "Programação Sistemática em Pascal", 4. ed., Campos, Rio de Janeiro.