UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA DE CAMPINAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

MÉTODO PARA RESOLVER UM PROBLEMA DE PROGRAMAÇÃO LINEAR DINÂMICA

EUNICE LUVIZOTTO MEDINA PISSOLATO

055/82

Orientadores

CELSO PASCOLI BOTTURA

HERMANO DE MEDEIROS FERREIRA TAVARES

Tese de Mestrado apresentada à Faculdade de Engenharia da Universidade Estadual de Campinas.

1982

UNICAMP
BIBLIOTECA

Aos meus pais,
Antonio (in memorian) e Wilma.
Ao Zé.

AGRADECIMENTOS

Agradeço sinceramente a todos aqueles que contribuiram, direta ou indiretamente, para a realização deste trabalho, salientando em especial:

- ao HERMANO, orientador e amigo, pelo acompanhamento do trabalho e profunda dedicação;
- ao BOTTURA, pela orientação e incentivo no direcionamento da tese;
- à ANA, pela grande amizade, além da participação e discussões que foram essenciais na realização deste trabalho;
- ao FRANÇA, JURA, CHRISTIANO, AKEBO, RAUL, DINO, PAULINHO, FLAVIO e ANILTON, na UNICAMP e NERY, DINA, MILTINHO, DAVID, QUIRINO, AMORIN, DIB, ANTONIO e SHEILA em Ilha Solteira, pela convivência muito agradável e apoio sempre recebido;
- ao HAROLDO e demais colegas do Departamento de Engenharia Elétrica da FEIS por possibilitarem meu afastamento para concluir este trabalho;
- e finalmente à ELZA e à JULIA, pelo trabalho de datilografia e ao HEIJI pelos desenhos.

ÍNDICE

Pág.

I N T R O D U Ç Ã O

O impacto dos métodos e modelos de Programação Linear em tomadas de decisões é bastante conhecido. Entretanto, sua aplicação tem sido, para a maioria dos problemas, estática e com somente um estágio sendo que, quando o sistema a ser otimizado se desenvolve no tempo, uma aproximação para um estágio único é inadequada. Neste caso, as decisões são estabelecidas sobre o tempo e o problema de otimização torna-se dinâmico, com múltiplos estágios. Dentro do contexto de Programação Linear Dinâmica (PLD) surgem novos problemas. Um grande número de problemas de planejamento e controle de sistemas econômicos, de organização, tecnológicos e outros, reduzem-se a problemas de PLD.

Até há pouco tempo, esses problemas eram resolvidos reduzindo-os a problemas estáticos e usando "pacotes" padrões de Programação Linear. Desde que os problemas de PLD são, por sua natureza, de grande porte, essas aproximações eram limitadas em suas possibilidades, passando a haver a necessidade de algoritmos e programas correspondentes, especialmente orientados para levar em conta sua natureza específica.

Um avanço foi verificado com o desenvolvimento de algoritmos eficientes, do tipo simplex , para resolver problemas lineares de grande dimensão , baseados em métodos de fatoração da matriz básica, levando em conta sua esparsidade.

Neste trabalho é desenvolvido um algoritmo utilizando o método simplex revisado, para resolver um proble

ma de PLD de grande porte, baseado no trabalho clássico de BARTELS e GOLUB |2|, que usa a decomposição LU de uma matriz. Os problemas de PLD apresentam uma matriz de restrições com uma estrutura particular chamada escada (staircase).

Desenvolvemos um programa computacional para resolver o algoritmo mencionado, sendo que neste desenvolvimento, devemos ressaltar os seguintes pontos:

- procura de técnicas de armazenamento adequadas à estrutura particular e ao algoritmo;
- adaptação da resolução dos passos usuais do Simplex, para aproveitar a estrutura escada, compatibilizando-a com os esquemas de armazenamento escolhidos. Alguns destes esquemas são conhecidos e outros foram por nós desenvolvidos, especificamente para o esquema tratado;
- estabelecimento de um compromisso entre a manutenção da estrutura escada e a estabilidade numérica.

Assim, apresentamos no capítulo 1 o algoritmo e o diagrama de blocos de um programa implementado para o método do simplex revisado com decomposição LU da base, que desenvolvemos para uma matriz básica qualquer, sem levar em conta esparsidade ou estrutura especial. A implementação deste programa deu subsídios ao desenvolvimento e implementação do programa para resolver o problema de PLD.

No capítulo 2 é apresentado em detalhes o algoritmo desenvolvido para resolver o problema de PLD proposto, mostrando a preocupação em manter a estrutura e assegurar estabilidade numérica.

As técnicas utilizadas para o armazenamento esparso são apresentadas no capítulo 3, mostrando através de exemplos, os arranjos que caracterizam as matrizes e os vetores do problema.

Finalmente, no capítulo 4, apresentamos o diagrama de blocos do programa que implementamos para o algoritmo desenvolvido no capítulo 2, dando os detalhes dos blocos do diagrama, principalmente nas partes em que ele difere do simplex clássico, desenvolvidas para este problema específico. É apresentado um exemplo, para ilustrar os passos do algoritmo a cada iteração.

CAPÍTULO 1

O MÉTODO SIMPLEX REVISADO COM DECOMPOSIÇÃO LU DA BASE

1.1. O método simplex revisado

Seja o seguinte problema de programação linear:

$$\text{Max} \quad Z = cx$$
$$\text{sujeito a} \quad Ax = b$$
$$\text{e} \quad x \geq 0$$

onde c é o vetor de custos, 1 x n
x é o vetor solução, n x 1
A é a matriz de restrições, m x n
b é o vetor de recursos, m x 1
0 é um vetor nulo, n x 1

Supondo conhecida uma base inicial para o problema, seja I o conjunto de índices básicos e J, o conjunto de índices não básicos, referentes à matriz de restrições A; chamaremos B à matriz básica, m x m, constituída pelas colunas associadas ao conjunto I e A^J à matriz constituída pelas colunas não básicas. Uma iteração do simplex revisado consiste dos seguintes passos:

Passo 1: Calcule a solução básica atual, através de $Bx_I = b$, onde x_I é o vetor solução básica.
Seja $x_I = x^o$ a solução;

Passo 2: Calcule o vetor de multiplicadores π, através de $B^T\pi = (c^I)^T$, onde c^I é o vetor de custos básicos. Faça $\hat{c}^J = c^J - \pi A^J$, onde c^J é o vetor de custos não básicos

cos. Se $\hat{c}^J \geq 0$, a solução é ótima;

Passo 3: Selecione uma variável não básica a entrar na base, isto é, uma para a qual $\hat{c}^j < 0$, $j \in J$. Calcule a representação da coluna A^j em termos da base B, resolvendo $B\hat{A}^j = A^j$;

Passo 4: Determine a variável a sair da base;

Passo 5: Atualize a base B. Retorne ao passo 1.

1.2. A decomposição LU da base

No algoritmo simplex é preciso resolver, a cada iteração, os sistemas lineares:

$$\begin{aligned} B\, x_I &= b \\ B\, \hat{A}^j &= A^j \\ B^T \pi &= (c^I)^T \end{aligned} \qquad (1.1)$$

Resolvemos estes sistemas decompondo inicialmente a matriz B em $B = LU$, onde L é uma matriz triangular inferior com 1's na diagonal e U é triangular superior.

No início do algoritmo faz-se uma decomposição da matriz B dada, escolhendo sempre como pivô o elemento de maior valor absoluto de cada coluna, para assegurar maior estabilidade numérica (BARTELS & GOLUB [2]). Para isto, pode ser necessário fazer permutações de linhas, obtendo-se na realidade, uma fatoração do tipo:

$$P\, B = LU$$

onde P é uma matriz de permutações.

Em cada iteração do simplex, a base B difere da base correspondente à iteração anterior somente em uma coluna. Queremos obter a fatoração da nova base a partir da fatoração da base anterior. Seja $B^{(\ell)}$ a base na iteração ℓ. Vamos supor que na iteração $(\ell + 1)$ sai a coluna r da base (r indica a posição da coluna na base e não na matriz de restrições). Seja A^s a coluna a entrar na base.

$$B^{(\ell)} = (B^1, B^2, \ldots, B^r, \ldots B^m) = L^{(\ell)} U^{(\ell)}$$

Vamos considerar $B^{(\ell+1)}$ da seguinte maneira:

$$B^{(\ell+1)} = (B^1, B^2, \ldots, B^{r-1}, B^{r+1}, \ldots, B^m, A^\delta)$$

Então:

$$(L^{(\ell)})^{-1} B^{(\ell+1)} = (U^1, \ldots, U^{r-1}, U^{r+1}, \ldots, U^m, (L^{(\ell)})^{-1} A^\delta) = H^{(\ell)}$$

A matriz H tem a seguinte estrutura:

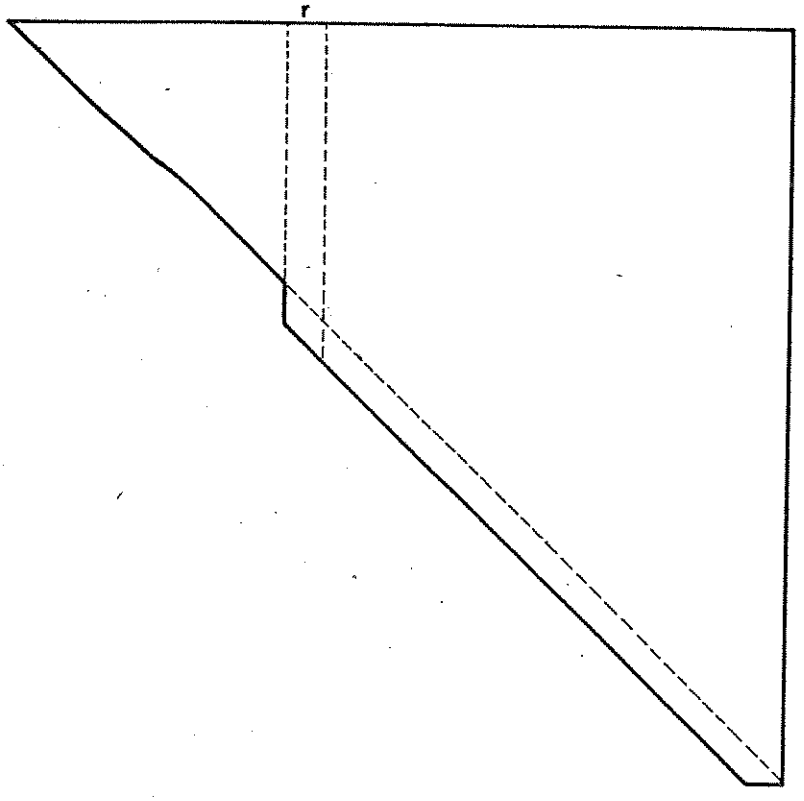


Figura 1.1.

Esta matriz não é triangular superior, pois tem elementos não nulos na sub-diagonal inferior, a partir da coluna r.

O próximo passo é, então, triangularizar a matriz $H^{(\ell)}$. É claro que para isto basta zerar os elementos da sub-diagonal, a partir da coluna r. Isto é feito com apenas uma operação de pivoteamento para cada coluna, a qual tem, como candidatos a pivô, os elementos $h_{i,i}$ e $h_{i+1,i}$, $i = r, \ldots, m-1$. Aqui também, para que o algoritmo seja mais estável, escolhemos como pivô aquele de maior valor absoluto. A triangularização de $H^{(\ell)}$ é obtida, portanto, mediante uma sucessão de permutações e pivoteamentos, isto é,

$$\text{Piv.Per.Piv....Piv.Per. } H^{(\ell)} = U^{(\ell+1)} \qquad (1.2)$$

num total de 2 $(m - r)$ operações. A matriz $U^{(\ell+1)}$ é triangular superior e Per é igual à identidade quando não foi necessária uma permutação.

Vamos chamar

$$G^{(\ell)} = \text{Piv.Per. } \ldots \text{ Piv.Per } (2(m-r) \text{ operações}) \tag{1.3}$$

Temos:

$$(L^{(\ell)})^{-1} B^{(\ell+1)} = H^{(\ell)}$$

$$G^{(\ell)} (L^{(\ell)})^{-1} B^{(\ell+1)} = G^{(\ell)} H^{\ell} = U^{(\ell+1)}$$

$$B^{(\ell+1)} = L^{(\ell)} (G^{(\ell)})^{-1} U^{(\ell+1)}$$

$$B^{(\ell+1)} = L^{(\ell+1)} U^{(\ell+1)}$$

onde $$L^{(\ell+1)} = L^{(\ell)} (G^{(\ell)})^{-1}$$

Obtivemos uma decomposição de $B^{(\ell+1)}$. O problema é que, devido às permutações de linhas em $H^{(\ell)}$, as modificações sobre L fazem com que $L^{(\ell+1)}$ deixe de ser triangular inferior. Como nosso objetivo é resolver os sistemas através da resolução de sistemas triangulares, mostramos agora como isso pode ser feito:

1.2.1. Cálculo da solução básica

Seja o sistema

$$B^{(\ell+1)} x_I = b \tag{1.4}$$

Chamemos de G o conjunto de todas as sucessões de permutações e pivoamentos realizados nas iterações anteriores, isto é:

$$G = G^{(\ell)} G^{(\ell-1)} \ldots\ldots G^{(2)} G^{(1)} \tag{1.5}$$

O sistema (1.4) é equivalente a

$$L (G^{(1)})^{-1} \ldots\ldots (G^{(\ell)})^{-1} U^{(\ell+1)} x_I = b$$

$$L G^{-1} U^{(\ell+1)} x_I = b$$

$$G^{-1} U^{(\ell+1)} x_I = L^{-1} b$$

Se chamarmos $L^{-1} b = y$, temos que y é a solução do sistema triangular inferior:

$$Ly = b$$

Consequentemente, para obtermos x_I é preciso resolver:

$$G^{-1}\ U^{(\ell+1)}\ x_I = y$$

que equivale a

$$U^{(\ell+1)}\ x_I = G\ y$$

Seja $\hat{y} = G\ y$ o vetor que resulta de fazermos sobre y todas as transformações feitas em $U^{(\ell+1)}$. Então, o sistema triangular superior a resolver é

$$U^{(\ell+1)}\ x_I = \hat{y}$$

Em vez de transformar L, isto é, aplicar todas as operações representadas por G sobre uma matriz, transformamos a solução de Ly = b, ou seja, aplicamos as operações indicadas em G sobre um vetor.

A mesma observação é válida no caso do sistema $B\ \hat{A}^j = A^j$, a cada iteração.

1.2.2. Cálculo dos multiplicadores

Para resolver o sistema

$$(B^{\ell+1})^T\ \pi = (c^I)^T$$

temos

$$(L\ G^{-1}\ U^{(\ell+1)})^T\ \pi = (c^I)^T$$

$$(U^{(\ell+1)})^T\ (G^{-1})^T\ L^T\ \pi = (c^I)^T$$

Seja Z o vetor solução do sistema triangular inferior

$$(U^{(\ell+1)})^T\ Z = (c^I)^T$$

Então, $Z = (G^{-1})^T L^T\ \pi$

ou $G^T Z = L^T\ \pi$

Chamemos de $\hat{Z}$ o vetor que resulta de aplicarmos G^T sobre Z, isto é

$$\hat{Z} = G^T Z$$

e então, o sistema triangular superior a ser resolvido é:

$$L^T\ \pi = \hat{Z}$$

Como no caso anterior, transformamos o vetor Z em vez de transformarmos a matriz L, lembrando que aplicamos

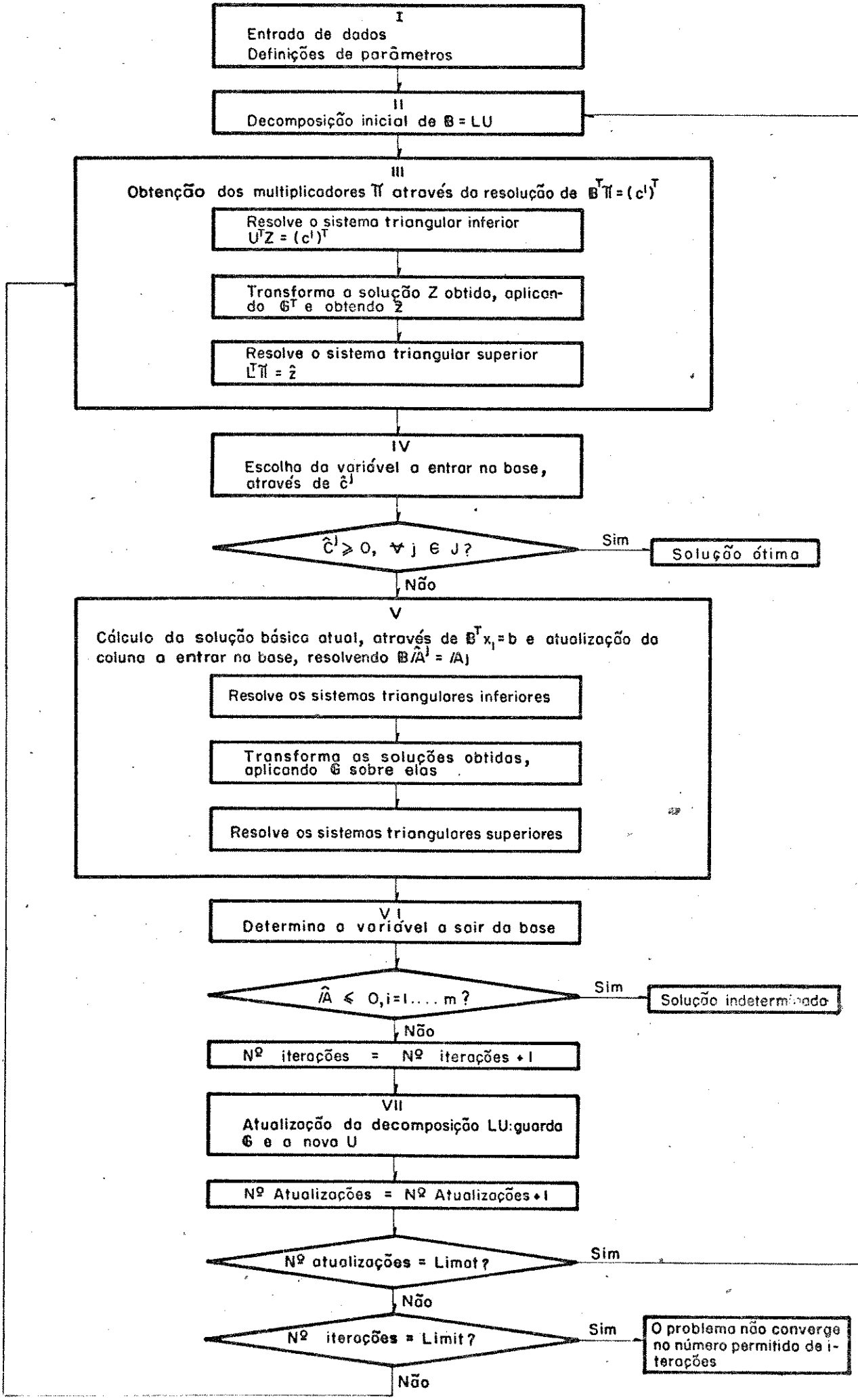
a transposta de G, o que equivale a aplicar as operações feitas em U na ordem inversa, ou seja, de trás para frente e ainda mais, considerando as operações de transposição.

1.3. Diagrama de blocos do algoritmo

Este algoritmo foi implementado no programa SIMPLU.F4 (ou SILUCA.F4 no caso de variáveis canalizadas), supondo conhecida uma base B inicial, isto é, não implementamos uma Fase I ao algoritmo. Usamos variáveis artificiais, partindo da matriz identidade como base B inicial, quando esta não era conhecida de imediato.

O diagrama de blocos de algoritmo e os detalhes de cada bloco, são apresentados a seguir:

DIAGRAMA DE BLOCOS



1.4. Detalhes dos blocos do diagrama

BLOCO (I): Entrada de dados e definições de parâmetros

- Os dados devem ser lidos na seguinte ordem:

1. M,N
   M = número de restrições do problema
   N = número de variáveis, incluindo as de folga
2. A
   A = matriz de restrições, m x n, lida por colunas
3. C ( ), V ( )
   C ( ) = vetor de custos
   V ( ) = vetor de recursos (corresponde no programa, ao vetor b do problema)
4. XMAX ( ) , XMIN ( )

   Estes vetores são lidos somente em SILUCA.F4 (programa com variáveis canalizadas)
   XMAX (I) = limite superior da variável X (I)
   XMIN (I) = limite inferior da variável X (I)

   No caso de variáveis que são limitadas apenas inferiormente colocamos, na posição correspondente a ela em XMAX, um valor muito grande e se a variável tem apenas o limite superior, fazemos o mesmo, colocando um valor muito pequeno em XMIN.
5. INDB ( ), INDNB ( )

   INDB ( ) = vetor que contém os índices das variáveis básicas.
   INDNB ( ) = vetor que contém os índices das variáveis não básicas.
   No programa SILUCA.F4, INDNB ( ) é negativo se a variável não básica correspondente estiver no limite inferior e positivo em caso contrário.

- Os parâmetros definidos são os seguintes:

1. LIMIT = número máximo de iterações permitido
2. LIMAT = número máximo de atualizações permitido

A cada iteração, quando da atualização da base, são criadas novas matrizes do tipo $G^{(i)}$, correspondentes às operações de pivoteamento e permutações realizadas.

Depois de várias iterações, o trabalho de aplicar as operações anteriores para obter nova atualização pode ser o mesmo ou até maior que aquele correspondente a uma decomposição da base. Daí a importância de se definir LIMAT. Excedido o limite de atualizações, na iteração seguinte é feita uma nova decomposição LU da base atual.

BLOCO (II): Decomposição de B em LU

Para cada coluna MC de B, vamos zerar os elementos abaixo da linha MC. Procuramos como pivô o elemento de maior valor absoluto entre $B(MC,MC)$.....$B(M,MC)$ e permutamos a linha que tem esse máximo com a linha MC. Fazemos a seguir as operações de pivoteamento. Depois da permutação, o pivô corresponde ao elemento $B(MC,MC)$. Para zerar os elementos das linhas abaixo da linha MC, pré-multiplicamos a matriz B por uma matriz do tipo:

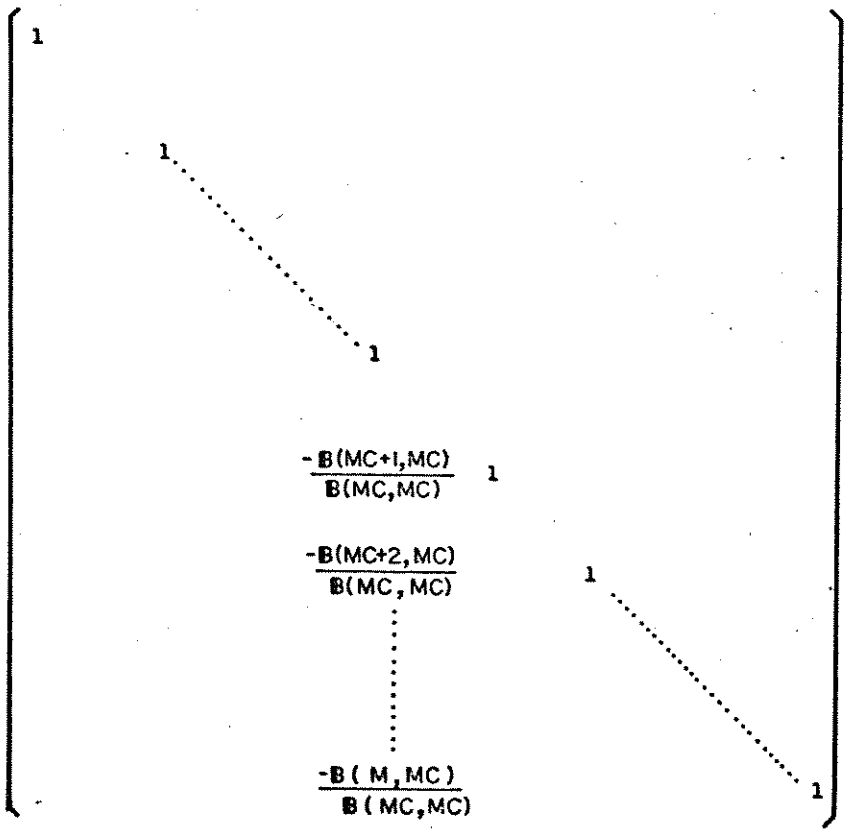


Figura 1.2.

Esta matriz corresponde à componente $(L^{(MC)})^{-1}$ da inversa da matriz L (lembrando que a matriz L é armazenada na forma produto) pois $L^{-1}PB = U$. Como a inversa de uma matriz triangular, com 1's na diagonal e apenas uma coluna com elementos não

nulos abaixo (ou acima) da diagonal, corresponde aos elementos da própria matriz multiplicados por - 1, guardamos no lugar dos zeros, os elementos de $(L^{(MC)})^{-1}$ multiplicados por -1, já que estes valores correspondem aos elementos da matriz L.

Deve-se observar que L vai sendo gerada ao mesmo tempo que U, que nada mais é que a matriz que fica no triângulo superior de B, incluindo a diagonal. Onde estava armazenada a base B inicial temos, no final da decomposição, no triângulo superior, incluindo a diagonal, os elementos da matriz U e no triângulo inferior as colunas da matriz L, lembrando que a diagonal de L é constituída de 1's.

Quando estamos fazendo as operações em uma coluna MC > 1, as colunas anteriores de L já foram geradas e é preciso permutar as linhas de L toda vez que forem permutadas linhas de B na procura do pivô adequado.

O vetor IPERM ( ), que inicialmente contêm os números de 1 a m, em ordem crescente, no final deste processo indica a ordem em que se encontram as linhas de B.

A seguir, para maior clareza, vamos mudar a ordem de detalhamento dos blocos, comentando o bloco (VII):

BLOCO (VII): Atualização da decomposição LU.

São gerados dois vetores paralelamente às transformações feitas em U, que é o triângulo superior de B. Esses dois vetores, INCSAI e ATUALI, contêm a informação necessária para reproduzir G.

INCSAI ( ): guarda as posições das colunas que saíram da base em cada iteração.

ATUALI ( ): guarda, em cada iteração, 2 $(m-r)$ valores, sendo r a posição da coluna que saiu nessa iteração. Desses valores, $(m-r)$ correspondem a pivoteamentos e

os $(m-n)$ restantes, a permutações.

As operações são guardadas na ordem indicada na equação (1.3), sendo que às operações Per corresponde valor zero se não houve permutação e *1* em caso contrário.

BLOCO (III): Obtenção dos multiplicadores

O sistema a resolver é:

$$(U^{(\ell+1)})^T \ (G^{-1})^T \ L^T \ \pi = (c^I)^T$$

Resolvemos primeiramente

$$(U^{(\ell+1)})^T \ Z = (c^I)^T$$

mediante uma subrotina para sistemas triangulares inferiores. Depois, se não foi feita nenhuma atualização da decomposição, isto é, $\ell = 0$, resolve-se a continuação:

$$Z = L^T \ \pi$$

obtendo-se os multiplicadores π. Se já foram feitas atualizações, precisamos transformar o vetor Z, fazendo:

$$\hat{Z} = G^T \ Z$$

A informação necessária para reproduzir G está em INCSAI e ATUALI. Como aqui é aplicada a transposta de G, as operações são feitas na ordem inversa, ou seja, primeiro aplicamos o último pivoteamento, depois a última permutação e assim consecutivamente, lembrando a transposição de cada matriz.

Com o vetor transformado, resolvemos o sistema

$$\hat{Z} = L^T \ \pi$$

mediante uma subrotina que resolve sistemas triangulares superiores, obtendo o vetor de multiplicadores π.

BLOCO (IV): Escolha da variável a entrar na base

A variável a entrar na base é a primeira que não satisfaz o teste de otimalidade.

BLOCO (V): Cálculo da solução básica e atualização da coluna a entrar na base

São resolvidos os sistemas:

$$Ly = b$$
$$Lv = A^j$$

Se a decomposição LU não foi atualizada, resolve-se:

$$Ux_I = y$$
$$U\hat{A}^j = v$$

obtendo-se os resultados desejados. Se já houveram atualizações anteriores, fazemos as transformações:

$$\hat{y} = G\, y$$
$$\hat{v} = G\, v$$

sendo que as informações para reproduzir G são obtidas em INCSAI e ATUALI. A seguir, resolvemos:

$$Ux_I = \hat{y}$$
$$U\hat{A}^j = \hat{v}$$

obtendo os valores de x_I e $\hat{A}^j$ desejados.

OBSERVAÇÃO: Como no cálculo da nova solução básica apenas uma coluna da base é alterada e o vetor de recursos continua o mesmo, resolvemos o sistema triangular apenas uma vez para obter a solução básica inicial, sendo que as soluções seguintes são calculadas a partir da anterior, trocando os valores da coluna.

BLOCO (VI): Escolha da variável a sair da base

É feita da maneira usual, tanto em SIMPLU.F4 como em SILUCA.F4. Para ilustrar, em SIMPLU.F4, escolhe-se a variável r tal que

$$\frac{\hat{b}r}{\hat{A}^j_r} = \min_{\hat{A}^j_i > 0} \left\{ \frac{\hat{b}i}{\hat{A}^j_i} \right\}$$

OBSERVAÇÃO: Existe um limite permitido de atualizações (LIMAT), que quando atingido, faz com que se proceda a uma nova decomposição da base B em LU.

1.5. Exemplo

Vamos agora resolver um exemplo que ilustra a aplicação do método. Seja o problema (P) a seguir:

$$(P)\quad \left[\begin{array}{ll} \text{Max} & x_4 + x_5 \\ \text{sujeito a} & x_1 - x_2 + 4x_3 - x_4 + 2x_5 = 3 \\ & 2x_1 + x_2 - x_3 + 3x_4 + x_5 = 3 \\ & -x_1 + 2x_2 + x_4 - x_5 = 1 \\ & 0 \leq x_1 \leq 3 \qquad 0 \leq x_4 \leq 7 \\ & 1 \leq x_2 \leq 5 \qquad 0 \leq x_5 \leq 7 \\ & 0 \leq x_3 \leq 10 \end{array}\right.$$

Como as variáveis são canalizadas, estaremos sempre nos referindo ao programa SILUCA.F4.

Os dados de entrada são os seguintes:

M = 3; N = 5

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

C(I) = (0.,0.,0.,-1.,-1.); V(I)= (3.,3.,1.)

XMAX (I) = (3.,5.,10.,7.,7.); XMIN (I) = (0.,1.,0.,0.,0.).

Supondo conhecida a base B inicial entramos com: INDB(I) = (1,2,3); INDNB (I) = (-4, -5)

$$B = \begin{bmatrix} 1. & -1. & 4. \\ 2. & 1. & -1. \\ -1. & 2. & 0. \end{bmatrix}$$

e o vetor de recursos "corrigido", isto é, descontado dos valores assumidos pelas variáveis não básicas, é igual a:

$$V = \begin{bmatrix} 3. \\ 3. \\ 1. \end{bmatrix} - \begin{bmatrix} -1. \\ 3. \\ 1. \end{bmatrix} \times 0. - \begin{bmatrix} 2. \\ 1. \\ -1. \end{bmatrix} \times 0. = \begin{bmatrix} 3. \\ 3. \\ 1. \end{bmatrix}$$

O vetor de recursos é o próprio vetor inicial, pois as variáveis não básicas estão no limite inferior, que é igual a zero para ambas.

a) Decomposição LU inicial

Escolhemos para cada coluna, o elemento de maior valor absoluto, fazendo a permutação correspondente.

Fazemos a primeira permutação em B, trocando a primeira e a segunda linhas, isto é, IPERM(1) = 2

$$B = \begin{bmatrix} 2. & 1. & -1. \\ 1. & -1. & 4. \\ -1. & 2. & 0. \end{bmatrix}$$

Vamos zerar os elementos b_{21} e b_{23}, sendo que os elementos $-b_{21}/b_{11}$ e $-b_{31}/b_{11}$ irão constituir a primeira coluna de L^{-1}, isto é, $(L^{-1})^1$

$$B' = \begin{bmatrix} 2.0 & 1.0 & -1.0 \\ 0.0 & -1.5 & 4.5 \\ 0.0 & 2.5 & -0.5 \end{bmatrix} \text{ e } (L^{-1})^1 = \begin{bmatrix} 1. & & \\ -0.5 & 1. & \\ 0.5 & & 1. \end{bmatrix}$$

Escolhemos agora, na segunda coluna, o maior elemento entre as linhas 2 e 3, e fazemos a permutação, tendo o cuidado de permutar $(L^{-1})^1$. Temos então que IPERM(2) = 3

$$B' = \begin{bmatrix} 2.0 & 1.0 & -1.0 \\ 0.0 & 2.5 & -0.5 \\ 0.0 & -1.5 & 4.5 \end{bmatrix} \text{ e } (L^{-1})^1 = \begin{bmatrix} 1. & & \\ 0.5 & 1. & \\ -0.5 & & 1. \end{bmatrix}$$

Fazemos então o pivoteamento, obtendo as matrizes $(L^{-1})^2$ e U:

$$U = \begin{bmatrix} 2.0 & 1.0 & -1.0 \\ & 2.5 & -0.5 \\ & & 4.2 \end{bmatrix} \text{ e } (L^{-1})^2 = \begin{bmatrix} 1. & & \\ & 1. & \\ & 0.6 & 1. \end{bmatrix}$$

A matriz L^{-1} é obtida quando fazemos o produto de $(L^{-1})^1$ e $(L^{-1})^2$, isto é:

$$L^{-1} = (L^{-1})^2 \ (L^{-1})^1$$

e L é o produto das inversas de cada componente de L^{-1}:

$$L = ((L^{-1})^2 \ (L^{-1})^1)^{-1} = L^1 L^2 =$$

$$\begin{bmatrix} 1. & & \\ -0.5 & 1. & \\ 0.5 & & 1. \end{bmatrix} \begin{bmatrix} 1. & & \\ & 1. & \\ & -0.6 & 1. \end{bmatrix} = \begin{bmatrix} 1. & & \\ -0.5 & 1. & \\ 0.5 & -0.6 & 1. \end{bmatrix}$$

O vetor de permutações é igual a: IPERM (I) = (2,3,1)

b) Obtenção dos multiplicadores π

Vamos resolver o sistema

$$B^T\pi = (c^I)^T$$

Para isso, resolvemos inicialmente

$$U^T Z = (c^I)^T$$

$$\begin{bmatrix} 2.0 & & \\ 1.0 & 2.5 & \\ -1.0 & -0.5 & 4.2 \end{bmatrix} Z = \begin{bmatrix} 0. \\ 0. \\ 0. \end{bmatrix} \longrightarrow Z = \begin{bmatrix} 0. \\ 0. \\ 0. \end{bmatrix}$$

Como não houve nenhuma atualização, resolvemos o sistema:

$$L^T\pi = Z \rightarrow \pi = \begin{bmatrix} 0. \\ 0. \\ 0. \end{bmatrix} = \pi$$

c) Escolha da variável a entrar na base

Escolhemos a primeira que não satisfaz o critério de otimalidade: x_4

d) Cálculo da solução básica atual e atualização da coluna correspondente a x_4

Resolvemos dois sistemas triangulares inferiores, tomando o cuidado de permutar as colunas, segundo as indicações do vetor IPERM.

$$L\upsilon = A^4$$

$$\begin{bmatrix} 1. & & \\ -0.5 & 1. & \\ 0.5 & -0.6 & 1. \end{bmatrix} \upsilon = \begin{bmatrix} 3. \\ 1. \\ -1 \end{bmatrix} \longrightarrow \upsilon = \begin{bmatrix} 3.0 \\ 2.5 \\ -1.0 \end{bmatrix}$$

$$Ly = V$$

$$\begin{bmatrix} 1. & & \\ -0.5 & 1. & \\ 0.5 & -0.6 & 1. \end{bmatrix} y = \begin{bmatrix} 3. \\ 1. \\ 3 \end{bmatrix} \longrightarrow y = \begin{bmatrix} 3.0 \\ 2.5 \\ 3.0 \end{bmatrix}$$

A seguir, resolvemos os dois sistemas triangulares superiores, pois a matriz U ainda não sofreu transformações.

$$UA^4 = \upsilon$$

$$\begin{bmatrix} 2.0 & 1.0 & -1.0 \\ & 2.5 & -0.5 \\ & & 4.2 \end{bmatrix} \quad \hat{A}^4 = \begin{bmatrix} 3.0 \\ 2.5 \\ -1.0 \end{bmatrix} \longrightarrow \hat{A}^4 = \begin{bmatrix} 0.9 \\ 0.9 \\ -0.2 \end{bmatrix}$$

$\hat{A}^4$ é a coluna atualizada correspondente a x_4

$$Ux_I = \hat{y}$$

$$\begin{bmatrix} 2.0 & 1.0 & -1.0 \\ & 2.5 & -0.5 \\ & & 4.2 \end{bmatrix} x_I = \begin{bmatrix} 3.0 \\ 2.5 \\ 3.0 \end{bmatrix} \longrightarrow x_I = \begin{bmatrix} 1.3 \\ 1.1 \\ 0.7 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

x_I é a solução básica atual.
A função objetivo é igual a:

$$Z = x_4 + x_5 = 0$$

e) Escolha da variável a sair da base

É feita da maneira usual para o simplex canalizado e x_2 é a variável escolhida para sair da base, indo para o limite inferior, isto é:

INDB (I) = (1,3,4) e INDNB(J) = (-2,-5)

f) Cálculo do novo vetor de recursos, considerando as permutações indicadas em IPERM.

Como x_4 era não básica, seu valor foi descontado do vetor de recursos anterior. Então agora devemos adicionar esse valor ao vetor V e tirar o valor assumido por x_2, que passou a ser não básica, isto é:

$$V = \begin{bmatrix} 3. \\ 1. \\ 3. \end{bmatrix} + \begin{bmatrix} 3. \\ 1. \\ -1. \end{bmatrix} \times 0. - \begin{bmatrix} 1. \\ 2. \\ -1. \end{bmatrix} \times 1. \longrightarrow V = \begin{bmatrix} 2. \\ -1. \\ 4. \end{bmatrix}$$

e o novo vetor de custos básicos passa a ser:

$$c^I = (0.,0.,-1.)$$

g) Atualização da decomposição LU

$$U = \begin{bmatrix} 2.0 & 1.0 & -1.0 \\ & 2.5 & -0.5 \\ & & 4.2 \end{bmatrix}$$

x_2 é a variável a sair da base, logo: INCSAI (1)=2 e a matriz H^1 é igual a:

$$H^1 = \begin{bmatrix} 2.0 & -1.0 & 3.0 \\ & -0.5 & 2.5 \\ & 4.2 & -1.0 \end{bmatrix}$$

Vamos agora triangularizar a matriz H^1, fazendo as operações Piv.Per...., gerando a matriz G. Essas operações estão indicadas em ATUALI, que neste caso é:

ATUALI(I)=(1.,0.12)

sendo que o número 1 indica que houve permutação e 0.12 é o valor pelo qual foi multiplicado o pivô para zerar o elemento.

Fazendo a operação de permutação indicada em ATUALI:

$$H = \begin{bmatrix} 2.0 & -1.0 & 3.0 \\ & 4.2 & -1.0 \\ & -0.5 & 2.5 \end{bmatrix}$$

e aplicando o pivoteamento, obtemos novamente uma matriz triangular superior

$$U^2 = \begin{bmatrix} 2.0 & -1.0 & 3.0 \\ & 4.2 & -1.0 \\ & & 2.4 \end{bmatrix}$$

h) Obtenção dos novos multiplicadores

Resolvemos o sistema

$$(U^2)^T Z = (c^I)^T$$

$$\begin{bmatrix} 2.0 & & \\ -1.0 & 4.2 & \\ 3.0 & -1.0 & 2.4 \end{bmatrix} Z = \begin{bmatrix} 0. \\ 0. \\ -1. \end{bmatrix} \longrightarrow Z = \begin{bmatrix} 0.00 \\ 0.00 \\ -0.42 \end{bmatrix}$$

Como a matriz U foi transformada, aplicamos as mesmas transformações em Z

$$\hat{Z} = G^T Z \longrightarrow \hat{Z} = \begin{bmatrix} 0.00 \\ -0.42 \\ -0.05 \end{bmatrix}$$

e resolvemos o sistema triangular superior

$$L^T \pi = \hat{Z}$$

$$\begin{bmatrix} 1.0 & -0.5 & 0.5 \\ & 1.0 & -0.6 \\ & & 1.0 \end{bmatrix} \pi = \begin{bmatrix} 0.00 \\ -0.42 \\ -0.05 \end{bmatrix} \longrightarrow \pi = \begin{bmatrix} -0.20 \\ -0.45 \\ -0.05 \end{bmatrix}$$

i) Escolha da variável a entrar na base

A primeira que não satisfaz o critério de otimalidade é a variável x_5.

j) Cálculo da solução básica atual e atualização da coluna correspondente a x_5

Resolvemos os dois sistemas triangulares inferiores:

$$L\upsilon = A^5 \longrightarrow \begin{bmatrix} 1.0 & & \\ -0.5 & 1.0 & \\ 0.5 & -0.6 & 1.0 \end{bmatrix} \quad \upsilon = \begin{bmatrix} 1.0 \\ -1.0 \\ 2.0 \end{bmatrix} \longrightarrow \upsilon = \begin{bmatrix} 1.0 \\ -0.5 \\ 1.2 \end{bmatrix}$$

$$Ly = V \longrightarrow \begin{bmatrix} 1.0 & & \\ -0.5 & 1.0 & \\ 0.5 & -0.6 & 1.0 \end{bmatrix} \quad y = \begin{bmatrix} 2.0 \\ -1.0 \\ 4.0 \end{bmatrix} \longrightarrow y = \begin{bmatrix} 2.0 \\ 0.0 \\ 3.0 \end{bmatrix}$$

e transformamos as soluções obtidas, aplicando G:

$$\hat{\upsilon} = G\upsilon \longrightarrow \hat{\upsilon} = \begin{bmatrix} 1.0 \\ 1.2 \\ -0.4 \end{bmatrix}$$

$$\hat{y} = Gy \longrightarrow \hat{y} = \begin{bmatrix} 2.0 \\ 3.0 \\ 0.4 \end{bmatrix}$$

A seguir, resolveremos os dois sistemas triangulares superiores, obtendo a solução básica x_I e a coluna atualizada $\hat{A}^5$:

$$U^2 \hat{A}^5 = \hat{\upsilon} \longrightarrow \begin{bmatrix} 2.0 & -1.0 & 3.0 \\ & 4.2 & -1.0 \\ & & 2.4 \end{bmatrix} \quad \hat{A}^5 = \begin{bmatrix} 1.0 \\ 1.2 \\ -0.4 \end{bmatrix} \longrightarrow \hat{A}^5 = \begin{bmatrix} 0.9 \\ 0.3 \\ -0.2 \end{bmatrix}$$

$$U^2 x_I = \hat{y} \longrightarrow \begin{bmatrix} 2.0 & -1.0 & 3.0 \\ & 4.2 & -1.0 \\ & & 2.4 \end{bmatrix} x_I = \begin{bmatrix} 2.0 \\ 3.0 \\ 0.4 \end{bmatrix} \longrightarrow x_I = \begin{bmatrix} 1.2 \\ 0.8 \\ 0.2 \end{bmatrix} \begin{matrix} 1 \\ 3 \\ 4 \end{matrix}$$

k) Escolha da variável a sair da base

A variável x_1 é escolhida para sair da base e assume o limite inferior.

l) Vetor de recursos "corrigido"

Devemos adicionar o valor de x_5 e tirar o de x_1:

$$V = \begin{bmatrix} 2. \\ -1. \\ 4. \end{bmatrix} + \begin{bmatrix} 2. \\ 1. \\ -1. \end{bmatrix} \times 0. - \begin{bmatrix} 1. \\ 2. \\ -1. \end{bmatrix} \times 0. \longrightarrow V = \begin{bmatrix} 2. \\ -1. \\ 4. \end{bmatrix}$$

m) Atualização da decomposição LU

x_1 é a variável a sair da base $\longrightarrow$ INCSAI(2) = 2 e a matriz H^2 é igual a

$$H^2 = \begin{bmatrix} -1.0 & 3.0 & 1.0 \\ 4.2 & -1.0 & 1.2 \\ & 2.4 & -0.4 \end{bmatrix}$$

Para triangularizar a matriz H^2, fazemos uma permutação e um pivoteamento, com o elemento que multiplica o pivô igual a 0.24. Permutando:

$$H^2 = \begin{bmatrix} 4.2 & -1.0 & 1.2 \\ -1.0 & 3.0 & 1.0 \\ & 2.4 & -0.4 \end{bmatrix}$$

e fazendo o pivoteamento

$$H^2 = \begin{bmatrix} 4.2 & -1.0 & 1.2 \\ & 2.8 & 1.3 \\ & 2.4 & -0.4 \end{bmatrix}$$

Acrescentamos então mais duas posições em ATUALI, que são (1,0.24), correspondentes à permutação e pivoteamento efetuados.

Agora não necessitamos fazer uma permutação e o elemento que multiplica o pivô é igual a -0.87. Com estas operações, a matriz fica triangular superior novamente, isto é, temos a matriz U^3:

$$U^3 = \begin{bmatrix} 4.2 & -1.0 & 1.2 \\ & 2.8 & 1.3 \\ & & -1.5 \end{bmatrix}$$

Acrescentamos as duas posições correspondentes às operações efetuadas em ATUALI: (0,-0.87).

O vetor ATUALI está com os seguintes valores: ATUALI(I)=(1., 0.12,1.,0.24,0.,-0.87)

n) Obtenção dos novos multiplicadores

O processo é o mesmo realizado anteriormente e é repetido até que se obtenha a solução ótima para o problema.

C A P Í T U L O 2

RESOLUÇÃO DE PROBLEMAS LINEARES DINÂMICOS ATRAVÉS DO SIMPLEX COM DECOMPOSIÇÃO LU DA BASE

2.1. Introdução e formulação do problema

Seja o problema linear dinâmico:

$$
(P)\begin{cases}
x(t+1) = A(t)\,x(t) + B(t)\,u(t) & t = 0,\dots,T-1 \quad (1)\\
x(0) = x^{o} & (2)\\
C(t)\,x(t) + D(t)\,u(t) = f(t) & t = 0,\dots,T-1 \quad (3)\\
\alpha(t+1) \leq x(t+1) \leq \beta(t+1),\ \gamma(t) \leq u(t) \leq \delta(t) & \\
\min J(x,u) = \sum_{t=0}^{T-1} \left[c(t+1)x(t+1) + d(t)u(t) \right] & t = 0,\dots,T-1 \quad (4)\\
 & (5)
\end{cases}
$$

onde:

- T: horizonte de planejamento considerado fixo;
- x(t): vetor de estado (nx1), com limites dados por $\alpha(t)$ e $\beta(t)$;
- u(t): vetor de controle (rx1), com limites dados por $\gamma(t)$ e $\delta(t)$;
- f(t): vetor de recursos, conhecido (mx1, $m \leq r$)
- $x(0) = x^{o}$: vetor de condições iniciais, conhecido;

- A(t), B(t), C(t), D(t): matrizes conhecidas de dimensões (n x n), (n x r), (m x n) e (m x r) respectivamente;
- c(t+1), d(t): vetores conhecidos de dimensões (1xn) e (1xr) respectivamente.

O problema (P) pode ser assim enunciado: encontre uma seqüência de vetores de controle {u(0),...,u(T-1)} satisfazendo conjuntamente (1),(2),(3) e (4) e tais que minimizem (5).

A matriz de restrições A, referente ao problema (P) é indicada na figura a seguir, assim como o vetor de custos Z.

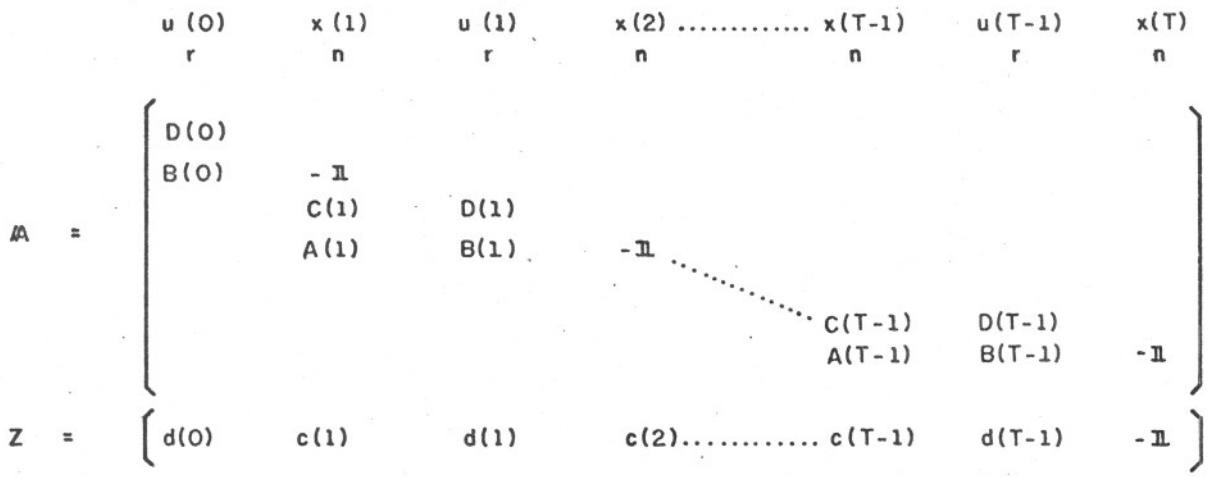


Figura 2.1.

Este problema caracteriza-se pela forma da matriz de restrições A, a qual, além de ser esparsa, tem uma estrutura particular conhecida como estrutura escada (staircase) sendo que o objetivo deste trabalho é implementar um algoritmo que aproveite esta estrutura. O método a ser utilizado é o Simplex Revisado, cujos passos foram enunciados no capítulo anterior.

O trabalho computacional concentra-se, a cada iteração, na resolução dos sistemas seguintes:

$$B \hat{A}^j = A^j$$

$$B^T \pi = (Z^I)^T$$

$$B x_I = V$$

Na resolução destes sistemas, é usada a decomposição L U da base B, a qual, para o problema (P) definido, herda a estrutura escada da matriz A, conforme mostra a figura abaixo:

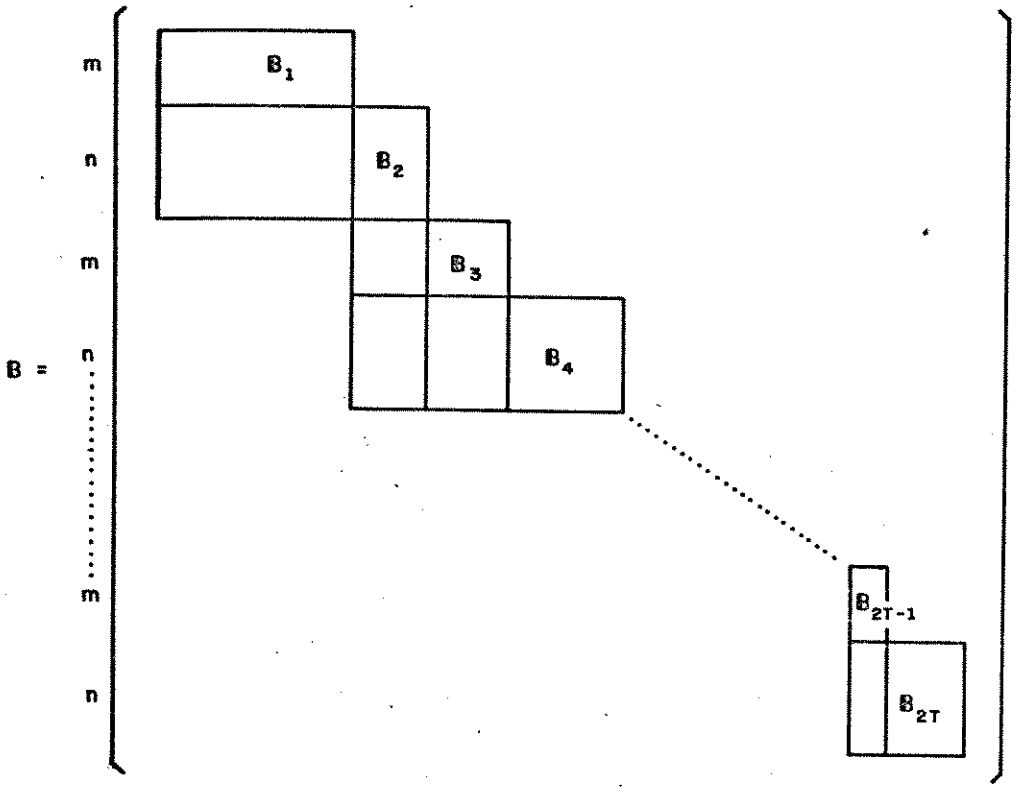


Figura 2.2

Esta matriz é esparsa, tendo todos os elementos não nulos concentrados na escada. Para tirar proveito desta estrutura especial, nossa preocupação passará a ser:

1. fatorar a matriz B de modo que os fatores L e U mantenham a mesma estrutura;
2. modificar a fatoração da base a cada troca de colunas, de modo a continuar mantendo a estrutura.

2.2. Fatoração da base

Para fatorar a base B, vamos zerar os elementos que estão abaixo da diagonal, coluna por coluna. Isto equivale a pré-multiplicar a matriz B por uma sucessão de matrizes elementares do tipo:

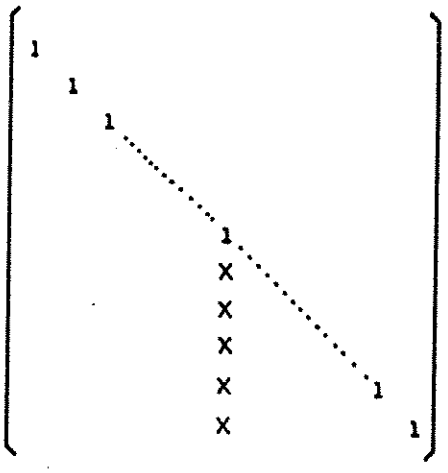


Figura 2.3

O produto de todas essas matrizes constitui a matriz L^{-1} que verifica $L^{-1} B = U$. O caso ideal seria obter uma matriz U com a seguinte estrutura:

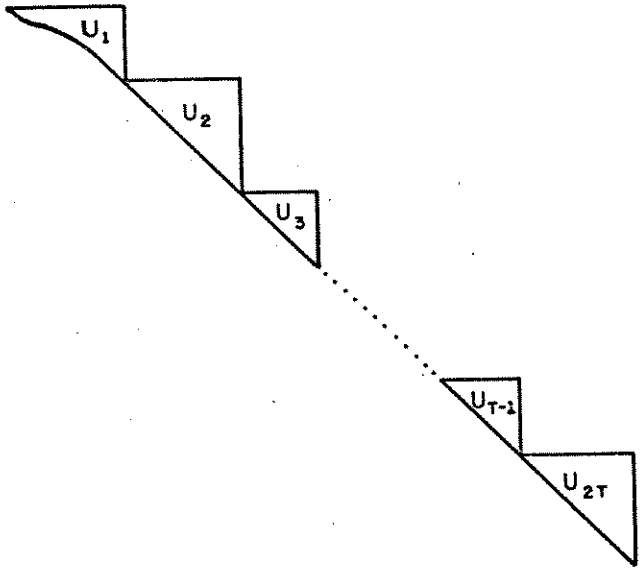


Figura 2.4

A fatoração implementada neste algoritmo busca um compromisso entre a estabilidade e a preservação da estrutura. Se pensássemos somente em estabilidade, escolheríamos como pivô o elemento de maior valor absoluto da coluna testada. A figura abaixo ilustra as conseqüências desta escolha sobre a estrutura:

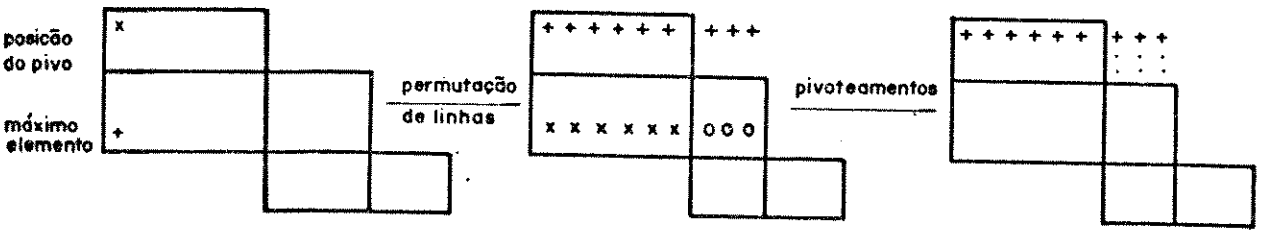


Figura 2.5

Os pontos (.) indicam a criação de elementos não nulos, mudando a estrutura da matriz. Como não desejamos que isto aconteça, limitaremos o conjunto de linhas entre as quais vamos escolher o pivô.

Vamos agora explicar em detalhes o processo: conforme a configuração da base B (figura 2.2), onde assinalamos os blocos B_1, B_2,..., B_{2T} para facilitar a exposição, temos que o bloco B_1 tem m linhas e como B é uma base, tem m colunas linearmente independentes.

Começando o processo de fatoração, escolhemos como pivô, na primeira coluna, o elemento de maior valor absoluto entre as m primeiras linhas. Fazemos a permutação, se necessário. Observa-se que assim não é alterada a estrutura. Passamos a zerar todos os elementos da coluna, a partir do pivô. A coluna que caracteriza a matriz elementar é representada abaixo:

$$\begin{pmatrix} 1 \\ -\dfrac{b_{12}}{b_{11}} \\ -\dfrac{b_{13}}{b_{11}} \\ \vdots \end{pmatrix}$$

Figura 2.6

onde b_{11} é o pivô.

Observe-se também que não é necessário armazenar toda a matriz, bastando apenas guardar a coluna acima.

Para fixar as idéias, vamos chamar P_1 à permutação e L_1 às operações de pivoteamento realizadas neste passo. O pivô b_{11} constitui a primeira coluna de U_1 (pequena matriz triangular superior, fazendo parte da matriz U, conforme mostra a figura 2.4), a qual, por ser a primeira coluna de uma matriz triangular superior, contém apenas o elemento da diagonal.

Tomamos agora a segunda coluna de $\mathcal{B}$. Antes de mais nada, temos que aplicar P_1 e L_1 sobre ela. O pivô vai ser escolhido entre os elementos das linhas 2 a m. Se estes elementos forem todos nulos, isto significa que a segunda coluna de $\mathcal{B}_1$ é linearmente dependente da anterior. Neste caso, esta coluna é abandonada e passamos a testar a coluna seguinte. Uma vez escolhido o pivô, faz-se a permutação P_2 e as operações necessárias para zerar os elementos a partir da linha 3, as quais irão constituir L_2. Os elementos da coluna, desde a primeira linha até a linha do pivô constituem a nova coluna de U_1. Repetimos o processo até estabelecermos um pivô a cada uma das m primeiras linhas. Isto é garantido pelo fato da matriz $\mathcal{B}$ ser não singular.

Neste instante, teremos efetuado as seguintes operações:

$L_{m-1}\ P_{m-1}\ L_{m-2}\ P_{m-2} \cdots\cdots\ L_2\ P_2\ L_1\ P_1$

estando na seguinte situação:

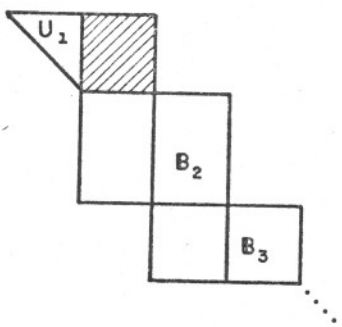


Figura 2.7

A área hachurada representa o conjunto das colunas de $\mathcal{B}_1$ que foram abandonadas no processo anterior, as quais chamamos de colunas sobrantes e que constituem a matriz Φ_1.

A existência destas colunas sobrantes impede que a matriz U tenha a configuração ideal da figura 2.4, fazendo com que apareçam elementos não nulos acima das U_i, i=2,.... 2T.

Continuando o processo de triangularização, vamos criar U_2. Consideremos a matriz cujas colunas são as colunas sobrantes e as colunas de $\mathcal{B}_2$, e cujas linhas são as n linhas a partir da linha do último pivô estabelecido. Como no caso anterior, pelo fato de $\mathcal{B}$ ser uma base, é garantida a obten

ção de n pivôs, os quais serão escolhidos entre estas n linhas.

Na implementação que fizemos, primeiro são testadas as colunas sobrantes e depois as colunas de B_2. Repete-se este processo, até que seja obtida a fatoração completa da matriz B. É importante notar que as colunas que sobraram na formação de cada U_i, podem fazer parte de qualquer U_j, j>i.

A configuração final é a seguinte:

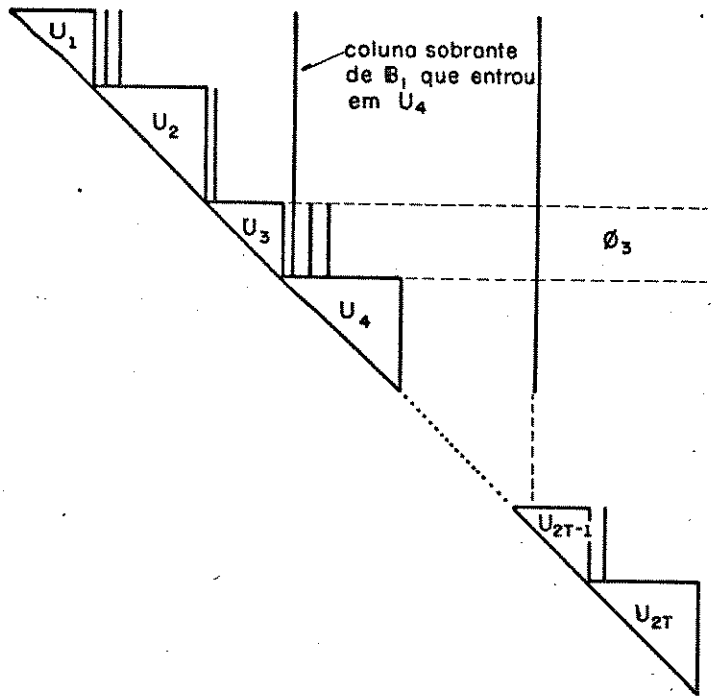


Figura 2.8

Observar que as variáveis de controle do período t, $1 \leqslant t \leqslant T$, estão associadas à matriz U_{2t-1} e as variáveis de estado estão associadas à matriz U_{2t}. Deve-se notar que, na verdade, as colunas de U_{2t-1} podem estar representando variáveis de controle ou de estado de períodos anteriores.

Chamamos Φ_i à matriz cujas colunas são as colunas sobrantes e cujas linhas são as linhas de U_i, $1 \leqslant i < 2T$.

Nesta altura, aplicamos $\{[(m-1)+(n-1)]\,T\}$ permutações e a mesma quantidade de operações de pivoteamento sobre B, ou seja:

$$L_{[(m-1)+(n-1)]\,T}\, P_{[(m-1)+(n-1)]\,T} \cdots\cdots L_1 P_1 B = U \longrightarrow L^{-1} B = U$$

isto é, o produto de todas as operações de permutação e pivoteamento constitui a L^{-1}. Lembramos que a matriz L^{-1} é armazenada na forma produto (esparsa).

2.3. Resolução dos sistemas

2.3.1. Multiplicadores

$$B^T\pi = (Z^I)^T$$

$$(LU)^T\pi = (Z^I)^T$$

$$U^T L^T\pi = (Z^I)^T \longrightarrow \begin{cases} U^T y = (Z^I)^T \\ L^T \pi = y \longrightarrow \pi = (L^{-1})^T y \end{cases}$$

Lembramos que $L^{-1} = L_{[(m-1)+(n-1)]\,T}\, P_{[(m-1)+(n-1)]\,T} \cdots\cdots L_1 P_1$

então $(L^{-1})^T = P_1 L_1^T P_2 L_2^T \cdots\cdots P_{[(m-1)+(n-1)]\,T}\, L^T_{[(m-1)+(n-1)]\,T}$

já que $P_i^T = P_i$ pois P_i é simétrica, $i=1,\ldots,[(m-1)+(n-1)]\,T$.

Para obter os multiplicadores, a seqüência é a seguinte:

→ Resolver $U^T y = (Z^I)^T$;

U^T é uma matriz triangular inferior, com a seguinte característica:

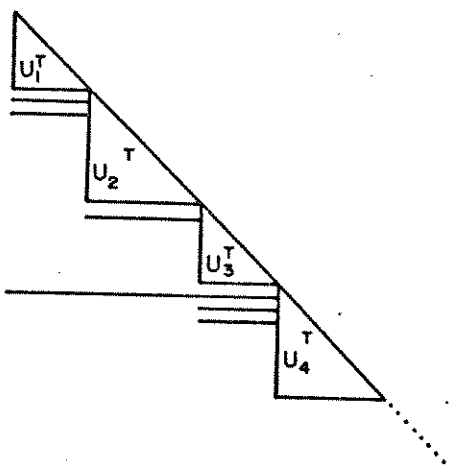


Figura 2.9

→ Aplicar $(L^{-1})^T$ sobre y, obtendo π

2.3.2. Solução básica

$$Bx_I = V$$

$$LU\, x_I = V$$

$$Ux_I = L^{-1}V = \tilde{V}$$

A seqüência de operações para obter a solução básica é, então:

→ Aplicar L^{-1} ao vetor V, obtendo $\tilde{V}$;

→ Resolver o sistema triangular superior $U x_I = \tilde{V}$, obtendo-se a solução básica.

2.3.3. Atualização da coluna a entrar na base

$$B\bar{A}^j = A^j$$
$$L\,U\,\bar{A}^j = A^j$$
$$U\bar{A}^j = L^{-1}A^j = \bar{A}^j$$

A seqüência de operações é, então, a seguinte:

→ Aplicar L^{-1} sobre a coluna A^j, obtendo $\bar{A}^j$;

→ Resolver o sistema triangular superior $U\bar{A}^j = \bar{A}^j$, obtendo a coluna atualizada em relação à base B.

2.4. Considerações sobre a posição da variável a entrar na base, relativamente à posição da variável a sair.

Suponhamos que a variável a sair esteja representada por uma matriz U_i e a variável a entrar, associada a U_j, com $j > i$

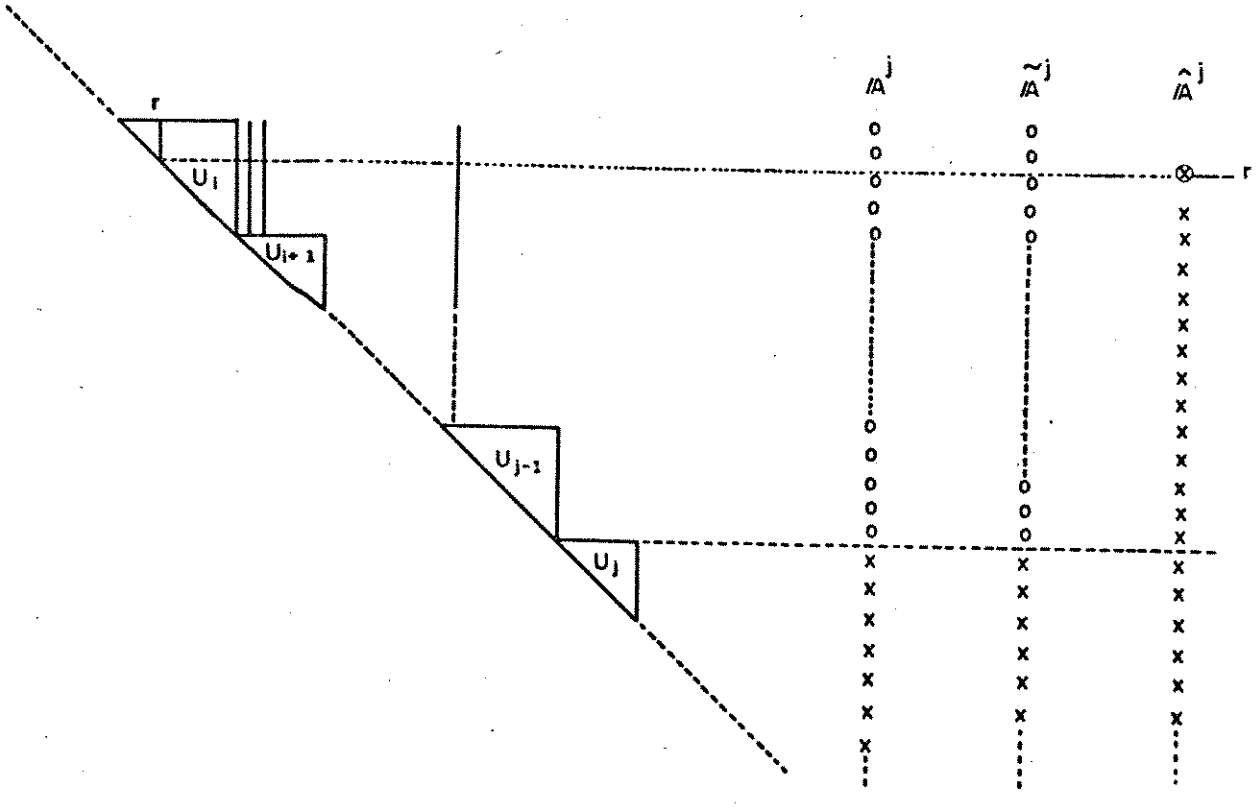


Figura 2.10

Seja r a posição da variável a sair, relativamente à U_i. Como mostra a figura, a variável A^j a entrar tem

zeros até a linha correspondente à primeira linha de U_j. Além disso, $\hat{A}^j = L^{-1} A^j$ mantém todos esses zeros.

Para que A^j possa entrar no lugar da variável a sair, é necessário e suficiente que $\hat{A}^j$ tenha um elemento não nulo na linha correspondente à linha r de U_i. Então, podemos afirmar que, necessariamente:

$\longrightarrow \Phi_i \neq 0$

e mais ainda:

$\longrightarrow$ a linha r de $U_i^{-1} \Phi_i$ tem algum elemento não nulo.

Demonstração:

Supondo U_i com m linhas, vamos chamar

$$\begin{bmatrix} \hat{a}_{il} \\ \vdots \\ \hat{a}_{ir} \\ \vdots \\ \hat{a}_{im} \end{bmatrix}$$

os elementos de $\hat{A}^j$ correspondentes às linhas de U_i.

Temos o sistema $U\hat{A}^j = \hat{A}^j = 0$

$$U_i \begin{bmatrix} \hat{a}_{il} \\ \vdots \\ \hat{a}_{ir} \\ \vdots \\ \hat{a}_{im} \end{bmatrix} + \Phi_i \hat{a}_k = 0$$

onde $\hat{a}_k$ consiste dos elementos de $\hat{A}^j$ nas linhas abaixo da última linha de U_i e só das linhas associadas às colunas de Φ_i.

Da equação acima temos que

$$\begin{bmatrix} \hat{a}_{il} \\ \vdots \\ \hat{a}_{ir} \\ \vdots \\ \hat{a}_{im} \end{bmatrix} = - U_i^{-1} \Phi_i \hat{a}_k$$

Vemos então que se a linha r de $U_i^{-1} \Phi_i$ for nula, é óbvio que $\hat{a}_{ir} = 0$

Vamos interpretar este resultado de uma forma

que será útil depois: temos que as colunas de U_i constituem uma base; se tirarmos a coluna r de U_i e introduzirmos em seu lugar uma coluna de Φ_i, por exemplo a coluna Φ_i^k, teremos novamente uma base somente se o elemento r de $U_i^{-1} \Phi_i^k$ for não nulo, isto é, se a coluna Φ_i^k for linearmente independente das colunas de U_i que restam, quando retiramos a coluna r.

2.5. Atualização da decomposição LU

Temos dois casos a estudar:

Caso 1: a variável a sair está colocada em U_i e a variável a entrar está associada a U_j, $j > i$.

Este é o caso representado na figura 2.10, a qual mostra que não podemos fazer a troca diretamente, pois a coluna $\bar{A}^j$ tem um zero na linha r (supondo novamente que r é a posição relativa da variável a sair em U_i), indicando dependência linear com as colunas de U_i. Devemos então caminhar com a coluna correspondente à variável a sair através das U_k, $k > i$, até conseguirmos fazer a troca.

Supondo U_i com m colunas, os passos do processo são os seguintes:

1. colocamos a coluna r na posição da última coluna de U_i, trazendo as outras colunas, a partir da (r+1)-ésima, uma posição para trás. A matriz U fica com a seguinte forma:

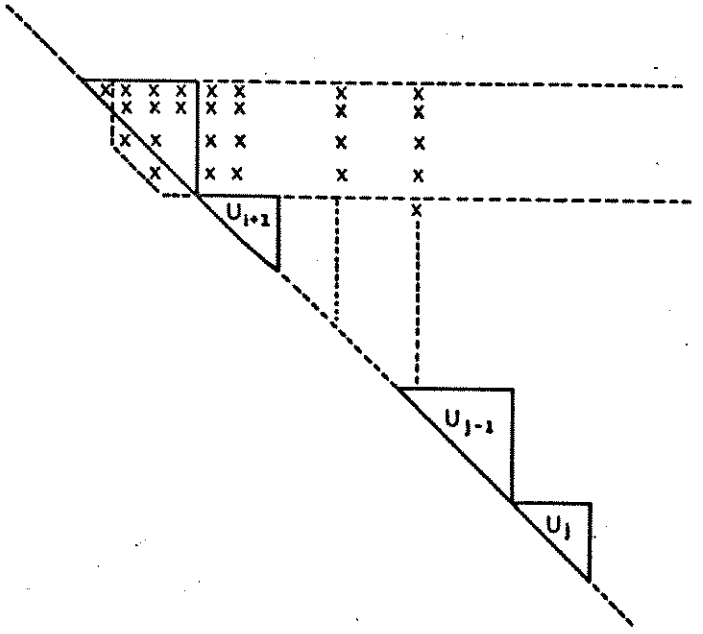


Figura 2.11

Esta permutação faz com que U_i perca a triangularidade. É preciso então, zerar os elementos da subdiagonal, a partir da coluna r. Estas operações são feitas com a escolha do maior elemento, em valor absoluto, como pivô, conforme o trabalho de decomposição LU descrito no capítulo 1.

Estas operações são armazenadas incrementando o arquivo das L's e, uma vez aplicadas, fazem com que a matriz fique com a seguinte forma:

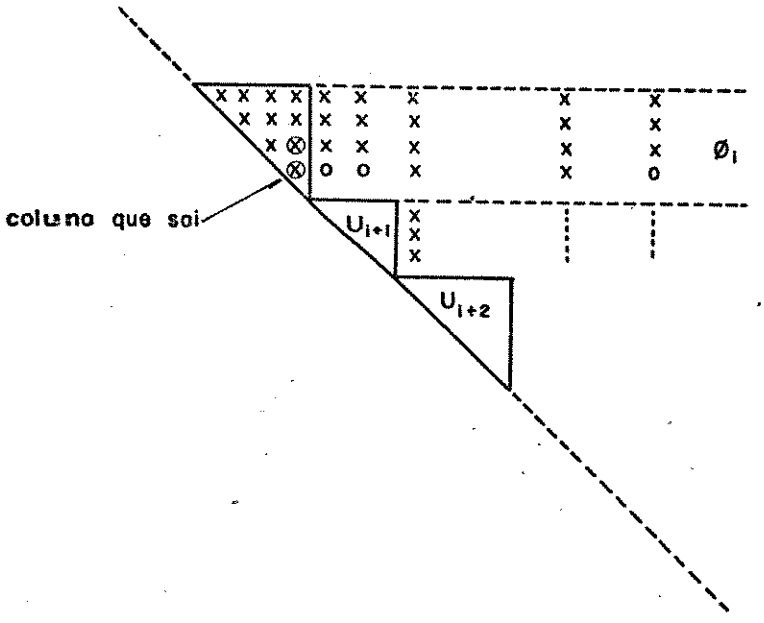


Figura 2.12

Note-se que somente as linhas correspondentes às linhas de U_i da matriz global foram modificadas. Os elementos marcados ⊗ foram criados a partir destas operações. As colunas da matriz Φ_i que apresentam um zero na última linha são aquelas linearmente dependentes das colunas de U_i, uma vez retirada a coluna que sai, isto é, elas não podem ser candidatas a substituir a coluna que sai da base. Todas essas operações devem ser feitas sobre A^j. Deve-se notar também que o elemento pivô passou a ocupar a ^{i}posição m, na diagonal de U_i.

2. procuramos a primeira coluna de Φ_i que tenha um elemento não nulo na linha m e permutamos esta coluna com a coluna que sai. Ficamos com a seguinte situação:

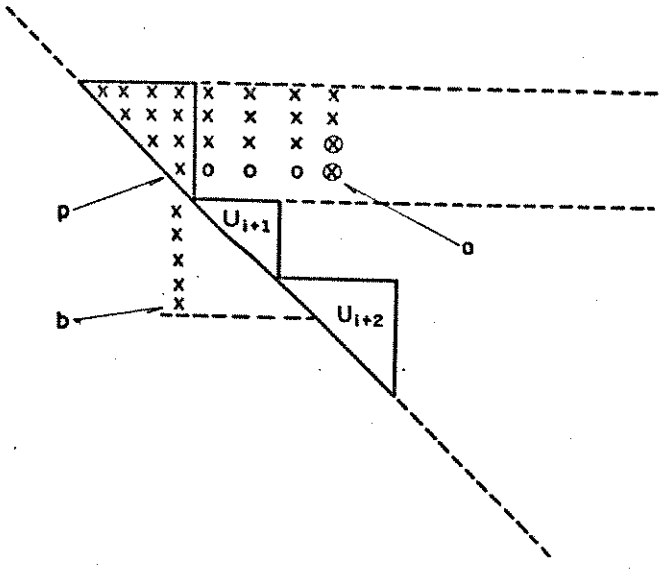


Figura 2.13

Esta permutação implica em que o elemento pivô de $\hat{A}^j$ está na linha correspondente à coluna de Φ_i escolhida.

3. passamos a zerar os elementos abaixo da diagonal, pois com a permutação, a matriz novamente perdeu a triangularidade.

Os zeros presentes em todas as colunas de Φ_i anteriores à coluna escolhida, garantem que não é alterada nenhuma coluna entre as duas permutadas. Isto garante a reobtenção da triangularidade da matriz. Além disso, só sofrerão alterações as colunas posteriores à coluna escolhida que tiverem um elemento não nulo na m-ésima linha de Φ_i.

Depois de zerar os elementos, a situação é a seguinte:

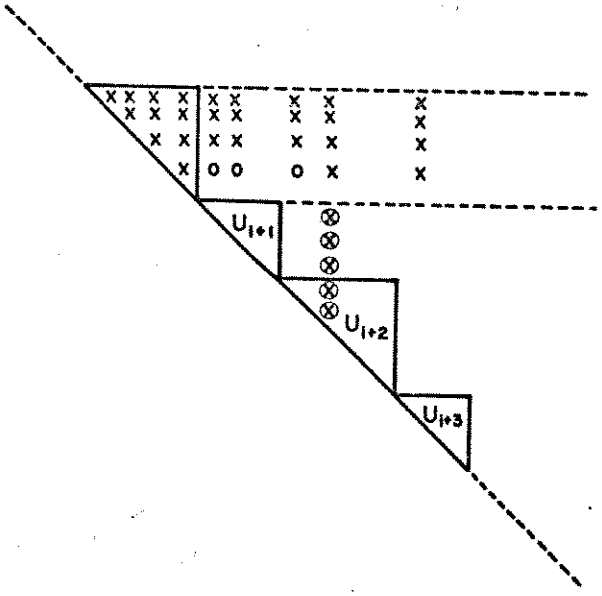


Figura 2.14

Todos os elementos ⊗ foram criados com as operações efetuadas e garante-se que o elemento ⊗ que está na diagonal é não nulo (pois é igual a $(\frac{-b}{p} \cdot a)$, com b, p e a não nulos, indicados na figura 2.13). Estas operações realizadas também são acrescentadas no arquivo das L's.

Observemos que a variável a sair foi transladada para uma matriz U_k, $k > i$. Se k ainda é menor que j, as operações do tipo L não afetam $\bar{A}^j$.

Este processo vai ser repetido com as mesmas características, até que a variável a sair chegue na última posição de U_j. Quando é efetuada a troca com uma coluna de Φ_j, as operações de pivoteamento para zerar os elementos abaixo do triângulo, devem ser feitas, a partir de agora, também sobre $\bar{A}^j$.

Neste instante, caímos no caso 2.

Caso 2: a variável a sair está colocada em uma U_i e a variável a entrar está associada a uma U_j, com $j \leq i$.

Os passos, nesta situação, são os seguintes: levamos a coluna a sair até a posição da última coluna de U_i e trazemos as outras uma posição para trás. O processo é idêntico ao do caso 1, sendo que aqui as L's devem ser aplicadas também em $\bar{A}^j$.

São possíveis duas situações: a primeira delas é ilustrada na figura abaixo:

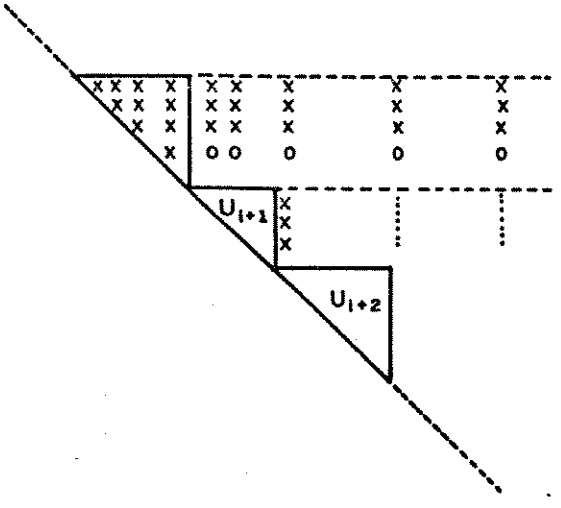


Figura 2.15

Neste caso, a matriz Φ_i tem a última linha nula. Retiramos então, a coluna a sair e colocamos a coluna correspondente a $\bar{A}^j$ atualizada, nesta posição. Fica claro que no processo de zerar os elementos de $\bar{A}^j$ que estão abaixo da diagonal, nenhuma coluna posterior da base global será alterada.

A segunda situação é aquela onde existe algum elemento não nulo na última linha da matriz Φ_i, conforme mostra a figura abaixo:

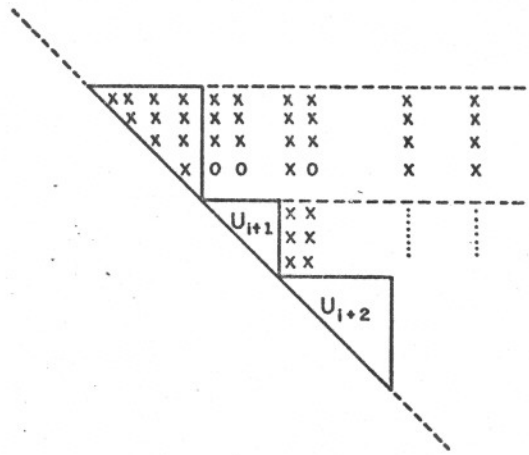


Figura 2.16

Escolhemos a primeira coluna de Φ_i com elemento não nulo na última linha e permutamos a variável a sair com ela. O processo é novamente análogo ao caso anterior e é repetido até cairmos ou na situação anterior ou na última coluna de U_{2T}, quando fazemos a troca.

UNICAMP
BIBLIOTECA CENTRAL

C A P Í T U L O 3

TÉCNICAS UTILIZADAS PARA O ARMAZENAMENTO ESPARSO

3.1. Introdução

Considerando o programa desenvolvido para o problema de grande porte esparso, com estrutura escada, objeto de nosso estudo, não faria sentido um armazenamento de toda a matriz, com todos os zeros, isto é, a esparsidade deve ser explorada, fazendo com que apenas os elementos não nulos sejam armazenados e processados. Então, não faremos uso dos arranjos bidimensionais, desde que não serão necessárias todas posições da matriz. Isto leva ao uso de vetores que não são visualmente idênticos à matriz correspondente, conhecidos como a forma compacta de armazenamento. É então necessário localizar cada um destes elementos não nulos durante o processo de fatoração e identificá-lo como um elemento particular da matriz correspondente, o que requer informação indexada, além do valor numérico do elemento, para que sua posição original possa ser estabelecida. Além disto, elementos não nulos, em posições anteriormente nulas podem continuamente ser gerados durante a fatoração, além de que elementos originalmente não nulos podem vir a anular-se. Isto exige que o esquema de compactação seja capaz de registrar eficientemente estas mudanças durante a fatoração, incorporando os novos elementos não nulos criados e indicando os novos espaços livres disponíveis.

Este armazenamento faz com que o tempo de execução aumente, simplesmente para identificar cada elemento, di

ficultando enormemente o processo de programação, antes relativamente simples. É claro que os problemas de programação com os dados na forma compacta estão associados não somente aos aspectos básicos da fatoração e solução numérica, como também com o problema de armazenamento e identificação dos elementos não nulos.

3.2. Armazenamento dos dados de entrada

Vamos agora detalhar como todos os dados de entrada foram armazenados na forma compacta, no programa que criamos para o algoritmo de Programação Linear Dinâmica desenvolvido no capítulo 2.

3.2.1. Matriz de restrições

Os elementos não nulos da matriz A são representados pelo vetor VALORA, com os indicadores LINHAA e ICOLA. O vetor VALORA (I) contém o valor numérico do elemento I, não nulo, da matriz A, a qual é armazenada por colunas; o apontador LINHAA (I) indica a linha da matriz A à qual corresponde o elemento armazenado em VALORA (I) e o apontador ICOLA (L), com $[(m+n)\ T + 1]$ posições, indica a posição de VALORA que contém o primeiro elemento da coluna L, sendo que a posição $[(m+n)T+1]$ de ICOLA indica a primeira posição livre em VALORA. Desta maneira, apenas os elementos não nulos de A são armazenados e cada um deles é identificado através dos três arranjos criados.

Vejamos, a título de ilustração, como ficaria a matriz A abaixo, armazenada na forma compacta:

$$A = \begin{bmatrix} 3. & 2. & & & \\ 2. & 1. & 2. & & \\ & & 1. & 4. & \\ & & & 2. & 3. \\ & & & 1. & 4. \end{bmatrix}$$

Neste caso, teríamos:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13
VALORA	3.	2.	2.	1.	2.	1.	4.	2.	1.	3.	4.		
LINHAA	1	2	1	2	2	3	3	4	5	4	5		
ICOLA	1	3	5	7	10	12							

Se quisermos reconstruir a coluna 3, por exemplo, procuramos ICOLA (3) = 5 e $[$ICOLA (4) - 1$]$ = 6, o que sig

nifica que a coluna começa na posição 5 de VALORA e termina na posição 6, isto é, contêm 2 elementos: o elemento de valor 2., na linha 2 (VALORA(5) e LINHAA(5)) e o elemento de valor 1. na linha 3(VALORA(6) e LINHAA(6)).

3.2.2. O vetor de custos

Os elementos não nulos do vetor Z são representados pelo vetor VALORC(I), que contêm o valor numérico do elemento I, juntamente com o indicador ICOLC(I), indicando a posição original no vetor Z, do elemento I armazenado em VALORC.

Como ilustração, seja Z o vetor representado abaixo:

Z = (1.,5.,-3.,2.,0.,0.,3.,4.,0.,0.,0.,1.,0.,0.,0.)

Este vetor, na forma compacta, é representado por:

POSIÇÃO	1	2	3	4	5	6	7
VALORC	1.	5.	-3.	2.	3.	4.	1.
ICOLC	1	2	3	4	7	8	12

3.2.3. O vetor de recursos

No programa implementado, o vetor de recursos é chamado de V e é representado na forma compacta por VALORV e ICOLV, com funções idênticas a VALORC e ICOLC que representam o vetor de custos.

3.3. Armazenamento da matriz triangular inferior

A matriz L^{-1}, gerada na decomposição da base B (L^{-1} B = U), é armazenada na forma produto e é representada na forma compacta pelo vetor VALORL, com os indicadores LINHAL e ICOLL, com funções idênticas às de VALORA, LINHAA e ICOLA para a matriz A, sendo que os elementos da diagonal não são armazenados, pois são todos iguais a 1. O apontador ICOLL tem um contador, KICOL, e o vetor VALORL tem um contador, KL, que apontam sempre para a primeira posição livre dos respectivos arranjos, a fim de que, quando uma nova coluna de L^{-1} deva ser gerada, ela seja armazenada a partir destas posições. Como exemplo, seja a matriz L^{-1} indicada a seguir:

$$L^{-1} = (L^{-1})^3\ (L^{-1})^2\ (L^{-1})^1 = \begin{bmatrix} 1. & & & \\ & 1. & & \\ & & 1. & \\ & & 0.1 & 1. \end{bmatrix} \begin{bmatrix} 1. & & & \\ & 1. & & \\ & 0.2 & 1. & \\ & 0.3 & & 1. \end{bmatrix} \begin{bmatrix} 1. & & & \\ 0. & 1. & & \\ 0.5 & & 1. & \\ 0.1 & & & 1. \end{bmatrix}$$

Na forma compacta, a matriz L é armazenada como se segue:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11.....
VALORL	0.5	0.1	0.2	0.3	0.1						
LINHAL	3	4	3	4	4						
ICOLL	1	3	5	6							

3.4. Armazenamento da matriz triangular superior

Conforme consta no capítulo 2, a matriz U tem a seguinte configuração:

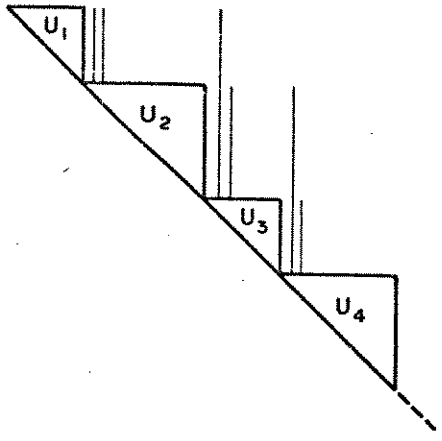


Figura 3.1.

isto é, é constituída de pequenas matrizes triangulares superiores, U_1, U_2, ...U_{2T}, com dimensões m,n,...m,n, respectivamente e de algumas colunas isoladas, constituindo as colunas sobrantes.

A forma de armazenamento esparso para a matriz U, escolhida como a melhor dentre várias testadas, é a seguinte: armazenamos independentemente as partes correspondentes às pequenas matrizes triangulares superiores U_i, i = 1,.... 2T e as partes correspondentes às colunas sobrantes. Temos então, dois conjuntos de valores a armazenar:

1. As matrizes U_i, i = 1,... 2T

As matrizes U_i, i = 1,.... 2T, foram armazenadas por colunas, incluindo os zeros, num vetor denominado U, com $\lceil m(m+1)/2+n(n+1)/2 \rceil T$ posições, sendo que qualquer coluna de U_i pode ser reproduzida a partir do vetor U.

Para ilustrar, suponhamos uma matriz U com os seguintes elementos:

```
1.  2.  x           x
    3.  x           x
        2.  1.  2.  x
            3.  0.  x
                1.  x
                    1.  0.  x
                        2.  x
                            1.  0.  3.
                                1.  1.
                                    2.
```

Esta matriz é composta de quatro pequenas matrizes triangulares, sendo U_1 e U_3 com m colunas (m=2) e U_2 e U_4 com n colunas (n=3). O vetor U será então igual a:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
U	1.	2.	3.	2.	1.	3.	2.	0.	1.	1.	0.	2.	1.	0.	1.	3.	1.	2.

Vamos reproduzir, a partir do vetor U, a segunda coluna de U_4 , como exemplo. Sendo a segunda coluna de uma matriz triangular, sabemos de imediato que ela tem duas posições. A primeira posição da coluna é calculada através da soma do número de elementos das U_is anteriores a U_4, com o número de elementos das colunas de U_4 anteriores àquela desejada, mais 1. Então:

1ª. posição da coluna 2 de U_4 =

$= \lceil m(m+1)/2 \rceil \times 2 + \lceil n(n+1)/2 \rceil \times 1 + 1 \times 2/2 + 1 = 14$ isto é, o primeiro elemento da coluna desejada está na posição 14 do vetor U. Os elementos da coluna 2 de U_4 são, então, iguais a 0.(posição 14) e 1.(posição 15 do vetor U).

2. as colunas sobrantes

Precisamos de vários arranjos para reproduzir, com exatidão, as colunas sobrantes.

O primeiro deles, USUJ, contém os valores de

cada elemento, inclusive os zeros, armazenados por coluna, a partir da primeira linha da U_i à qual a coluna sobrante corresponde. Acompanhando o vetor USUJ, temos o indicador IPROS, que indica qual é a posição de USUJ que contém o elemento seguinte àquele que estamos apontando. O último elemento da coluna sobrante é apontado por um zero em IPROS (isto é, não existe um elemento seguinte a ele). A primeira posição livre de USUJ é indicada por LIVRES e sempre que alguma coluna sobrante for criada, será armazenada em USUJ, a partir da posição indicada por LIVRES.

Os outros apontadores que identificam a coluna sobrante são os seguintes:

ICOSUJ: indica a coluna de U onde está a coluna sobrante;
ICONSU: indica a posição em USUJ onde está o primeiro elemento da coluna sobrante;
LINHAS: indica a linha de U em que começa a colúna sobrante;
IPROC : indica qual é a próxima coluna sobrante;

A primeira posição livre de ICOSUJ (correspondentemente de ICONSU e LINHAS) é indicada por LIVREC e se for criada uma nova coluna sobrante durante o processo, ela será anotada na posição LIVREC e os valores de IPROC sofrerão as alterações correspondentes.

A título de ilustração, vejamos como seriam armazenadas as colunas sobrantes de uma pequena matriz U, indicada a seguir:

```
x x 2.        1.
  x 1.        2.
    x x x 3.  3.
      x x 0.  2.
        x 2.  0.
          x x 1.1.
            x 0.3.
              x x x
                x x
                  x
                    x x 1.
                      x 2.
                        x x x
                          x x
                            x
```

Temos cinco colunas sobrantes. Os arranjos correspondentes seriam os seguintes:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21...
USUJ	2.	1.	3.	0.	2.	1.	2.	3.	2.	0.	1.	0.	1.	3.	1.	2.					
IPROS	2	0	4	5	0	7	8	9	10	11	12	0	14	0	16	0	18	19	20	21	22...

LIVRES = 17

POSIÇÃO	1	2	3	4	5	6	7.....
ICOSUJ	3	6	8	9	13		
ICONSU	1	3	6	13	15		
LINHAS	1	3	1	6	11		
IPROC	2	3	4	5	0	7	8.....

LIVREC = 6

O vetor IPROC é inicializado definindo-se IPRIMC = 1, sendo que este valor pode ser alterado durante as trocas de colunas. IPROC(5) = 0 indica que a coluna localizada na posição 5 de ICOSUJ é a última coluna sobrante.

Suponhamos agora que, durante a troca de colunas, a terceira coluna sobrante seja eliminada. Os arranjos sofrem as alterações correspondentes e passam a ter as seguintes configurações:

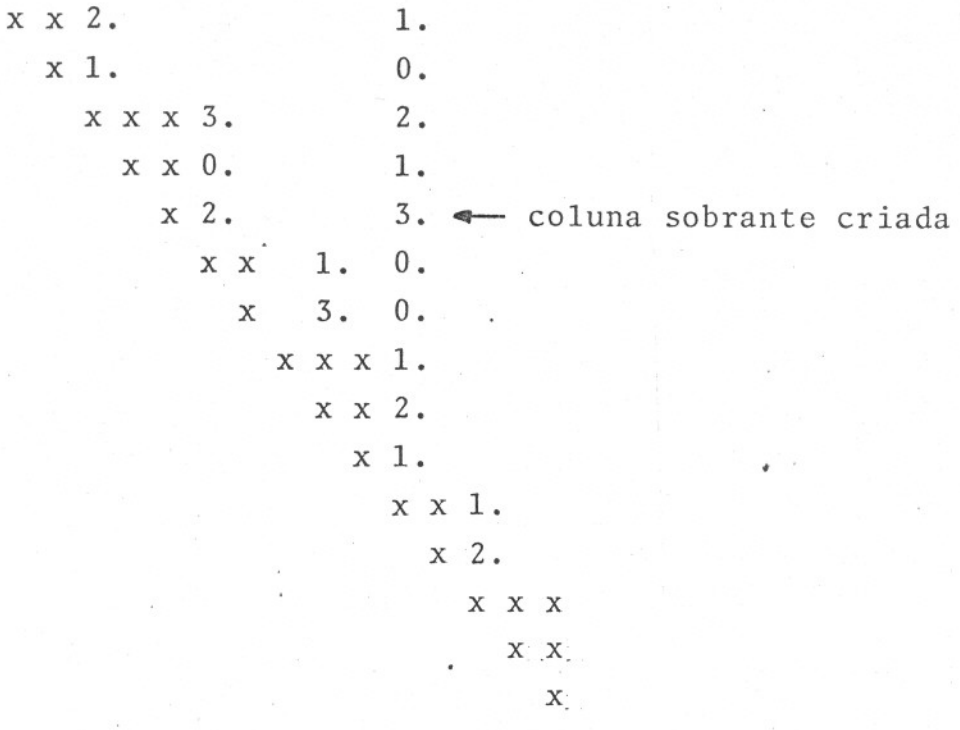
POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20.....
USUJ	2.	1.	3.	0.	2.								1.	3.	1.	2.				
IPROS	2	0	4	5	0	7	8	9	10	11	12	17	14	0	16	0	18	19	20	21.....

LIVRES = 6

POSIÇÃO	1	2	3	4	5	6	7	8.....
ICOSUJ	3	6		9	13			
ICONSU	1	3		13	15			
LINHAS	1	3		6	11			
OPROC	2	4	6	5	0	7	8	9.....

LIVREC = 3

Quando é criada uma nova coluna, os arranjos também sofrem as correspondentes alterações. Suponhamos uma nova coluna de U criada no processo, de maneira a que a nova configuração seja a seguinte:



Os arranjos devem conter as informações da nova coluna criada, partindo de LIVRES em USUJ e LIVREC em ICOSUJ, ICONSU e LINHAS:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21.....
USUJ	2.	1.	3.	0.	2.	1.	0.	2.	1.	3.	0.	0.	1.	3.	1.	2.	1.	2.	1.		
IPROS	2	0	4	5	0	7	8	9	10	11	12	17	14	0	16	0	18	19	0	21	22.....

LIVRES = 20

POSIÇÃO	1	2	3	4	5	6	7	8.....
ICOSUJ	3	6	11	9	13			
ICONSU	1	3	6	13	15			
LINHAS	1	3	1	6	11			
IPROC	2	4	5	3	0	7	8	9.....

LIVREC = 6

CAPÍTULO 4

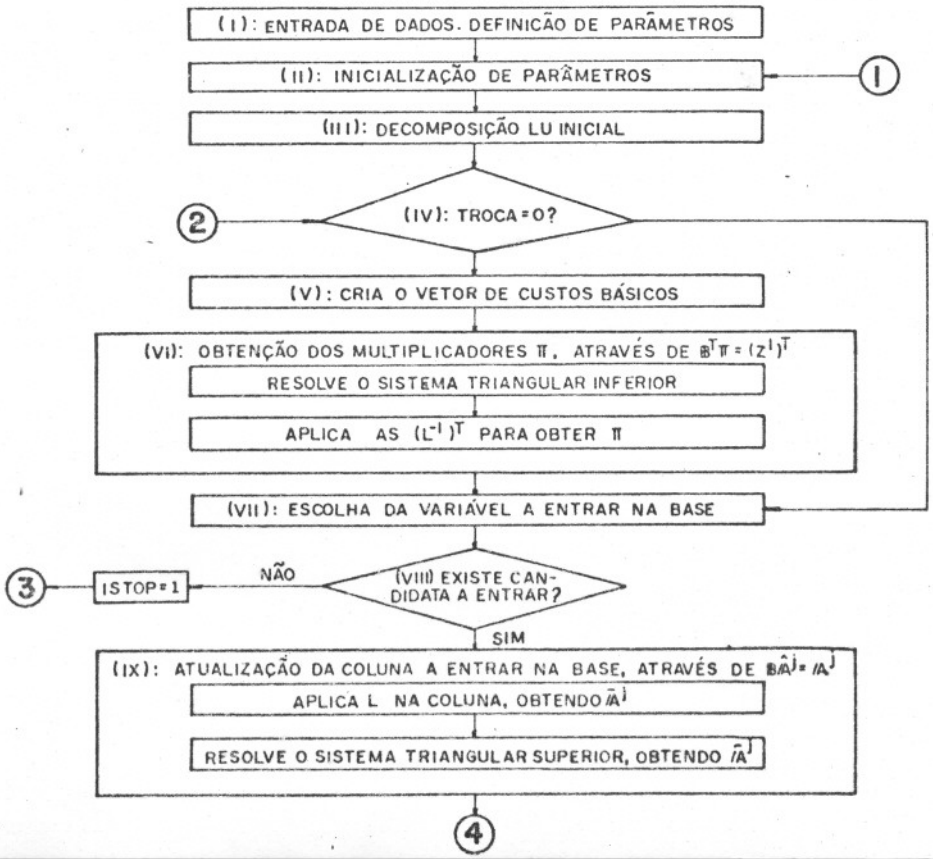
PROGRAMA COMPUTACIONAL PARA O ALGORITMO DE PROGRAMAÇÃO LINEAR DINÂMICA

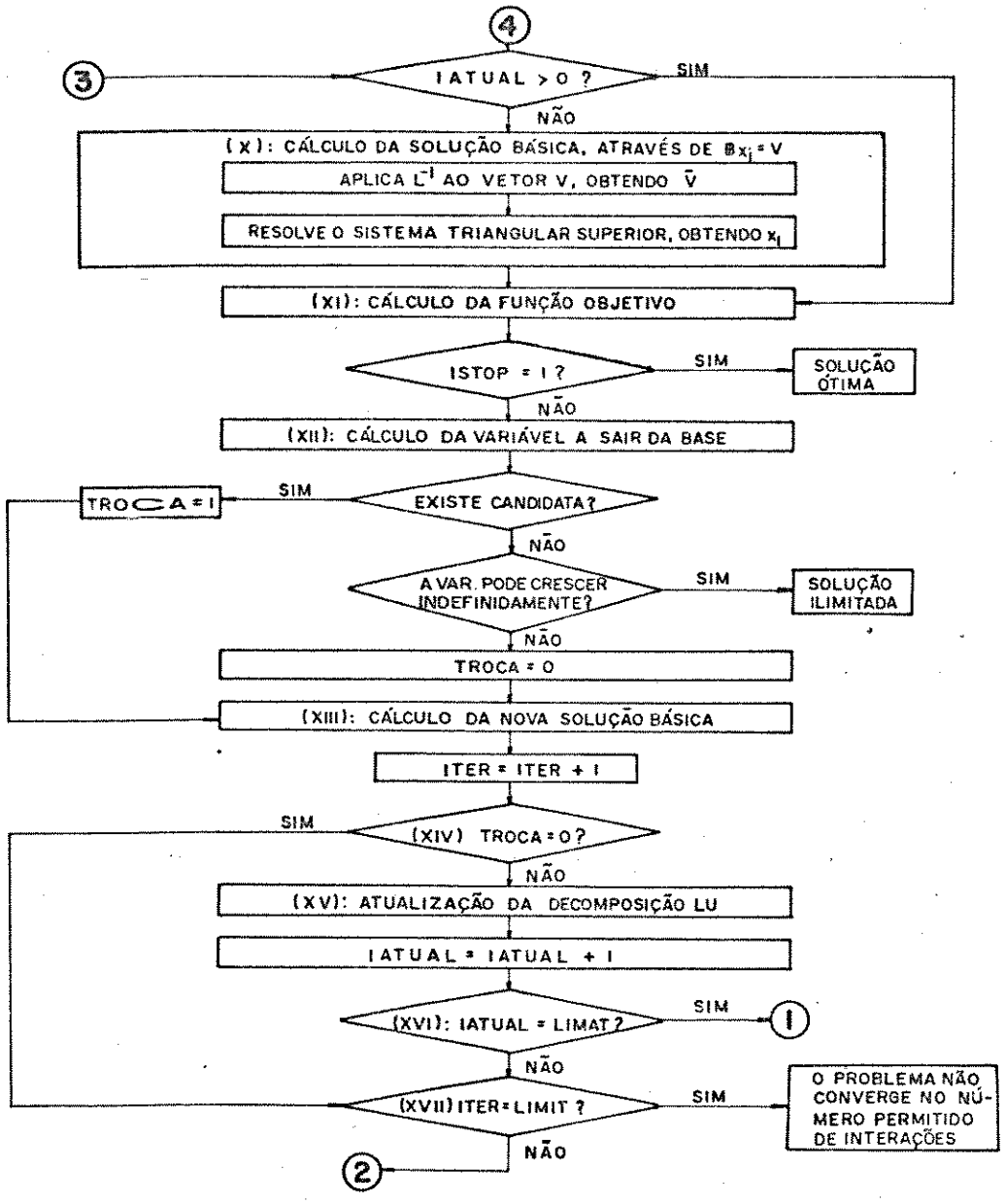
Neste capítulo, faremos o diagrama de blocos, detalhando a seguir, cada bloco do diagrama, conforme o programa computacional implementado para o algoritmo desenvolvido no capítulo 2 deste trabalho.

4.1. O diagrama de blocos do algoritmo

Este algoritmo foi implementado no programa SIMESC.LU e resolve o problema linear dinâmico (P) definido no capítulo 2, supondo conhecida uma base B inicial.

DIAGRAMA DE BLOCOS





Daremos a seguir, os detalhes deste diagrama de blocos, acentuando sobretudo os pontos onde ele se distingue de um programa simplex clássico:

BLOCO (I): ENTRADA DE DADOS E DEFINIÇÕES DE PARÂMETROS

- Entrada de Dados

Os dados são lidos na seguinte ordem:

1. R, M, N, T, NATRAS

   R = número de variáveis de controle a cada período

   M = número de restrições de controle a cada período

   N = número de variáveis de estado a cada período

   T = horizonte de planejamento

   NATRAS = máximo atraso nas variáveis de estado e controle na equação de estado.

O programa foi implementado considerando-se a possibilidade de que a equação de estado apresente atrasos tanto no vetor de estado quanto no vetor de controle. Neste caso, a equação de estado para o problema (P) enunciado no capítulo 2,

passa a ser representada como:

$$x(t+1) = \sum_{j=0}^{NATRAS} \left[A_j(t)\, x(t-j) + B_j(t)\, u(t-j) \right], t=0,\ldots,T-1$$

sendo que as outras equações permanecem as mesmas. A conseqüência da introdução dos retardos é o fato de que a matriz de restrições A e, consequentemente a base B, tornam-se mais cheias no triângulo inferior, implicando numa menor esparsidade nas matrizes L obtidas no processo, sem que se altere a matriz U. O método de resolução não sofre alteração alguma.

2. VALORA ( )

VALORA (I) contém o valor numérico do elemento I não nulo da matriz de restrições A, armazenada por colunas.

3. LINHAA ( )

LINHAA (I) indica a linha da matriz A à qual corresponde o elemento I armazenado em VALORA.

4. ICOLA ( )

ICOLA (L) corresponde à posição de VALORA que contém o primeiro elemento não nulo da coluna L.

5. INDB ( )

INDB ( ) indica os índices das variáveis básicas.

6. INDNB ( )

INDNB ( ) indica os índices das variáveis não básicas. INDNB (I) é negativo se a variável não básica I estiver no limite inferior e positivo em caso contrário.

7. VALORC ( )

VALORC ( ) contém o valor numérico do elemento I não nulo do vetor de custos Z.

8. ICOLC ( )

ICOLC (I) indica qual é a posição, no vetor de custos Z, do elemento I armazenado em VALORC (I).

9. VALORV ( )

VALORV (I) contém o valor numérico do elemento I não nulo do vetor de recursos V.

10. ICOLV ( )

ICOLV (I) indica qual é a posição, no vetor de recursos V, do elemento I armazenado em VALORV (I).

11. XMAX ( )

XMAX (I) é o limite superior da variável x(I).

12. XMIN ( )

XMIN (I) é o limite inferior da variável x(I).

- Definição de parâmetros.

Os parâmetros definidos são os seguintes:

1. LIMIT

LIMIT é o número máximo permitido de iterações.

2. LIMAT

LIMAT é o número máximo permitido de atualizações.

3. NELA

NELA é o número de elementos não nulos da matriz A.

4. NN

NN é o número de colunas da matriz de restrições A.

5. NCNN

NCNN é o número de elementos não nulos do vetor de custos.

6. NVNN

NVNN é o número de elementos não nulos do vetor de recursos.

7. NELL

NELL é o número de posições reservadas para elementos não nulos de L.

8. NELU

NELU é o número de posições reservadas para os elementos das matrizes U_i, i = 1....2T da matriz U.

9. NESUJ

NESUJ é o número de posições reservadas para os elementos das colunas sobrantes da matriz U.

10. NCOSUJ

NCOSUJ é o número de colunas sobrantes previstas.

BLOCO (II): INICIALIZAÇÃO DE PARÂMETROS

Os parâmetros que são inicializados são os seguintes:

1. ISTOP = 0

ISTOP é o indicador da solução ótima. Só terá seu valor alterado quando não houver mais variáveis candidatas a entrar na base. Teremos então ISTOP = 1.

2. ITER = 0

ITER é o contador de iterações. A cada iteração realizada seu valor é aumentado de uma unidade.

3. IATUAL = 0

IATUAL é o indicador do número de atualizações da decomposição LU realizadas quando há troca de variáveis na base.

4. TROCA = 1

O parâmetro TROCA indica se houve ou não troca de variáveis na base. TROCA = 0 indica que a base continua a mesma e que uma variável não básica teve seu valor alterado de um limite ao outro. TROCA = 1 indica que houve troca de variáveis na base.

BLOCO (III): DECOMPOSIÇÃO LU INICIAL

A decomposição da base B é feita através da subrotina DELUST, sendo que, para cada bloco B_i, i = 1...2T, indicado na figura 2.2., juntamente com as colunas sobrantes dos blocos anteriores ao bloco i, o procedimento é o mesmo que aquele descrito no algoritmo apresentado no capítulo 1.

Para este caso, inicialmente é criado NCAT, número de colunas disponíveis para compor a U_i a ser criada neste instante. A seguir, faz-se a decomposição LU deste bloco de colunas, armazenando os resultados na forma compacta descrita no capítulo 3, isto é, criando os vetores VALORL, LINHAL e ICOLL, representando a matriz L^{-1}, além de U,USUJ, IPROS, ICOSUJ, ICONSU, LINHAS e IPROC, caracterizando a matriz U. Os índices das colunas que sobram na criação desta U_i são indicados no vetor ISOB. As colunas cujos índices são indicados em ISOB, juntamente com as colunas de B_{i+1}, constituirão o conjunto de colunas disponíveis para criar U_{i+1}, sendo que o programa implementado testa primeiro as colunas sobrantes e depois aquelas de

$\mathcal{B}_{i+1}$. Os índices das colunas sobrantes que entraram na formação de U_{i+1} são retirados do vetor ISOB e os índices de $\mathcal{B}_{i+1}$ que não entraram na formação de U_{i+1} passam a fazer parte de ISOB. O procedimento é repetido até que se tenha obtido a decomposição $L^{-1}\ \mathcal{B} = U$ global.

BLOCO (IV): TROCA = 0?

A pergunta é feita aqui porque depois da primeira iteração, se TROCA = 0, não houve troca de colunas na base implicando em que os multiplicadores π são os mesmos. Passamos então diretamente à escolha da variável a entrar na base.

BLOCO (V): CRIA O VETOR DE CUSTOS BÁSICOS

Como o vetor Z de custos básicos está armazenado na forma compacta, aqui ele é reproduzido e colocado num vetor auxiliar, para ser utilizado no cálculo dos multiplicadores.

BLOCO (VI): OBTENÇÃO DOS MULTIPLICADORES

Os multiplicadores π são obtidos através de:

$$\mathcal{B}^T\ \pi = (Z^I)^T$$

$$(LU)^T\ \pi = (Z^I)^T$$

$$U^T L^T\ \pi = (Z^I)^T \longrightarrow \begin{cases} U^T\ y = (Z^I)^T \\ L^T\ \pi = y \longrightarrow \pi = (L^{-1})^T\ y \end{cases}$$

No programa implementado, eles são obtidos através da chamada de duas subrotinas: a primeira delas é MULTI1 que resolve o sistema $U^T y = (Z^I)^T$, obtendo y. Conforme mostra a figura 2.9., a matriz U^T é triangular inferior, composta das matrizes U_i, i = 1.... 2T transpostas e com algumas linhas isoladas, correspondentes às colunas sobrantes transpostas. A subrotina MULTI1 aproveita esta estrutura, resolvendo o sistema para cada U_i separadamente, através da rotina criada para resolver o sistema correspondente, no problema exposto no capítulo 1 e, a seguir, calculando as linhas isoladas, nas componentes correspondentes às colunas da U_i em questão. O processo é repetido até que o sistema seja resolvido para U^T_{2T}, significan-

do que o vetor y foi obtido.

A segunda subrotina é MULTI2, que aplica $(L^{-1})^T$ na solução y do sistema resolvido em MULTI1, obtendo os multiplicadores π.

BLOCO (VII): ESCOLHA DA VARIÁVEL A ENTRAR NA BASE

É feita através da subrotina VAENBA, que escolhe a primeira variável que não satisfaz o teste de otimalidade do simplex.

BLOCO (VIII): EXISTE CANDIDATA A ENTRAR?

Se não existir uma variável candidata a entrar na base, temos satisfeito o critério de otimalidade e consequentemente, a solução básica atual é ótima. Fazemos ISTOP = 1.

BLOCO (IX): ATUALIZAÇÃO DA COLUNA A ENTRAR NA BASE

O sistema a ser resolvido é:

$$B\hat{A}^j = A^j$$

$$LU\hat{A}^j = A^j$$

$$U\hat{A}^j = L^{-1} A^j = \bar{A}^j$$

Novamente, criamos duas subrotinas: ALCOLU, que aplica L^{-1} na coluna, obtendo $\bar{A}^j$ e AUCOLU, que resolve o sistema triangular superior $U\hat{A}^j = \bar{A}^j$. Neste caso também aproveitamos a rotina criada para resolver o sistema correspondente no programa detalhado no capítulo 1, resolvendo cada U_i, i = 1 .... 2T separadamente, resolvendo em seguida as colunas sobrantes.

BLOCO (X): CÁLCULO DA SOLUÇÃO BÁSICA

Antes de calcular a solução básica, perguntamos se IATUAL > 0. Isto porque, a cada iteração, a base B muda apenas em uma coluna. Então, resolvemos o sistema para calcular a solução básica apenas a primeira vez (IATUAL = 0), calculando depois a nova solução a partir da anterior. Se IATUAL = 0, resolvemos:

$$Bx_I = V$$

$$LUx_I = V$$

$$Ux_I = L^{-1} V = \tilde{V}$$

Como o sistema é idêntico àquele resolvido para atualizar a coluna a entrar na base, a menos dos vetores envolvidos, aplicamos as mesmas subrotinas criadas para aquele caso: ALCOLU e AUCOLU, obtendo a solução básica x_I.

BLOCO (XI): CÁLCULO DA FUNÇÃO OBJETIVO

É feito da maneira usual para o simplex canalizado.

A seguir, perguntamos se ISTOP = 1. Se sim, a solução é ótima e a função objetivo tem o valor calculado acima. Se não, passamos ao bloco seguinte.

BLOCO (XII): CÁLCULO DA VARIÁVEL A SAIR DA BASE

É calculada da maneira usual para o simplex canalizado, através da subrotina VASABA. Se existir uma candidata a sair da base, fazemos TROCA = 1 e passamos a calcular a nova solução básica. Se uma das variáveis não básicas apenas passou de um de seus limites ao outro, fazemos TROCA = 0, alteramos o sinal da variável em INDNB e calculamos a nova solução básica. No caso da variável poder crescer indefinidamente, a solução é ilimitada.

BLOCO (XIII): CÁLCULO DA NOVA SOLUÇÃO BÁSICA

É calculada a partir da anterior, considerando ou a troca de colunas ou a variável não básica assumindo valor diferente. Neste instante, acrescentamos uma unidade ao contador de iterações ITER.

BLOCO (XIV): TROCA = 0 ?

Se TROCA = 0, isto é, se não houve troca de variáveis na base, não fazemos a atualização da decomposição LU.

BLOCO (XV): ATUALIZAÇÃO DA DECOMPOSIÇÃO LU

A atualização da decomposição LU é feita conforme descrito no capítulo 2 e realiza-se através da chamada de treze subrotinas, sendo que grande parte do trabalho de atualização deve-se ao fato de trabalharmos com os dados armazenados na forma compacta.

Usaremos uma matriz U como exemplo, para facilitar a compreensão do trabalho realizado nas subrotinas. Seja, então, a matriz U representada abaixo, com m = 2, n = 3 e T = 2:

```
2. 1 . 2.      1.
   3 . 0.      2.
       1. 2. 3. 0.    1.
          2. 0. 2.    2.
             2. 1.    0.
                3. 1. 1. 2.
                   1. 0. 1.
                      2. 0. 2.
                         2. 3.
                            2.
```

Na forma compacta, esta matriz é representada por:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19...
U	2.	1.	3.	1.	2.	2.	3.	0.	2.	3.	1.	1.	2.	0.	2.	2.	3.	2.	

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17...
USUJ	2.	0.	1.	2.	0.	2.	1.	1.	2.	0.	1.	0.	2.	1.			
IPROS	2	0	4	5	6	7	0	9	10	11	12	0	14	0	16	17	18...

LIVRES = 15

POSIÇÃO	1	2	3	4	5	6....
ICOSUJ	3	6	8	9		
ICONSU	1	3	8	13		
LINHAS	1	1	3	6		
IPROC	2	3	4	0	6	7....

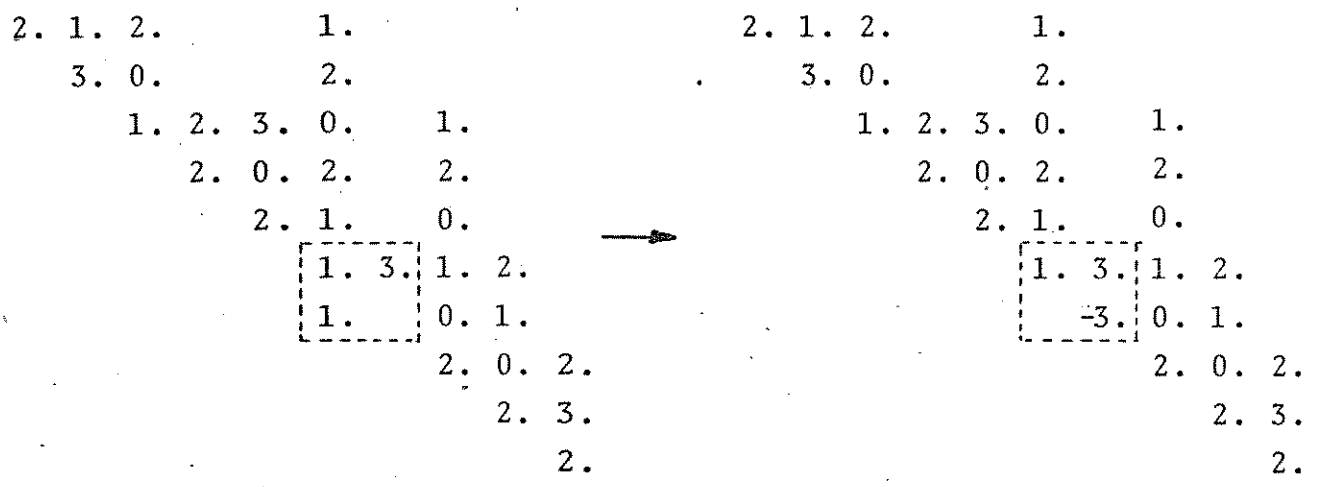
IPRIMC = 1

LIVREC = 5

A primeira subrotina chamada, DADSUJ, localiza a coluna a sair da base, em relação às colunas sobrantes: se a coluna que sai é uma coluna sobrante, qual é a coluna sobrante imediatamente anterior e imediatamente posterior àquela que sai e se existem outras colunas sobrantes nesta U_i onde está a variável a sair da base. Suponhamos que a coluna a sair da base seja a coluna 6, localizada em U_3. A subrotina determina então que a coluna é uma coluna sobrante, que a coluna imediatamente ante-

rior a ela é a coluna 1 (ISANT = 1, sendo que este valor indica a posição da coluna sobrante e não a posição da coluna na base, a qual é representada por ICOSUJ (ISANT)) e que a coluna imediatamente posterior é a coluna 3 (ISPOS = 3). A subrotina determina ainda que não existem mais colunas sobrantes em U_3, depois daquela que vai sair.

As subrotinas DPPBGL, ZEMATH, BGVAUX e PERUZI são chamadas para colocar a variável a sair da basena última posição de U_i, colocando as outras uma posição para trás e realizando os pivoteamentos necessários para triangularizar novamente U_i, conforme o processo de atualização exposto no capítulo 1. Vejamos através do exemplo, as alterações em U_3:



Como realizamos pivoteamentos para triangularizar novamente U_3, os pivôs são acrescentados em VALORL, LINHAL e ICOLL, a partir dos valores de KL e KICOL. Suponhamos que terminada a decomposição da base, em nosso exemplo, KL tenha um valor igual a 36. Sabemos que o indicador ICOLL tem 9 posições definidas (número de colunas da base sobre as quais foram realizados pivoteamentos para a triangularização) além de já estar definida a posição seguinte, onde terá início o armazenamento de uma nova coluna criada (ICOLL(10) = 36). Definimos agora VALORL (36) = - 1, LINHAL(36) = 7 e ICOLL(11) = 37, já indicando a posição de VALORL onde começará a ser armazenada uma nova coluna.

O novo vetor U passa a ser:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19...
U	2.	1.	3.	1.	2.	2.	3.	0.	2.	1.	3.	-3.	2.	0.	2.	2.	3.	2.	

A seguir, a subrotina PERSUJ permuta as colunas sobrantes, de maneira a realizar sobre elas as mesmas permutações que foram realizadas em U, (armazenada no vetor U). Deve

mos nos lembrar que a parte da coluna sobrante que não pertence à U_i está armazenada em USUJ, sendo portanto, permutada separadamente. Teremos então, no exemplo:

```
2. 1. 2.          1.
   3. 0.          2.
      1. 2. 3.    0. 1.
         2. 0.    2. 2.
            2.    1. 0.
               1. 3. 1. 2.
                  -3. 0. 1.
                      2. 0. 2.
                         2. 3.
                            2.
```

Como não mudou a ordem em que aparecem as colunas sobrantes, apenas o indicador ICOSUJ será alterado, pois agora ICOSUJ(2) = 7.

Se a coluna a sair da base estiver em U_{2T} (= U_4 no exemplo), procedemos diretamente à troca; se não, chamamos a subrotina BGUSUJ, que faz nas colunas de Φ_i as mesmas operações de permutação e pivoteamento feitas para triangularizar novamente U_i. Em nosso exemplo, aplicamos VALORL(36) à linha 7 de Φ_3:

```
2. 1. 2.          1.
   3. 0.          2.
      1. 2. 3.    0. 1.
         2. 0.    2. 2.
            2.    1. 0.
               1. 3. 1. 2.
                  -3. -1. -1.
                      2. 0. 2.
                         2. 3.
                            2.
```

e neste caso, o vetor USUJ é alterado para:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15...
USUJ	2.	0.	1.	2.	0.	2.	1.	1.	2.	0.	1.	-1.	2.	-1.	
IPROS	2	0	4	5	6	7	0	9	10	11	12	0	14	0	16...

LIVRES = 15

Neste instante, se todas as colunas de Φ_i tiverem um zero na linha correspondente à última linha de U_i, fazemos a troca das colunas correspondentes às variáveis a sair e a entrar na base. Se isto não acontecer, escolhemos a primeira coluna de Φ_i que tenha o elemento da última linha não nulo. Esta coluna será trocada com a coluna correspondente à variável a sair da base, que está agora na última posição de U_i. Chamamos então as subrotinas ZESUJ1 e ZESUJ2. A primeira zera as posições da coluna escolhida de Φ_i que estão abaixo da linha do pivô e que estão armazenadas em USUJ e a subrotina ZESUJ2 zera as posições desta mesma coluna e que constituem uma coluna de U_j, $j \geq i + 1$. No nosso exemplo, devemos verificar as posições de Φ_3, correspondentes à linha 7 da matriz U. Estas posições não são nulas, portanto não podemos fazer a troca das variáveis a entrar e sair da base. Escolhemos então a primeira coluna de Φ_3 com valor não nulo na linha 7, que corresponde à coluna 1 de Φ_3. As posições dessa coluna que estão abaixo da linha 7 e portanto, que devem ser zeradas, fazem parte de U_4, isto é, não existe uma parte da coluna armazenada em USUJ. Passamos diretamente à subrotina ZESUJ2, não sendo necessário chamar ZESUJ1. Temos:

```
2. 1. 2.          1.
   3. 0.          2.
      1. 2. 3.    0. 1.
         2. 0.    2. 2.
            2.    1. 0.
               1. 3. 1. 2.
                  -3.-1.-1.
                        0. 2.
                        2. 3.
                           2.
```

e definimos VALORL (37) = 2, LINHAL (37) = 8 e ICOLL (12) = 38.

A seguir, chamamos LTROCA, a subrotina que aplica as operações de pivoteamento realizadas para zerar os elementos da coluna de Φ_i, sobre a coluna correspondente à variável a sair e também a todas as outras colunas correspondentes às colunas sobrantes com elementos não nulos na linha do pivô, as quais são alteradas por estas operações de pivoteamento.

No exemplo, temos então:

```
2. 1. 2.          1.
   3. 0.          2.
      1. 2. 3.    0. 1.
         2. 0.    2. 2.
            2.    1. 0.
                  1. 3. 1. 2.
                  -3. -1. -1.
                  -6.    -2. 2.
                         2. 3.
                            2.
```

Chamamos então as subrotinas TTROCA e TRVAUX, que realizam a troca das duas colunas, sendo que agora, a coluna correspondente à variável a sair está em U_j. Estas subrotinas tem também a função de atualizar os vetores que caracterizam as colunas sobrantes, os quais foram alterados com esta troca. A matriz U fica então com a seguinte configuração:

```
2. 1. 2.          1.
   3. 0.          2.
      1. 2. 3.    1. 0.
         2. 0.    2. 2.
            2.    0. 1.
                  1. 1. 3. 2.
                  -1. -3. -1.
                  -6. -2. 2.
                      2. 3.
                         2.
```

e os arranjos que a caracterizam são alterados para:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19...
U	2.	1.	3.	1.	2.	2.	3.	0.	2.	1.	1.	-1.	-6.	-2.	2.	2.	3.	2.	

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17...
USUJ	2.	0.	1.	2.	0.	2.	1.	1.	2.	0.	3.	-3.	2.	-1.			
IPROS	2	0	4	5	6	7	11	9	10	0	12	0	14	0	16	17...	

LIVRES = 15

POSIÇÃO	1	2	3	4	5	6	7	8...
ICOSUJ	3	8	7	9				
ICONSU	1	3	8	13				
LINHAS	1	1	3	6				
IPROC	3	4	2	0	6	7...		

IPRIMC = 1

LIVREC = 5

Note-se que com as trocas, não mudamos a ordem dos elementos das colunas sobrantes em USUJ e sim, indicamos as alterações através de IPROS, assim como os elementos que caracterizam as colunas sobrantes são indicados nas mesmas posições de ICOSUJ, ICONSU e LINHAS, sendo que a troca é indicada pela alteração em IPROC.

O processo agora começa a se repetir e só se encerra quando atingimos uma das duas situações citadas (ou a coluna a sair corresponde à última coluna de U_{2T} ou a última linha de Φ_j, correspondente à U_j na qual está a coluna correspondente à variável a sair, é nula), favoráveis à troca. Chamamos então a subrotina TRODEF, que realiza a troca das variáveis a entrar e a sair da base, atualizando INDB e INDNB. Em nosso exemplo, a coluna correspondente à variável a sair está na coluna 1 de U_4, neste caso U_{2T}. Então, devemos levá-la à posição da última coluna de U_4, fazer os pivoteamentos necessários para triangularizar novamente U_4 e em seguida fazer a troca de colunas, pois atingimos uma situação favorável à troca. Levando a coluna 1 à posição 3 de U_4, teremos:

```
2. 1. 2.                1.
   3. 0.                2.
      1. 2. 3.    1. 0.
         2. 0.    2. 2.
            2.    0. 1.
               1. 1. 3. 2.
                  -1. -3. -1.
                     -2. 2. -6.
                      2. 3.
                         2.
```

Para triangularizar novamente U_4, fazemos os pivoteamentos:

```
2. 1. 2.                1.
   3. 0.                2.
      1. 2. 3.    1. 0.
         2. 0.    2. 2.
            2.    0. 1.
               1. 1. 3. 2.
                  -1. -3. -1.
                     -2. 2. -6.
                         5. -6.
                            2.
```

para este pivoteamento realizado, definimos VALORL (38) = 1, LINHAL (38) = 9 e ICOLL (13) = 39. A seguir, zeramos o último elemento da coluna 2 de U_4:

```
2. 1. 2.                1.
   3. 0.                2.
      1. 2. 3.    1. 0.
         2. 0.    2. 2.
            2.    0. 1.
               1. 1. 3. 2.
                  -1.-3.-1.
                     -2. 2.-6.
                         5.-6.
                             2.4
```

definindo VALORL(39) = - 0.4, LINHAL(39) = 10 e ICOLL(14) = 40.

O vetor U foi então alterado para:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19...
U	2.	1.	3.	1.	2.	2.	3.	0.	2.	1.	1.	-1.	-2.	2.	5.	-6.	-6.	2.4	

Agora é chamada a subrotina que permuta as colunas sobrantes:

```
2. 1. 2.                        1.
   3. 0.                        2.
      1. 2. 3.    1.            0.
         2. 0.    2.            2.
            2.    0.            1.
               1. 1.      2. 3.
                  -1.    -1.-3.
                     -2. 2.-6.
                         5.-6.
                             2.4
```

e os arranjos que caracterizam as colunas sobrantes sofrem as seguintes alterações:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16...
USUJ	2.	0.	1.	2.	0.	2.	1.	1.	2.	0.	3.	-3.	2.	-1.		
IPROS	2	0	4	5	6	7	11	9	10	0	12	0	14	0	16...	

LIVRES = 15

POSIÇÃO	1	2	3	4	5	6	7...
ICOSUJ	3	10	7	9			
ICONSU	1	3	8	13			
LINHAS	1	1	3	6			
IPROC	3	0	4	2	6	7...	

IPRIMC = 1
LIVREC = 5

Estamos agora em condições de realizar a troca das colunas na base. Como a coluna correpondente à variável a sair é a coluna 3 de U_4, a variável a entrar na base assumirá esta posição, sendo que os arranjos que representam a matriz U sofrerão as alterações correspondentes, e INDB e INDNB serão atualizados.

Realizada a troca de colunas, acrescentamos uma unidade ao contador IATUAL.

BLOCO (XVI): IATUAL = LIMAT?

Se já atingimos o limite permitido de atualizações, estamos com um arquivo muito grande de matrizes elementares do tipo L a serem aplicadas nos passos do processo, valendo a pena proceder a uma nova decomposição LU da base B. Voltamos então a inicializar o parâmetro IATUAL, reiniciando o processo de decomposição. Se IATUAL < LIMAT, passamos ao bloco seguinte.

BLOCO (XVII): ITER = LIMIT?

Se atingimos o limite de iterações, paramos o processo, avisando que o problema não converge neste limite. Caso contrário, voltamos ao Bloco (IV), repetindo o processo, até obtermos a solução ótima para o problema.

4.2. Exemplo

Vamos agora verificar, através de um exemplo, como se processam as iterações do método. Seja o problema (P) a seguir:

$$
(P)\begin{cases}
x(t+1)=A(t)x(t) + B(t)u(t) + k(t) & t=0 \ldots T-1 \\
x(0)=x^0 & \\
C(t)x(t) + D(t)u(t)=f(t) & t=0 \ldots T-1 \\
\alpha(t+1)\leq x(t+1)\leq\beta(t+1);\quad \gamma(t)\leq u(t)\leq\delta(t) & t=0 \ldots T-1 \\
\min J(x,u)= \sum_{t=0}^{T-1} \{c(t+1)x(t+1) + d(t)u(t)\} &
\end{cases}
$$

onde T=4, e r=3 , m=1, n=2 e

$$A(t)=\begin{bmatrix}-1 & 1\\ 2 & 1\end{bmatrix} \quad ; \quad B(t)=\begin{bmatrix}-1 & 2 & 0\\ 0 & 1 & -1\end{bmatrix} \; ; \; C(t)=\begin{bmatrix}2 & 1\end{bmatrix} \; ; \; D(t)=\begin{bmatrix}1 & 2 & -1\end{bmatrix}$$

$$\alpha(t+1)=\begin{bmatrix}0 & 0\end{bmatrix} \; ; \quad \gamma(t)=\begin{bmatrix}0 & 0 & 0\end{bmatrix} \; , \quad t=0 \ldots T-1$$

$$\delta(0)=\begin{bmatrix}20 & 30 & 50\end{bmatrix} \; ; \quad \beta(1)=\begin{bmatrix}50 & 50\end{bmatrix} ; \; \delta(1)=\begin{bmatrix}50 & 50 & 50\end{bmatrix} ; \; \beta(2)=\begin{bmatrix}50 & 50\end{bmatrix}$$

$$\delta(2)=\begin{bmatrix}40 & 60 & 30\end{bmatrix} \; ; \quad \beta(3)=\begin{bmatrix}100 & 100\end{bmatrix} ; \; \delta(3)=\begin{bmatrix}100 & 100 & 100\end{bmatrix} ; \; \beta(4)=\begin{bmatrix}100 & 100\end{bmatrix}$$

$$d(0)=\begin{bmatrix}-3 & 1 & 3\end{bmatrix} \; ; \quad c(1)=\begin{bmatrix}0 & 5\end{bmatrix} \; ; \quad d(1)=\begin{bmatrix}4 & 7 & -2\end{bmatrix} ; \; c(2)=\begin{bmatrix}-3 & 2\end{bmatrix}$$

$$d(2)=\begin{bmatrix}-1 & -2 & 3\end{bmatrix} \; ; \quad c(3)=\begin{bmatrix}8 & 6\end{bmatrix} \; ; \quad d(3)=\begin{bmatrix}1 & 8 & -2\end{bmatrix} ; \; c(4)=\begin{bmatrix}-1 & 1\end{bmatrix}$$

$$k(0)=\begin{bmatrix}56\end{bmatrix} ; \; f(0)=\begin{bmatrix}12\\ 7\end{bmatrix} \; ; \; k(1)=\begin{bmatrix}85\end{bmatrix} ; \; f(1)=\begin{bmatrix}45\\ 10\end{bmatrix} ;$$

$$k(2)=\begin{bmatrix}205\end{bmatrix} ; \; f(2)=\begin{bmatrix}45\\ 125\end{bmatrix} \; ; \; k(3)=\begin{bmatrix}205\end{bmatrix} \quad f(3)=\begin{bmatrix}5\\ 90\end{bmatrix}$$

A matriz de restrições para este problema é dada por:

$$\begin{array}{cccccccccccccccccccc}
1 & 2 & -1 & & & & & & & & & & & & & & & & & \\
-1 & 2 & 0 & -1 & 0 & & & & & & & & & & & & & & & \\
0 & 1 & -1 & 0 & -1 & & & & & & & & & & & & & & & \\
 & & & 2 & 1 & 1 & 2 & -1 & & & & & & & & & & & & \\
 & & -1 & 1 & -1 & 2 & 0 & -1 & 0 & & & & & & & & & & & \\
 & & 2 & 1 & 0 & 1 & -1 & 0 & -1 & & & & & & & & & & & \\
 & & & & & & & 2 & 1 & 1 & 2 & -1 & & & & & & & & \\
 & & & & & & -1 & 1 & -1 & 2 & 0 & -1 & 0 & & & & & & & \\
 & & & & & & 2 & 1 & 0 & 1 & -1 & 0 & -1 & & & & & & & \\
 & & & & & & & & & & & 2 & 1 & 1 & 2 & -1 & & & & \\
 & & & & & & & & & & -1 & 1 & -1 & 2 & 0 & -1 & 0 & & & \\
 & & & & & & & & & & 2 & 1 & 0 & 1 & -1 & 0 & -1 & & &
\end{array}$$

Esses valores são armazenados, por colunas, em VALORA, com os indicadores LINHAA e ICOLA. Os indices básicos são indicados em INDB:

INDB = (1, 2, 4, 5, 7, 9, 10, 11, 14, 16, 17, 19)

e os índices não básicos são os seguintes:

INDNB = (-3, -6, 8, 12, 13, -15, 18, -20)

lembrando que o sinal negativo indica que a variável está no limite inferior e o positivo, no limite superior.

O vetor de recursos é armazenado em VALORV:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12
VALORV	56	12	7	85	45	10	205	45	125	205	5	90
ICOLU	1	2	3	4	5	6	7	8	9	10	11	12

e o vetor de custos é armazenado em VALORC:

POSIÇÃO	1	2	3	4	5	6	7	8	9	10	11	12	13	14
VALORC	-3	1	3	5	4	7	-2	-3	2	-1	-2	3	8	6
ICOLC	1	2	3	5	6	7	8	9	10	11	12	13	14	15

POSIÇÃO	15	16	17	18	19
VALORC	1	8	-2	-1	1
ICOLC	16	17	18	19	20

Inicialmente, vamos fazer a decomposição LU da Base B. Tomamos a primeira coluna de INDB. Como m=1, o primeiro elemento da coluna torna-se o pivô, sem necessitarmos permutar linhas na procura do pivô adequado. Zeramos então os elementos que estão abaixo desta linha, criando as posições da matriz L^{-1}. O pivô é armazenado em U. Temos então: U(1)=1, VALORL(1)=1, LINHAL(1)=2 ICOLL(1)=1 e ICOLL(2)=2, lembrando que sempre definimos a primeira posição livre de VALORL em ICOLL.

Criamos U_1 e a segunda coluna de B_1 passa a ser uma coluna sobrante. Indicamos seu índice em ISOB. Esta coluna será a primeira a ser testada na criação de U_2. Repetimos o processo até obtermos a decomposição final.

REFERÊNCIAS BIBLIOGRÁFICAS

|1| ARMENTANO, V.A., "Programação linear dinâmica", Tese de Mestrado, DEE-FEC-UNICAMP, outubro 1979.

|2| BARTELS, R. & GOLUB, G., "The simplex method of linear programming using LU decomposition", Communications of the ACM vol. 12: 266-268, 1969.

|3| BRAMELLER, A. & ALLAN, R.N., "Sparsity: its practical application to systems analysis", Pitman Publishing, 1976.

|4| FORREST, J.J.H. & TOMLIN, J.A., "Updated triangular factors of the basis to maintain sparsity in the product form simplex method", Math. Programming, vol, 2: 263-278, 1972.

|5| FOURER, R., "Sparse gaussian elimination of staircase systems", Technical Report Sol 79-17, Department of Operations Research, Stanford University, 1979.

|6| KRIVONOZHKO, V.E. & CHEBOTAREV, S.P., "Factorization method for linear dynamic programming problems", Automation and Remote Control, nº 7, julho 1976.

|7| LUENBERGER, D.G., "Introduction to linear and nonlinear programming", Addison-Wesley, Reading, Massachusetts, 1973.

|8| PROPOI, A.I., "Problems of dynamic linear programming", RM-76-78, International Institute for Applied Systems Analysis (IIASA), Laxemburg, Austria, novembro 1976.

|9| PROPOI, A.I. & KRIVONOZHKO, V.E., "The simplex method for dynamic linear programs", RR-78-14, International Institute for Applied Analysis (IIASA), Laxemburg, Austria, setembro 1978.

|10| SAKAROVITCH, M., "Notes on linear programming", Van Nostrand, New York, 1971.