OTIMIZAÇÃO COM ESTRUTURA ESCADA NAS RESTRIÇÕES

Ana Friedlander

RELATÓRIO TÉCNICO Nº 09/87

RESUMO. O Problema de Programação Linear Dinâmica (P.L.D.) é resolvido mediante um algorítmo baseado no trabalho de Bartels-Golub, que usa a decomposição L-U duma matriz.

As matrizes de restrições do PLD têm uma estrutura particular chamada estrutura escada (stair case) e o algorítmo apresentado aqui tira proveito desta estrutura particular. É estabelecido um compromisso entre a presevação da estrutura escada e a estabilidade numérica do algorítmo.

Um programa computacional foi implementado, no qual as seguintes questões foram especialmente consideradas:

a) Técnicas de armazenamento esparso apropriadas a esta estrutura e ao algorítmo a ser programado.

b) Adaptação dos passos usuais do Método Simplex, com o objetivo de aproveitar os esquemas especiais para a estrutura escada introduzidos. São apresentadas experiências computacionais e um estudo de caso.

É discutida a aplicação dos resultados apresentados na resolução de problemas de programação não linear.

Universidade Estadual de Campinas
Instituto de Matemática, Estatística e Ciência da Computação
IMECC – UNICAMP
Caixa Postal 6065
13.081 - Campinas, SP
BRASIL

Fevereiro – 1987

ÍNDICE

INTRODUÇÃO

As primeiras publicações formais, na área da programação linear apareceram na revista Econométrica(Julho-Outubro 1949) , com ênfase situada na resolução de problemas sequenciais(dinâmicos)lineares. Nesses artigos(Wood-Dantzig, 1949; Dantzig, 1949) são discutidos modelos matemáticos aplicados à economia e formulado o modelo sequencial "escada"(staircase) ou dinâmico na forma:

$$\max \quad \gamma^{(1)} x^{(1)} + \ldots\ldots\ldots\ldots\ldots\ldots + \gamma^{(T)} x^{(T)}$$

sujeito a

$$\begin{aligned}
\alpha^{(1)} x^{(1)} &= a^{(1)} \\
-\bar{\alpha}^{(1)} x^{(1)} + \alpha^{(2)} x^{(2)} &= a^{(2)} \\
-\bar{\alpha}^{(2)} x^{(2)} + \alpha^{(3)} x^{(3)} &= a^{(3)} \\
&\ddots \\
-\bar{\alpha}^{(T-1)} x^{(T-1)} + \alpha^{(T)} x^{(T)} &= a^{(T)}
\end{aligned}$$

onde $x^{(t)}$ é um vetor em E^n de elementos não negativos e $\bar{\alpha}^{(t)}$, $\alpha^{(t)}$ matrizes de dimensões coerentes.

Logo após a formulação deste modelo ficou claro para os au

tores que, mesmo para problemas simples, o tamanho do programa linear resultante era excessivo para ser resolvido com as técnicas e recursos disponíveis.

Desde sua apresentação inicial até hoje o método Simplex foi implementado em pacotes comerciais que resolvem problemas com milhares de variáveis. Um histórico interessante sobre os primeiros passos no desenvolvimento de pacotes comerciais de Programação Linear P.L. encontra-se no livro de Orchard-Hays(1978a) Para mais detalhes sobre o projeto e implementação de Programas de P.L. ver Orchard-Hays(1968, 1978b,c), Benichou e outros(1977) e Murtagh(1981).

Os métodos de resolução de P.L.diferem essencialmente na maneira em que os sistemas Bx=b e B'x=d são resolvidos. As primeiras implementações armazenavam explícitamente a inversa da matriz B. Isto é proibitivo quando os problemas são de grande porte, mesmo usando o método Simplex revisado. A maioria dos problemas de grande porte têm a propriedade de que a matriz associada tem poucos elementos não nulos, fato que pode ser aproveitado pelo Simplex revisado utilizando a forma produto como esquema de armazenamento da inversa da base(Orchard-Hays,1968; Lasdon,1970). Desde 1962, tem aumentado o interesse em esquemas de representação compacta da matriz básica. Dantzig(1963) sugere uma representação compacta para uma base com estrutura escada baseada na forma produto da inversa. Mas até aqui, o problema da estabilidade numérica dos algoritmos não é comtemplada. Bartels e Golub

(1969) apontam à vulnerabilidade dos métodos que usam a forma produto da inversa, do ponto de vista da estabilidade numérica e apresentam como alternativa o uso da decomposição L-U da base. Uma análise dos efeitos dos erros de arredondamento sobre os cálculos realizados no método Simplex quando a forma produto é usada é apresentada por Bartels(1971). No mesmo artigo é analisado o comportamento da implementação que utiliza a decomposição L-U da base e é mostrado que o algoritmo introduzido em Bartels-Golub(1969) é estável numéricamente.

Os avanços em P.L.devem-se em grande parte ao crescente conhecimento concernente à resolução de sistemas lineares de grande porte.

Um sistema linear de equações Bx=b é chamado esparso se é possível resolve-lo mais eficientemente através do conhecimento da distribuição dos elementos nulos e não nulos, respectivamente, na matriz B. Dizemos que um sistema é estruturalmente esparso se os elementos não nulos da matriz estão confinados a certas partes de B conhecidas.

O método de eliminação gaussiana(Forsythe-Moler,1967) é o mais usado na resolução de sistemas esparso. Este método consiste essencialmente, na fatoração B= LU , onde L é uma matriz triangular inferior e U uma matriz triangular superior. O sistema Bx=b é reduzido à resolução de dois sistemas triangulares L(Ux)=b que são simples de resolver. Eliminação esparsa gaussiana é o no

me dado a qualquer adaptação deste método básico que aproveite a esparsidade de B, no sentido de que a proporção de elementos não nulos e a estrutura de B, se for o caso, sejam mantidas nas matrizes L e U.

Um dos primeiros esquemas de seleção de pivôs na eliminação duma matriz esparsa deve-se a Markowitz(1957). Neste esquema as linhas e colunas da matriz são escolhidas a cada passo da fatoração L-U de modo a minimizar a criação de novos elementos não nulos nas linhas e colunas que ainda restam para ser fatoradas (fill-in). Este esquema foi implementado por Reid(1975,1976).

Hellerman e Rarick(1971,1972) apresentaram um outro esquema para eliminação gaussiana esparsa, no qual a ordem das linhas e colunas é determinada antes de proceder a fatoração da matriz. No procedimento por eles descrito, a matriz tem as suas linhas e colunas permutadas de maneira tal, que a matriz resultante é triangular inferior exceptuando um certo número de blocos diagonais chamados "bumps". A seguir cada "bump" é processado até transformá-lo em triangular inferior a menos de algumas colunas com elementos não nulos acima da diagonal do "bump", chamadas "spikes"(O conceito de colunas "spikes" será muito usado em nosso trabalho, portanto chamaremos as colunas "spikes" de colunas "espeto"). É possível demonstrar que o processo de fatoração L-U aplicado sobre uma matriz com a estrutura assim obtida, só pode provocar a aparição de novos elementos não nulos nas colunas "espe

to".

Atualmente, os métodos mais eficientes para resolver problemas de programação linear esparsa de grande porte utilizam a fatoração L-U. As duas técnicas mais conhecidas para atualizar a fatoração L-U são as de Bartels e Golub(1969) e Forrest e Tomlin (1972). O esquema de Forrest e Tomlin é atrativo do ponto de vista da esparsidade mas não garante a estabilidade numérica do processo de atualização. O método de Bartels e Golub é o único método estável de atualização da fatoração L-U, mas sua aplicação direta provoca a perda da esparsidade nos fatores. Saunders(1976) mostrou que quando é aplicada a pré-seleção de pivôs de Hellerman e Rarick(1971,1972) antes da fatoração inicial, é possível predizer a região de U na qual serão criados novos elementos não nulos na atualização seguinte, permitindo uma implementação do esquema de Bartels e Golub que trabalhe exclusivamente com esta parte da matriz U.

O desenvolvimento de algoritmos especializados para aproveitar estruturas particulares, pode resultar em ganhos significativos, tanto na eficiência da resolução dos problemas como na utilização da memória do computador. Estes métodos são obrigatórios no caso de problemas realmente grandes, que não poderiam ser resolvidos com técnicas gerais devido a limitações de tempo e memória dos equipamentos disponíveis.

Vários trabalhos tem sido feitos na tentativa de resolver

problemas de P.L. com estrutura escada, também chamados problemas de programação linear dinâmica(P.L.D.) Métodos baseados na decomposição de Dantzig-Wolfe(1960) são apresentados por Glassey(1971, 1973) e Ho e Manne(1974).

Um outro enfoque, encontrado nos artigos de Grinold(1972)e Aonuma(1978), utiliza os métodos chamados de transformação, que começam com um P.L. mais simples e a partir da solução deste caminham para a resolução do P.L. original.

Finalmente, temos os métodos de representação compacta da base ou da sua inversa, junto com alguma variante do método primal Simplex. Este tratamento do problema foi sugerido por Dantzig (1955 , 1963) . Alguns algoritmos com este enfoque são apresentados nos trabalhos de Heesterman e Sandee(1965), Dantzig(1973), Propoi e Krivonzhko (1978), Wollmer(1977) e Perold e Dantzig (1978).

Nenhum método tem se mostrado mais eficiente que o Simplex clássico para a resolução duma grande variedade de problemas com estrutura escada. Isto motivou o desenvolvimento de pesquisas que procuram melhoramentos adicionais no método Simplex para o caso de problemas com estrutura escada. Alguns desses trabalhos tem sido muito bem sucedidos.

Simultâneamente surgiram técnicas novas para problemas de P.L. de grande porte, esparsos, sem uma estrutura particular. Estas técnicas foram adaptadas por Fourer(1979,1982,1983,1984)para

o caso de estrutura escada. Nesses artigos ele faz um estudo exaustivo das particularidades das matrizes com estrutura escada, discute várias alternativas de fatoração, de resolução dos sistemas lineares com esta estrutura e técnicas de cálculo dos custos reduzidos apropriadas a este caso.

Em nosso trabalho damos continuidade a esta linha de pesquisa introduzindo um novo método para modificar as fatorações L-U duma matriz quando uma única coluna desta é trocada. O processo, conhecido como de "atualização da fatoração" é necessário a cada iteração do método Simplex, quando há efetivamente uma troca de colunas. No método apresentado estabelecemos um compromisso entre a estabilidade numérica do algoritmo e a preservação da estrutura escada. Elaboramos um algoritmo para a resolução do P.L.D. utilizando este novo esquema de atualização e desenvolvemos um programa computacional para este algoritmo.

No capítulo I apresentamos o algoritmo. No capítulo II discutimos alguns aspectos computacionais do programa implementado e apresentamos resultados de experiências numéricas. A aplicação deste algoritmo na resolução do problema de otimização da operação dum sistema hidroelétrico é comentada no capítulo III.

As técnicas desenvolvidas para P.L.D. podem ser utilizadas também, na resolução de problemas dinâmicos com função objetivo não linear. Uma extensa discusão sobre as particularidades deste problema, encontra-se no livro de Gill, Murray e Wright(1981). O

primeiro objetivo no desenvolvimento de algoritmos para resolver estes problemas é a determinação de direções de decréscimo eficientes. Rosen(1960) propôs o método do gradiente projetado e Wolfe(1962) apresentou o conhecido método do gradiente reduzido (G.R.). Murtagh e Saunders(1978) desenvolveram uma especialização do G.R. para problemas de grande porte com função objetivo não linear. As idéias deste algoritmo foram implementadas num programa computacional de muito sucesso MINOS, (Murtagh e Saunders,1977) desenvolvido na Universidade de Stanford.

No capítulo IV mostramos como as técnicas desenvolvidas nos capítulo I e II para P.L.D. podem ser usadas na resolução de problemas de programação dinâmica com restrições lineares e função objetivo não linear. Discutimos também a implementação dum programa baseado no método do gradiente reduzido para problemas de grande porte com essas características e estrutura escada. O programa utiliza o conceito de variáveis superbásicas proposto por Murtagh e Saunders(1977). No mesmo capítulo fazemos uma exposição detalhada deste conceito e sua vinculação à estratégia de restrições ativas e discutimos a aplicação do algoritmo em problema análogo ao tratado no capítulo III.

O problema de minimizar uma função objetivo sujeita a restrições não lineares é consideravelmente mais complicado. Esta complicação resulta em métodos bem mais complexos que os utilizados para restrições lineares. Uma ampla discussão sobre os pro-

blemas que surgem e os vários métodos de resolução destes, estão no livro de Gill, Murray e Wright(1981). Rosen(1961)sugeriu um método de gradiente projetado para restrições não lineares baseado no seu próprio método para restrições lineres. Um método de gradiente reduzido generalizado(G.R.G.) foi apresentado pela primeira vez por Abadie e Carpentier(1965,1969). O programa G.R.G. implementado na Eletricité de France(Abadie e Guigou,1970;Abadie 1978). tem sido usado com sucesso na solução de muitos problemas reais. Num estudo comparativo de Colville(1968) os métodos G.R.G. foram considerados os mais eficientes para tratar restrições não lineares. Trabalhos nesta linha foram realizados por Sargent e Murtagh(1973) e Lasdon e outros(1974,1975).

Entrando especificamente na utilização de métodos G.R.G. para resolução de problemas dinâmicos de grande porte, esparsos , com restrições não lineares temos os trabalhos de Abadie e Bichara(1973), Facó(1977) e Drud(1976a, 1976b, 1985).

No capítulo V expomos as idéias fundamentais para o projeto de algoritmos do tipo G.R.G. para o caso de problemas dinâmicos com restrições não lineares. Apresentamos também a discussão de uma implementação computacional e um estudo de caso, em planejamento da operação de sistemas hidroelétricos.

CAPÍTULO I

UM ALGORITMO PARA PROGRAMAÇÃO LINEAR DINÂMICA

I.1. FORMULAÇÃO DO PROBLEMA

O problema de programação linear dinâmica (P.L.D.) é representado como segue

$$\min J(x,u) = \sum_{t=0}^{T-1} (c(t+1)' x(t+1) + d(t)'u(t))$$

sujeito a

$$(1) \quad x(t+1) = A(t)x(t) + B(t)u(t)$$

$$(2) \quad C(t)x(t) + D(t)u(t) = f(t)$$

$$(\text{P.L.D.}) \quad (3) \quad x(0) = x^{o}$$

$$(4) \quad \underline{x} \leq x(t+1) \leq \overline{x}$$

$$(5) \quad \underline{u} \leq u(t) \leq \overline{u}$$

$$\text{para} \quad t = 0,1,\ldots,T-1$$

onde:

$x(t)$: vetor de estados de E^{n} no período t.

$u(t)$: vetor de controles de E^r no período t.

$f(t)$: vetor de recursos, conhecidos de E^m com $m \leq r$.

$x(0) = x^o$: vetor de estados iniciais, conhecido.

T: horizonte de planejamento considerado fixo.

$A(t), B(t), C(t), D(t)$: matrizes conhecidas de dimensões $n \times n$, $n \times r$, $m \times n$ e $m \times r$ respectivamente.

$c(t+1)$, $d(t)$: vetores dados de E^n e E^r respectivamente.

As equações (1), chamadas equações dinâmicas, caracterizam o aspecto dinâmico do P.L.D.

I.2. ASPECTOS GERAIS DO MÉTODO DE SOLUÇÃO

O problema P.L.D. pode ser reescrito na forma padrão de um programa linear

$$\min \; J(s)$$

sujeito a

$$As = b$$

$$\underline{s} \leq s \leq \bar{s}$$

A matriz A de restrições do P.L.D. é representada na figura I.1.

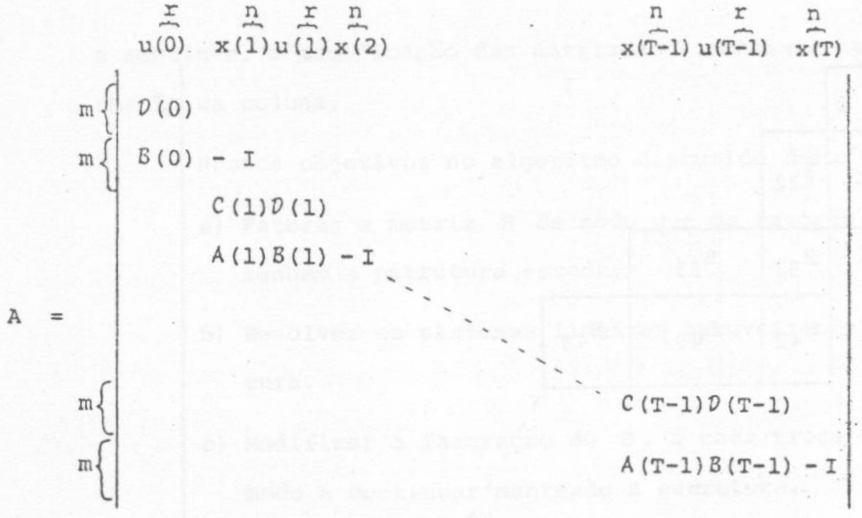


Figura I.1

Esta matriz é estruturalmente esparsa, e a sua estrutura é chamada escada (staircase). Neste capítulo apresentamos um algoritmo baseado na implementação estável do método Simplex de Bartels-Golub (1969) para resolver o problema P.L.D.

A cada iteração do método Simplex é necessária a resolução de sistemas lineares do tipo $Bx = b$ e $B'x = b$, onde B é uma submatriz da matriz A, formada pelas colunas básicas. O esforço computacional para a solução do problema, concentra-se, precisamente na resolução desses sistemas.

No caso do P.L.D., as matrizes B (bases), com as suas colunas reordenadas convenientemente, herdam a estrutura escada de A conforme mostra a figura I.2. Os elementos não nulos concentram-se na escada (parte hachurada).

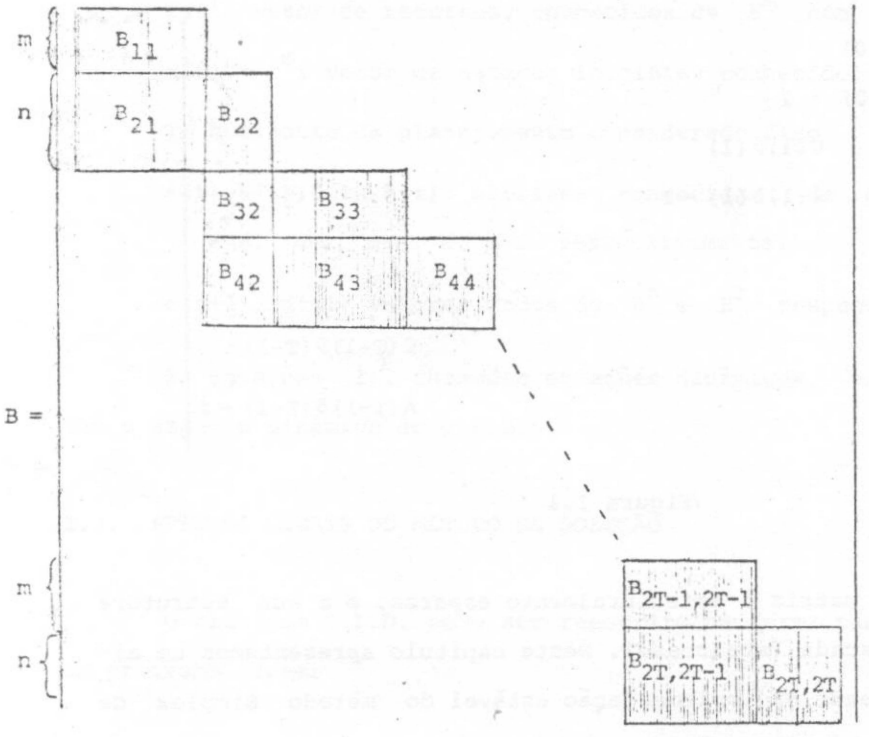


Figura I.2.

Para resolver os sistemas lineares usaremos a conhecida decomposição LU de B (Forsythe-Moler, 1967), ou seja, a matriz B será fatorada (salvo permutações) na forma B = LU onde L é uma matriz triangular inferior com 1's na diagonal e U triangular superior.

Lembramos que na passagem de uma iteração do Simplex para

a seguinte, a modificação das matrizes B consiste na troca de uma única coluna.

Nossos objetivos no algoritmo discutido neste capítulo são:

a) Fatorar a matriz B de modo que os fatores L e U mantenham a estrutura escada.

b) Resolver os sistemas lineares aproveitando essa estrutura.

c) Modificar a fatoração de B, a cada troca de coluna, de modo a continuar mantendo a estrutura.

Os itens a seguir detalham os aspectos a) e c) do algoritmo. A questão b) será discutida no capítulo II.

I.3. FATORAÇÃO DA BASE

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

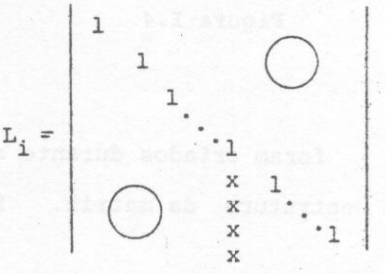


Figura I.3

O produto de todas essas matrizes constitui a matriz L^{-1} que verifica $L^{-1}B = U$. (Forstythe-Moler, 1967).

A fatoração implementada no algoritmo que apresentamos busca um compromisso entre a estabilidade e a preservação da estrutura.

Se pensássemos somente na estabilidade nossa estratégia de pivotamento seria, por exemplo, a de pivotamento parcial por colunas, que escolhe como pivô o elemento de maior valor absoluto da coluna. A figura I.4 ilustra as consequencias desta escolha sobre a estrutura escada.

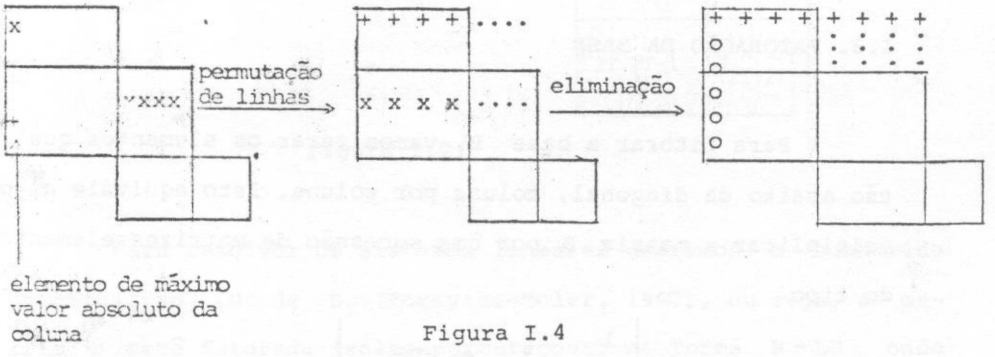


Figura I.4

Os elementos marcados + foram criados durante as operações de eliminação, mudando a estrutura da matriz. Este fenômeno é

conhecido como preenchimento (fill-in) sendo desejável evitá-lo o máximo possível; portanto, a escolha dos pivôs também deve considerar este objetivo.

Vamos agora explicar em detalhe o processo de fatoração usado para bases B com a estrutura escada ilustrada na figura I.2.

O bloco B_{11} tem m linhas e como B é uma base nesse bloco deve haver 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. (Na prática, para evitar algumas permutações, esta exigência pode ser relaxada aceitando como pivô o elemento b_{i1}, se $|b_{i1}| \geq \varepsilon |b_{k1}|$ para $k \in \{1,2,\ldots,m\}$, $k \neq i$ e ε dado). Permutamos as linhas se for necessário e passamos a zerar todos os elementos da coluna que estão abaixo do pivô. Observe-se que assim não é preenchida nenhuma posição estruturalmente nula da matriz B. A coluna que caracteriza a matriz elementar aplicada sobre B nestas operações está representada na figura I.5.

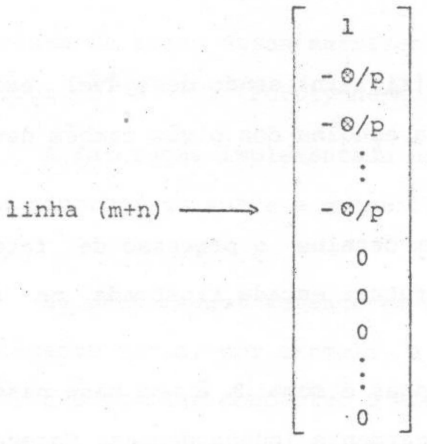


Figura I.5.

As marcas Ø indicam os elementos estruturalmente não nulos da primeira coluna de B e 'p' é o pivô.

Para fixar as idéias, vamos chamar P_1 à matriz de permutação e L_1 à matriz elementar que representa as operações de eliminação realizadas neste passo. O pivô 'p' caracteriza a primeira coluna de U. Note-se que a coluna da figura I.5 tem, no máximo, tantos elementos não nulos quantos tinha a primeira coluna de B antes de ser eliminada. Portanto, para representar L_1 precisamos somente desses elementos. Tomamos agora a segunda coluna de B. Aplicamos sobre ela P_1 e L_1, depois escolhemos como pivô o elemento de maior valor absoluto dessa coluna entre as linhas 2 a m. Se estes elementos forem todos nulos, significa que esta coluna de B_{11} depende linearmente da anterior. Neste

caso a abandonamos temporariamente e passamos a testar a coluna seguinte. Uma vez aceita uma coluna e escolhido o segundo pivô, fazemos a permutação P_2 e as operações necessárias para zerar os elementos abaixo da terceira linha desta coluna. Estas operações definem uma matriz L_2 caracterizada pela coluna da figura I.6.

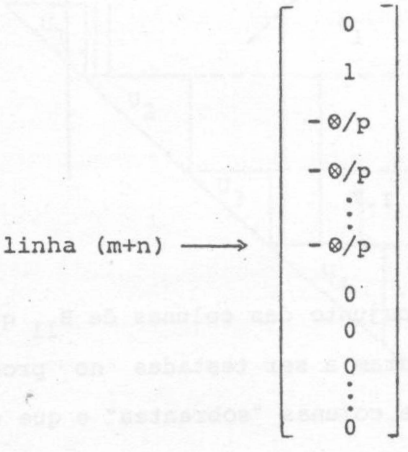


Figura I.6

As duas primeiras linhas da segunda coluna eliminada em B caracterizam a segunda coluna de U.

Repetimos o processo até obter um pivô para cada uma das m primeiras linhas de B. Isto é garantido pelo fato da matriz B ser uma base. Até aqui teremos efetuado as seguintes operações:

$$L_{m-1}P_{m-1}L_{m-2}P_{m-2} \cdots L_2P_2L_1P_1B$$

transformando a matriz B em $\tilde{B}$, como mostra a figura I.7

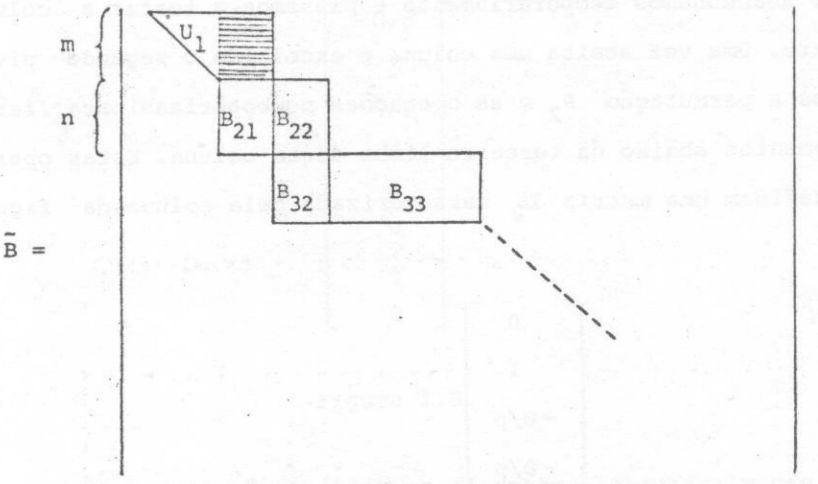


Figura I.7

A área hachurada representa o conjunto das colunas de B_{11} que foram abandonadas ou que não chegaram a ser testadas no processo anterior, as quais chamaremos de colunas "sobrantes" e que constituem uma matriz que chamaremos Φ_1. Durante todo este procedimento nenhuma posição estruturalmente nula de B foi preenchida acima da diagonal e o número de elementos necessários para caracterizar as L_i é exatamente o número de elementos não nulos abaixo da diagonal em B.

Para continuar o processo de fatoração consideramos a matriz cujas colunas são as sobrantes e as colunas de B_{22}, e cujas linhas são as n linhas a partir da linha do último pivô

estabelecido. Como no caso anterior, o fato de B ser uma base garante a obtenção de n pivôs, os quais serão escolhidos entre estas n linhas como antes.

Repetimos o processo descrito até completar a fatoração de B. A configuração final de U está representada na figura I.8

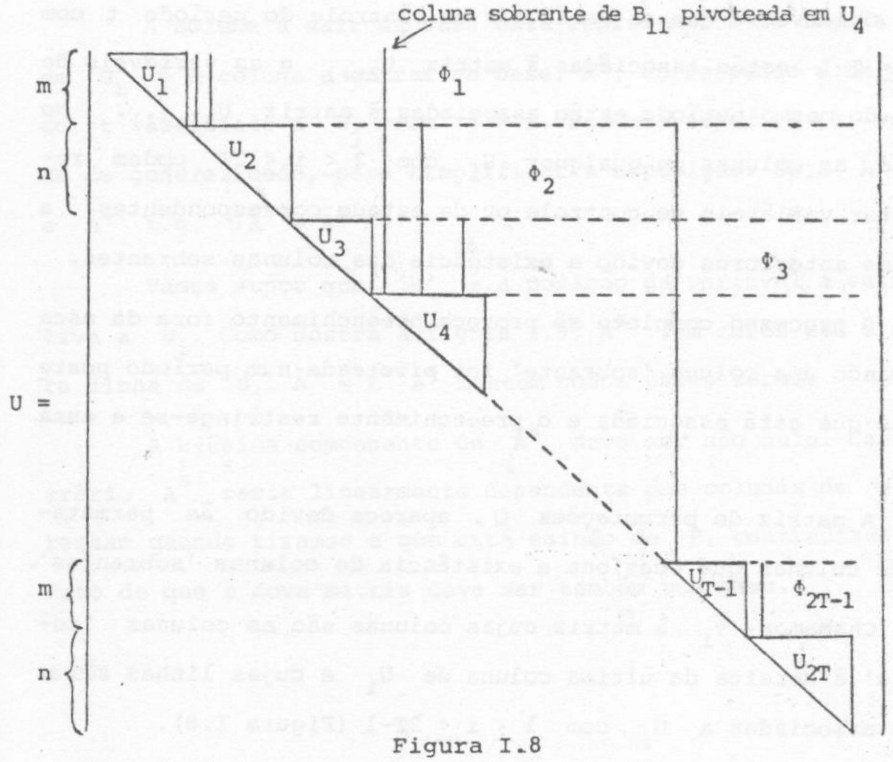


Figura I.8

Se definirmos

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

temos que $L^{-1} B Q = U$ onde Q é uma matriz de permutações.

Observe-se que as variáveis de controle do período t com $0 \leq t \leq T-1$ estão associadas à matriz U_{2t+1} e as variáveis de estado do mesmo período estão associadas à matriz $U_{2(t+1)}$. No entanto, as colunas de qualquer U_i com $2 \leq i \leq 2T$ podem representar variáveis de controle ou de estado correspondentes a períodos anteriores devido a existência das colunas sobrantes.

O processo completo só provoca preenchimento fora da escada, quando uma coluna 'sobrante' foi pivoteada num período posterior ao que está associada e o preenchimento restringe-se a essa coluna.

A matriz de permutações Q, aparece devido às permutações de colunas que ocasiona a existência de colunas 'sobrantes'.

Chamamos Φ_i à matriz cujas colunas são as colunas 'sobrantes' à direita da última coluna de U_i e cujas linhas são as linhas associadas a U_i, com $1 \leq i \leq 2T-1$ (Figura I.8).

I.4. ATUALIZAÇÃO DA FATORAÇÃO LU.

A cada iteração do Simplex uma coluna é escolhida para

entrar na base, e em função desta, uma coluna da base é determinada para sair.

Antes de descrever o processo de modificação da fatoração LU, mais conhecido como atualização da fatoração, faremos algumas considerações sobre as posições relativas entre as colunas a ser trocadas.

A coluna a sair da base está representada na figura I.9 em U_i e a coluna a entrar na base, A^s, corresponde a um período t associado a U_j com $j > i$. Podemos supor $Q = I$ sem perda de generalidade, para simplificar a exposição. Seja $\tilde{A}^s = L^{-1}A^s$ e $\hat{A}^s$ t.q. $U\hat{A}^s = \tilde{A}^s$.

Vamos supor que 'k' é a posição da variável a sair relativa a U_i. Como mostra a figura I.9, A^s tem zeros até a primeira linha de U_j. $\tilde{A}^s = L^{-1}A^s$ mantém todos esses zeros.

A k-ésima componente de $\hat{A}^s$ deve ser não nula. Caso contrário A^s seria linearmente dependente das colunas de B que restam quando tiramos a que está saindo de B, contradizendo o fato de que a nova matriz deve ser também uma base.

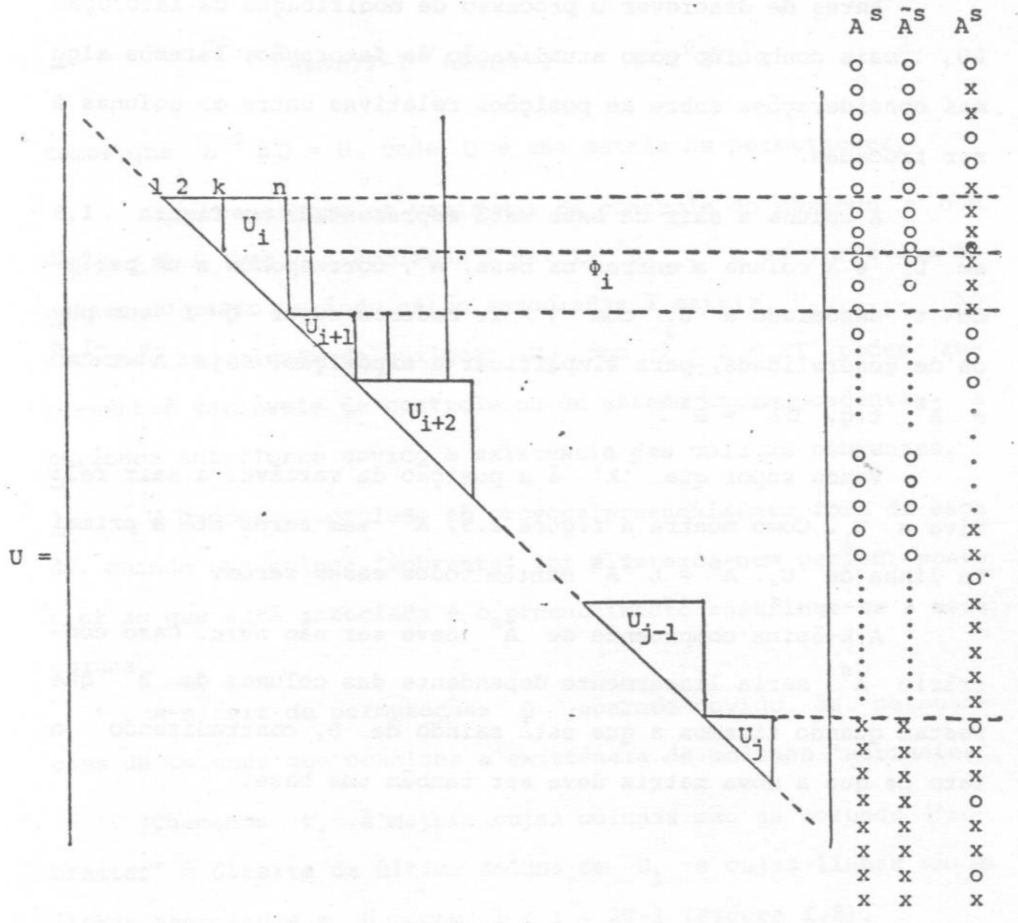


Figura I.9

Então seja $\hat{a}_i' = (\hat{a}_{i1}, \hat{a}_{i2}, \ldots, \hat{a}_{ik}, \ldots, \hat{a}_{in})$, cujas componentes são as de $\hat{A}^s$ que estão nas linhas associadas a U_i. (Supomos U_i com n linhas). Como $U\hat{A}^s = \tilde{A}^s$, temos que

$$U_i \bar{a}_i + \Phi_i \bar{a}_i = 0$$

onde $\bar{a}_i$ consiste dos elementos de $\hat{A}^s$ nas linhas abaixo da última linha de U_i. Obtemos

$$\hat{a}_i = -U_i^{-1} \Phi_i \bar{a}_i$$

Como $\hat{a}_{ik} \neq 0$, necessariamente a k-ésima linha de $U_i^{-1} \Phi_i$ deve possuir algum elemento não nulo. Este resultado, aliado ao fato de que as colunas de U_i constituem uma base de E^n nos assegura que alguma coluna de Φ_i pode substituir a k-ésima coluna de U_i, de maneira a continuar formando uma base de E^n.

Estamos agora em condições de descrever o processo de atualização. Consideramos dois casos:

a) A variável a sair da base está em U_i e a variável a entrar está associada a U_j com $j > i$.

b) A variável de saída está em U_i com $j \leq i$.

Analisamos o primeiro caso:

a) A coluna que sai está em U_i com $j > i$.

Seja k a posição da variável a sair, relativamente a U_i (ver figura I.9).

É claro que se fizermos a troca diretamente, U_i torna-se uma matriz singular. Qualquer permutação de linhas para introduzir um pivô aceitável e as operações necessárias para eliminar os elementos não nulos abaixo da linha k de U_i, que resultariam de fazer uma troca direta, provocariam um preenchimento indesejável em U. Para evitar isto, elaboramos a estratégia descrita a seguir.

Caminhamos com a coluna a sair até a posição da última coluna de U_i e trazemos cada uma das outras colunas a partir de (k+1)-ésima uma posição para trás. A matriz U fica com a forma representada na figura I.10.

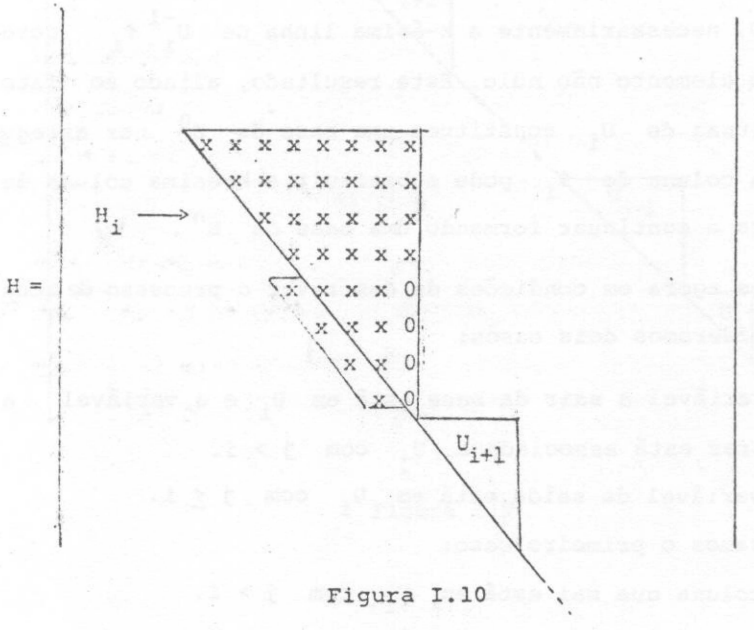


Figura I.10

A permutação de colunas faz com que U_i perca a triangularidade, transformando-a numa matriz Hessenberg, H_i. A matriz H_i é triangularizada, escolhendo como pivô em cada coluna a ser eliminada o elemento de maior valor absoluto entre $u_{\ell,\ell+1}$ e $u_{\ell+1,\ell+1}$ para $\ell = k,\ldots,n-1$. (Bartels-Golub , 1969). Estas operações aumentam o número de P_i's e L_i's em L^{-1}, e uma vez aplicadas tornam a matriz U novamente triangular, com a variável a sair colocada na posição da última coluna de U_i. Embora a matriz U_i seja modificada, para facilitar a exposição, continuaremos a chamar U_i à nova matriz triangular obtida.

Note-se que somente as colunas de U_i e de Φ_i são modificadas pelas operações descritas acima. As colunas de Φ_i que no fim deste passo apresentem um zero na linha n são linearmente dependentes das (n-1) primeiras colunas de U_i, em consequência, elas não serão consideradas candidatas para substituir a coluna que está saindo da base, em U_i.

Escolhemos a primeira coluna de Φ_i com um elemento não nulo na linha n. (As considerações feitas no começo deste item garantem a viabilidade desta escolha). Permutamos esta coluna de Φ_i com a coluna que sai da base. A matriz U fica com o aspecto apresentado na figura I.11.

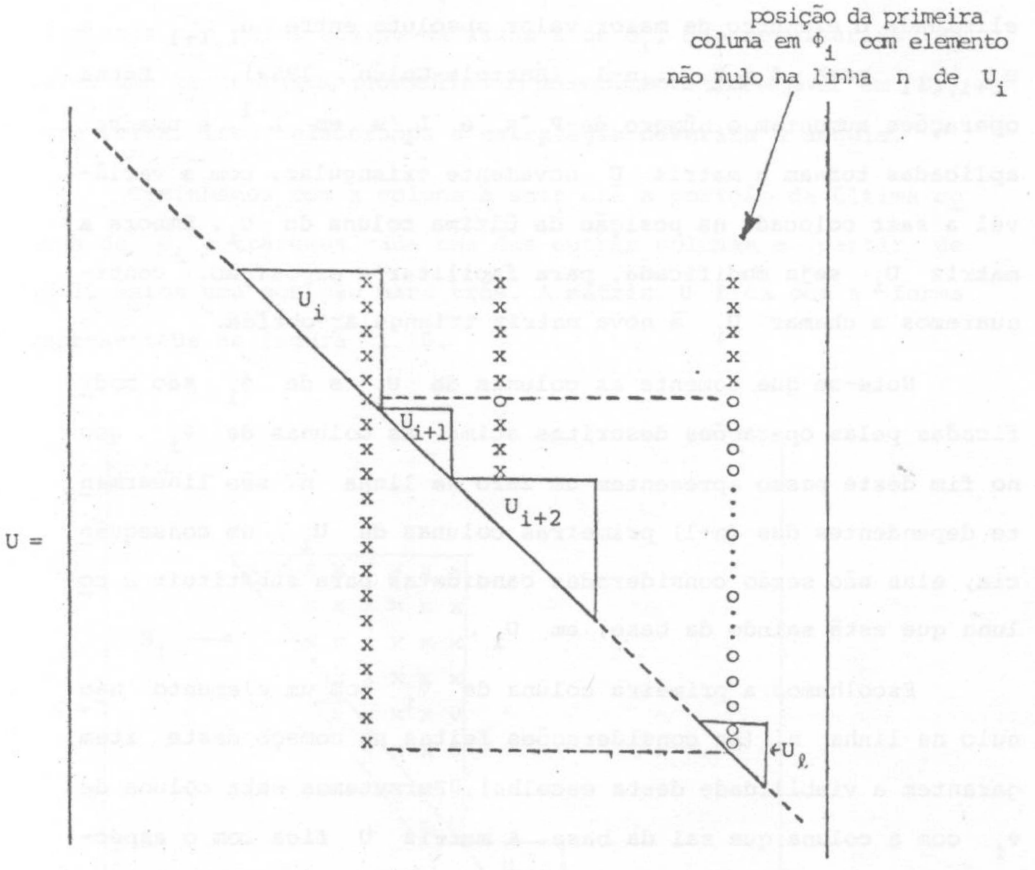


Figura I.11 - Aspecto parcial da matriz U após a permutação das colunas

Com esta permutação de colunas, U perde novamente a triangularidade. Passamos, então, a eliminar os elementos não nulos abaixo da diagonal (ver figura I.11).

Pensemos na linha n : os zeros presentes nas colunas de Φ_i anteriores à coluna escolhida garantem que não é alterado nenhum elemento da matriz U situado entre as duas colunas permutadas. Além disso, só sofrerão alterações os elementos situados em colunas posteriores à coluna de Φ_i escolhida que tiverem um elemento não nulo na linha correspondente a n-ésima linha de U_i. Estas operações também aumentam o número de L_i's em L^{-1}.

Agora a nossa coluna de saída está em U_ℓ com $\ell > i$. Repetimos este processo até colocar a coluna de saída na posição da última coluna de U_r com $r \geq j$. Esta situação configura o segundo caso:

b) A coluna que sai está em U_i com $j \leq i$.

A diferença reside em que neste caso, é possível que toda a linha de Φ_i na qual estamos procurando um elemento não nulo, seja nula. Quando isto acontece retiramos definitivamente a coluna que sai da base colocando no seu lugar a coluna que entra devidamente atualizada, isto é, todas as operações feitas sobre B e U, devem ser aplicadas sobre a coluna que entra. Quando fazemos esta troca de colunas, mais uma vez é possível que elementos não nulos surjam abaixo da diagonal de U. Mas como toda a linha de Φ_i correspondente à linha do pivô da eliminação que segue,

é nula, estas operações não provocam preenchimento. Se na linha de Φ_i pesquisada existe algum elemento não nulo (figura I.12) procedemos como em a) até que para algum ℓ com $\ell \geq i$, Φ_ℓ tenha a sua última linha nula ou $\ell = 2T$, isto é, a coluna de saída esteja colocada na posição da última coluna da matriz U. Então fazemos a troca de colunas introduzindo A^s devidamente atualizada na base.

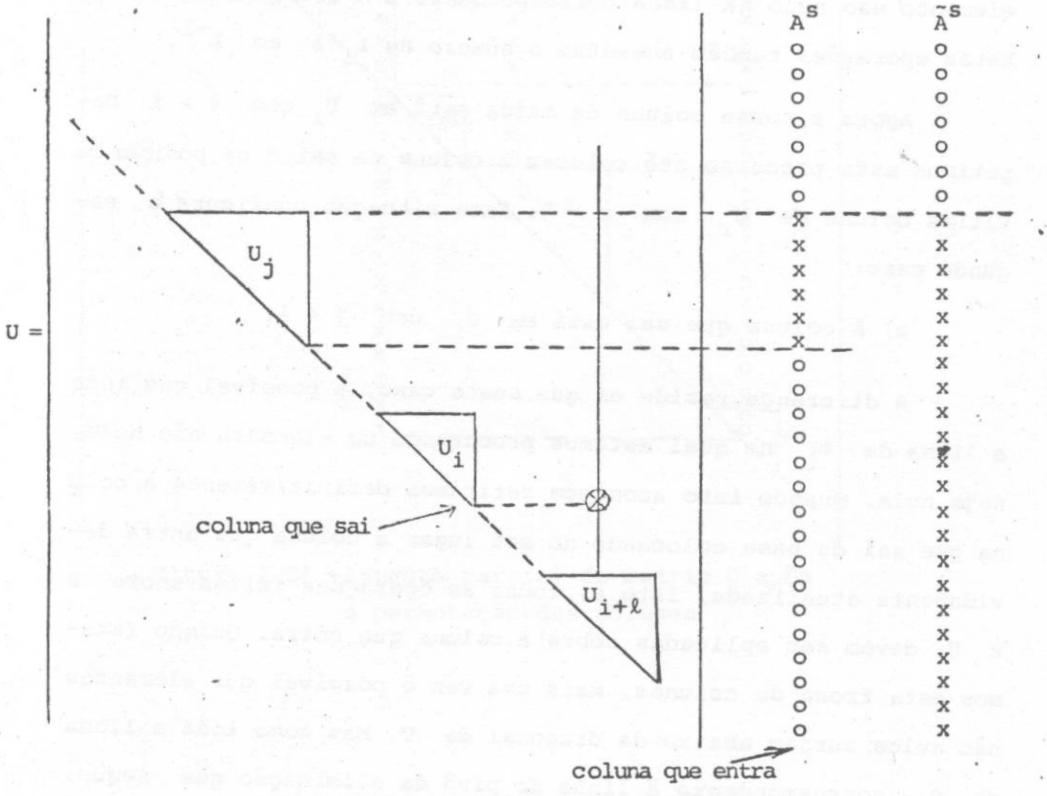


Figura I.12

Todo este processo preserva a estrutura escada, exceptuando as colunas 'sobrantes' da fatoração. Durante o processo de atualização alguma nova coluna sobrante pode ser criada nas Φ_i ou algumas colunas de Φ_i existentes podem ser eliminadas.

O algoritmo que propomos para resolver o P.L.D. é o clássico método Simplex revisado com decomposição LU da base, usando os esquemas apresentados para fatorar as matrizes básicas e atualizar essas fatorações. Também na resolução dos sistemas lineares que surgem a cada iteração do Simplex faremos uso da estrutura escada das LU obtidas. Depois de várias iterações nas quais a fatoração inicial foi sucessivamente modificada, é possível que o processo se desestabilize numericamente ou que as posições de memória necessárias para armazenar o crescente arquivo das L_i e P_i sejam excessivas. Para evitar estes problemas a matriz básica é refatorada quando o número de atualizações atinge um valor dado, que depende das particularidades de cada problema e dos recursos computacionais disponíveis.

I.4. EXEMPLO NUMÉRICO

Ilustramos a seguir o algoritmo com um pequeno exemplo numérico.

Seja B a matriz representada na figura I.13.

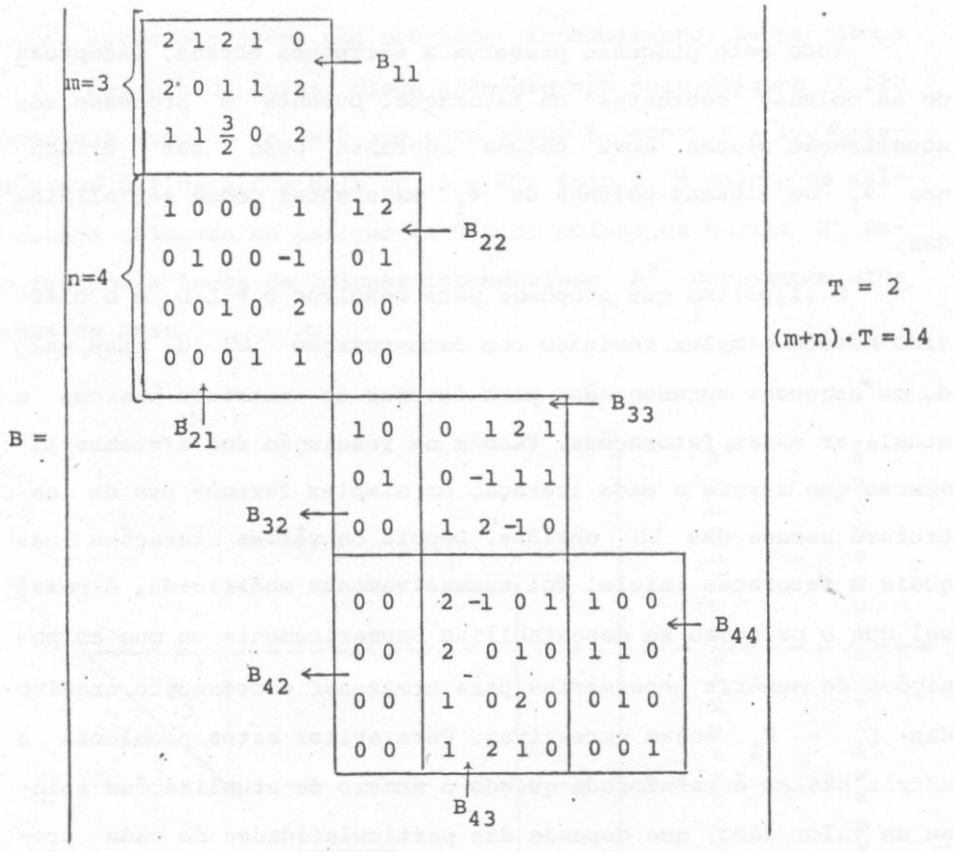


Figura I.13.

Em B_{11} escolhemos como pivô na primeira coluna o primeiro elemento e zeramos essa coluna até a quarta linha. Não havendo permutação de linhas $P_1 = I$. O vetor que caracteriza L_1 é

$$(1,-1,-1/2,-1/2,0,0,\ldots,0)'$$

e a primeira coluna de U_1 é $(2,0,0)'$.

Aplicamos L_1P_1 à segunda coluna resultando em

$$(1, \underline{-1}, \underline{1/2}, -1/2, 1, 0, \ldots, 0)'$$

Escolhemos o pivô entre os elementos sublinhados, obtendo o valor -1. Temos $P_2 = I$ e L_2 caracterizada por

$$(0, 1, 1/2, -1/2, 1, 0, \ldots, 0)'$$

e a segunda coluna de U_1 é dada por $(1,-1,0)'$.

Aplicamos $L_2P_2L_1P_1$ à terceira coluna, obtendo

$$(2, -1, \underline{0}, -1/2, -1, 1, 0, 0, \ldots, 0)'$$

O único elemento candidato a pivô nesta coluna é zero, portanto esta coluna é abandonada temporariamente.

Aplicamos então, $L_2P_2L_1P_1$ à quarta coluna e obtemos

$$(1, 0, \underline{-1/2}, -1/2, 0, 0, 1, 0, \ldots, 0)'$$

Esta coluna é aceita e o pivô vale $-1/2$, $P_3 = I$ e L_2 é representada por

$$(0, 0, 1, -1, 0, 0, 2, 0, 0, \ldots, 0)'$$

A terceira coluna de U_1 é $(1,0,-1/2)'$.

A matriz U_1 foi completada,

$$U_1 = \begin{vmatrix} 2 & 1 & 1 \\ & -1 & 0 \\ & & -1/2 \end{vmatrix}$$

Temos duas colunas 'sobrantes', a terceira e a quinta que não chegou a ser testada.

Começamos o processo de construção de U_2. Pegamos a primeira coluna sobrante que é a terceira. Aplicamos a ela $L_3P_3L_2P_2L_1P_1$ obtendo

$$(2, -1, 0, \underline{-1/2, -1, 1,} \ 0,0,\ldots,0)'$$

O pivô deve ser escolhido entre os valores sublinhados. Elegemos o valor -1. Então permutamos as linhas 4 e 5. P_4 representa esta operação e L_4 é caracterizada por

$$(0, 0, 0, 1, -1/2, 1, 0, 0, \ldots, 0)' .$$

A primeira coluna de Φ_1 é $(2,-1,0)'$, e estará colocada na posição da quarta coluna de U. A primeira coluna de U_2 é $(-1,0,0,0)'$. Aplicamos $L_4P_4L_3P_3L_2P_2L_1P_1$ à segunda coluna sobrante que é a

quinta de B, resultando

$$(0,\ 2,\ 3,\ 1,\ \underline{-7/2,\ 3,\ 7},\ 0,\ldots,0)' .$$

O pivô escolhido é 7. P_5 consiste em permutar as linhas 5 e 7 e L_5 é caracterizada por

$$(0,\ 0,\ 0,\ 0,\ 1,\ -3/7,\ 1/2,\ 0,0,\ldots 0)'$$

A segunda coluna de Φ_1 é $(0,2,3)'$ e estará situada na posição da quinta coluna de U. A segunda coluna de U_2 é $(1,7,0,0)'$.

Todas as colunas correspondentes a B_{11} foram usadas e passamos a testar colunas de B_{22} para determinar os dois pivôs que faltam para completar U_2.

A primeira coluna correspondente a B_{22} é

$$(0,0,0,\ 1\ ,0\ ,0\ ,0\ ,1,0\ \ 0,\ 0\ ,\ldots,0)' .$$

As operações L_i e P_i com $i = 1,2,3$ não a modificam. Se aplicamos $L_5P_5L_4P_4$ sobre ela resulta

$$(0,\ 0\ ,\ 0\ ,\ 0\ ,\ 0\ ,\ \underline{0\ ,\ 1}\ ,\ 1\ ,\ 0,0,0,\ldots,0)'$$

Escolhemos como pivô o valor 1 e P_6 consiste em permutar as

linhas 6 e 7. L_6 é caracterizada por

$$(0, 0, 0, 0, 0, 1, 0, -1, 0, 0, \ldots, 0)' .$$

A terceira coluna de U_2 é $(0,0,1,0)'$.

Testamos agora, a última coluna de B_{22} . Também sobre ela só precisamos aplicar as operações $L_i P_i$ com $i \geq 4$. Obtemos

$$(0, 0, 0, 1, 0, 3/2, 1, -3/2, 1, 0, \ldots, 0)' .$$

O único candidato a pivô é o valor 1. Então $P_7 = I$ e L_7 é caracterizada por

$$(0, 0, 0, 0, 0, 0, 1, 3/2, -1, 0, \ldots, 0)' .$$

A quarta e a última coluna de U_2 é $(1, 0, 3/2, 1)'$.

A matriz U_2 completa é

$$U_2 = \begin{vmatrix} -1 & 1 & 0 & 1 \\ & 7 & 0 & 0 \\ & & 1 & 3/2 \\ & & & 1 \end{vmatrix} .$$

Não temos nenhuma coluna sobrante portanto para formar U_3 começamos pela primeira coluna de B_{33}. As colunas correspondentes

a B_{33} têm zeros até a sétima linha, portanto as operações anteriores não as modificam.

O pivô deve ser escolhido entre os elementos sublinhados que constituem a primeira coluna de B_{33}

$$(0, 0, \ldots, \underline{0, 0, 1}, 2, 2, 1, 1)' .$$

O valor do pivô é 1 e P_8 indica a permutação das linhas 8 e 10. L_8 é representada por

$$(0, 0, 0, 0, 0, 0, 0, 1, 0, 0, -2, -2, -1, -1)' .$$

A primeira coluna de U_3 é $(1,0,0)'$.

Aplicamos L_8P_8 à coluna correspondente a segunda coluna de B_{33} obtendo

$$(0, \ldots, 0, 2, \underline{-1, 1}, -5, -4, 2, 0)' .$$

O pivô escolhido é -1, $P_9 = I$ e L_9 é representada por

$$(0, \ldots, 1, 1, -5, -4, -2, 0)' .$$

A segunda coluna de U_3 é $(2,-1,0)'$.

Aplicamos $L_9P_9L_8P_8$ à coluna correspondente a terceira coluna de B_{33} resultando

$$(0, \ldots, -1, 1, \underline{3}, -3, -1, 1, 2)' .$$

O pivô é 3 , $P_{10} = I$ e L_{10} é caracterizada por

$$(0,0, \ldots, 1, 1, 1/3, -1/3, -2/3)' .$$

A terceira e última coluna de U_3 é $(-1, 1, 3)'$.

A matriz U_3 completa é

$$U_3 = \begin{vmatrix} 1 & 2 & -1 \\ & -1 & 1 \\ & & 3 \end{vmatrix} .$$

Temos uma coluna sobrante de B_{33} que não foi testada. Aplicamos sobre ela $L_{10}P_{10}L_9P_9L_8P_8$ obtendo

$$(0, \ldots, 0, 1, \underline{2, 1/3, -1/3, -2/3})' .$$

O pivô escolhido é 2 e $P_{11} = I$, L_{11} é caracterizada por

$$(0, \ldots, 1, -1/6, 1/6, 1/3)' .$$

Assim, a única coluna de Φ_3 é $(0,0,1)'$ e a primeira coluna de U_4 é $(2,0,0,0)'$.

Para formar as colunas restantes de U_4 , testaremos as

colunas de B_{44}. Estas são modificadas somente por L_i e P_i se $i \geq 11$. Então aplicamos $L_{11} P_{11}$ sobre a primeira coluna de B_{44} e temos

$$(0, \ldots, 1, \underline{5/6, 1/6, 1/3})' .$$

O pivô escolhido é 5/6 e $P_{12} = I$. L_{12} é representada por

$$(0, \ldots, 0, 0, 1, -1/5, -2/5)' .$$

A segunda coluna de U_4 é $(1, 5/6, 0, 0)'$.

Aplicamos $L_{12}P_{12}L_{11}P_{11}$ sobre a segunda coluna de B_{44} obtendo

$$(0, \ldots, 0, 1, \underline{4/5, -2/5})'$$

O pivô escolhido é 4/5, $P_{13} = I$ é L_{13} é caracterizada por

$$(0, \ldots, 1, 1/2)'$$

A terceira coluna de U_4 é $(0, 1, 4/5, 0)'$.

Finalmente aplicamos $L_{13}P_{13}L_{12}P_{12}L_{11}P_{11}$ sobre a última coluna de B_{44} obtendo

$$(0, \ldots, 0, 0, 0, 1)' .$$

Fica determinada a última coluna de U_4 como $(0, 0, 0, 1)'$.

A matriz U_4 completa é

$$U_4 = \begin{vmatrix} 2 & 1 & 0 & 0 \\ & 5/6 & 1 & 0 \\ & & 4/5 & 0 \\ & & & 1 \end{vmatrix}$$

A matriz U está representada na figura I.14.

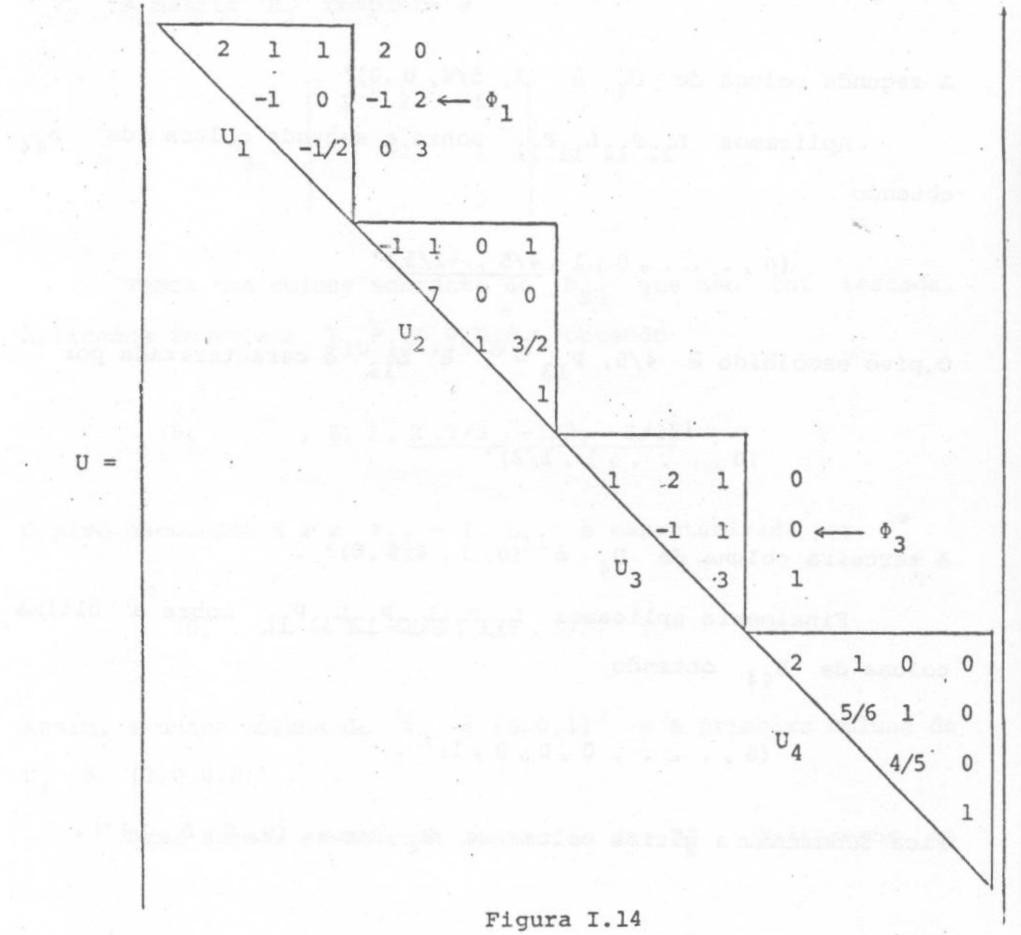
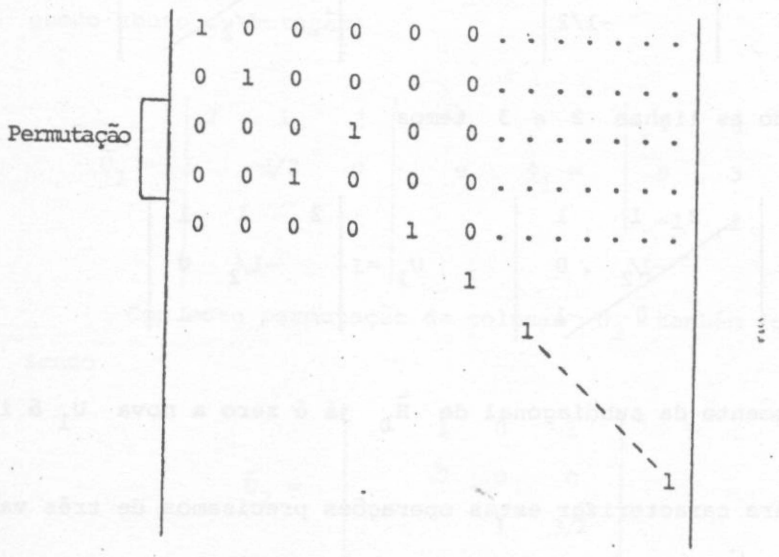


Figura I.14

A matriz Q tal que $L^{-1} B Q = U$ neste caso é



já que só houve permutação das colunas 3 e 4 de B.

Para ilustrar o processo de atualização da fatoração de B vamos supor que a segunda coluna de U sai da base e que no seu lugar deve entrar a seguinte coluna

$$\tilde{A}^s = (0, 0, 0, 1, -1, 1, 1, 1, 1, 1, 0, 0, 0, 0)'$$

A coluna de saída é a segunda de U_1

$$U_1 = \begin{vmatrix} 2 & 1 & 1 \\ & -1 & 0 \\ & & -1/2 \end{vmatrix} \Longrightarrow H_1 = \begin{vmatrix} 2 & 1 & 1 \\ & 0 & -1 \\ & -1/2 & \end{vmatrix}$$

permutando as linhas 2 e 3 temos

$$\tilde{H}_1 = \begin{vmatrix} 2 & 1 & 1 \\ & -1/2 & 0 \\ & 0 & -1 \end{vmatrix} \qquad U_1 = \begin{vmatrix} 2 & 1 & 1 \\ & -1/2 & 0 \\ & & -1 \end{vmatrix}$$

Com o elemento da subdiagonal de $\tilde{H}_1$ já é zero a nova U_1 é imediata.

Para caracterizar estas operações precisamos de três valores:

a) a posição da coluna de saída em U_i; neste caso é 2.

b) um parâmetro que indique se houve ou não permutação de linhas.

c) o fator que caracteriza a eliminação do elemento na subdiagonal. Neste caso 0.

Aplicamos as operações anteriores sobre Φ_1, obtendo

$$\Phi_1 = \begin{vmatrix} 2 & 0 \\ 0 & 3 \\ -1 & 2 \end{vmatrix}$$

A primeira coluna de Φ_1 tem um elemento não nulo na terceira linha. Então permutamos as colunas 3 e 4 de U. Obtemos (com pequeno abuso de notação)

$$\tilde{U}_1 = \begin{vmatrix} 2 & 1 & 2 \\ & -1/2 & 0 \\ & & -1 \\ & & -1 \end{vmatrix} \quad \text{e} \quad \Phi_1 = \begin{vmatrix} 1 & 0 \\ 0 & 3 \\ -1 & 2 \end{vmatrix}.$$

Com esta permutação de colunas U_2 também foi modificada sendo

$$\tilde{U}_2 = \begin{vmatrix} 0 & 1 & 0 & 1 \\ & 7 & 0 & 0 \\ & & 1 & 3/2 \\ & & & 1 \end{vmatrix}$$

Devemos triangular $\tilde{U}_1$. As operações para isto ficam caracterizadas por:

a) linha do pivô destas operações. Neste caso 3.

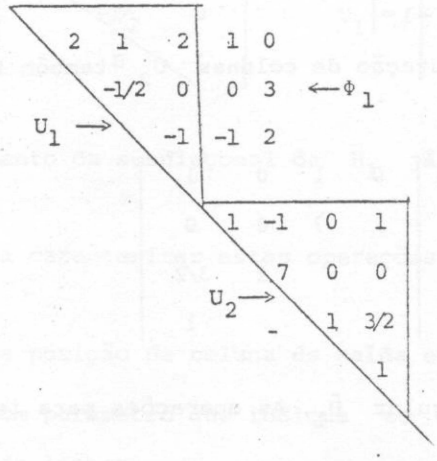
b) fatores característicos desta eliminação. Neste caso -1.

c) linhas a ser zeradas. Neste caso 4.

Aplicamos estas operações a $\tilde{U}_1$ e $\tilde{U}_2$ obtendo

$$U_1 = \begin{vmatrix} 2 & 1 & 2 \\ & -1/2 & 0 \\ & & -1 \end{vmatrix} \quad e \quad U_2 = \begin{vmatrix} 1 & -1 & 0 & 1 \\ & 7 & 0 & 0 \\ & & 1 & 3/2 \\ & & & 1 \end{vmatrix}$$

A configuração total das partes modificadas da matriz U é



Agora a coluna de saída está na posição da primeira coluna de U_2. Levamos esta coluna até a posição da quarta coluna de U_2, retrocedendo uma posição com as outras, obtendo H_2

$$H_2 = \begin{vmatrix} -1 & 0 & 1 & 1 \\ 7 & 0 & 0 & \\ & 1 & 3/2 & \\ & & 1 & \end{vmatrix}$$

Permutando as linhas 1 e 2 em H_2, temos

$$\tilde{H}_2 = \begin{vmatrix} 7 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 \\ & 1 & 3/2 & \\ & & 1 & \end{vmatrix}$$

Realizando a primeira eliminação, vem

$$\tilde{H}_2 = \begin{vmatrix} 7 & 0 & 0 & 0 \\ & 0 & 1 & 1 \\ & 1 & 3/2 & \\ & & 1 & \end{vmatrix}$$

Esta operação é representada por

a) fator 1/7

b) linha 5

Permutando as linhas 2 e 3 em U_2 temos

$$\tilde{\tilde{H}}_2 = \begin{vmatrix} 7 & 0 & 0 & 0 \\ & 1 & 3/2 & 0 \\ & 0 & 1 & 1 \\ & & 1 & \end{vmatrix}$$

Não havendo necessidade de eliminação a operação é representada por

a) fator 0

b) linha 6.

Finalmente obtemos a nova U_2,

$$U_2 = \begin{vmatrix} 7 & 0 & 0 & 0 \\ & 1 & 3/2 & 0 \\ & & 1 & 1 \\ & & & -1 \end{vmatrix}$$

Não permutamos linhas e a operação de eliminação é representada por

a) fator -1

b) linha 7

Como a matriz Φ_2 é nula não temos modificações adicionais em U, que após as operações apresenta a forma ilustrada na figura I.15.

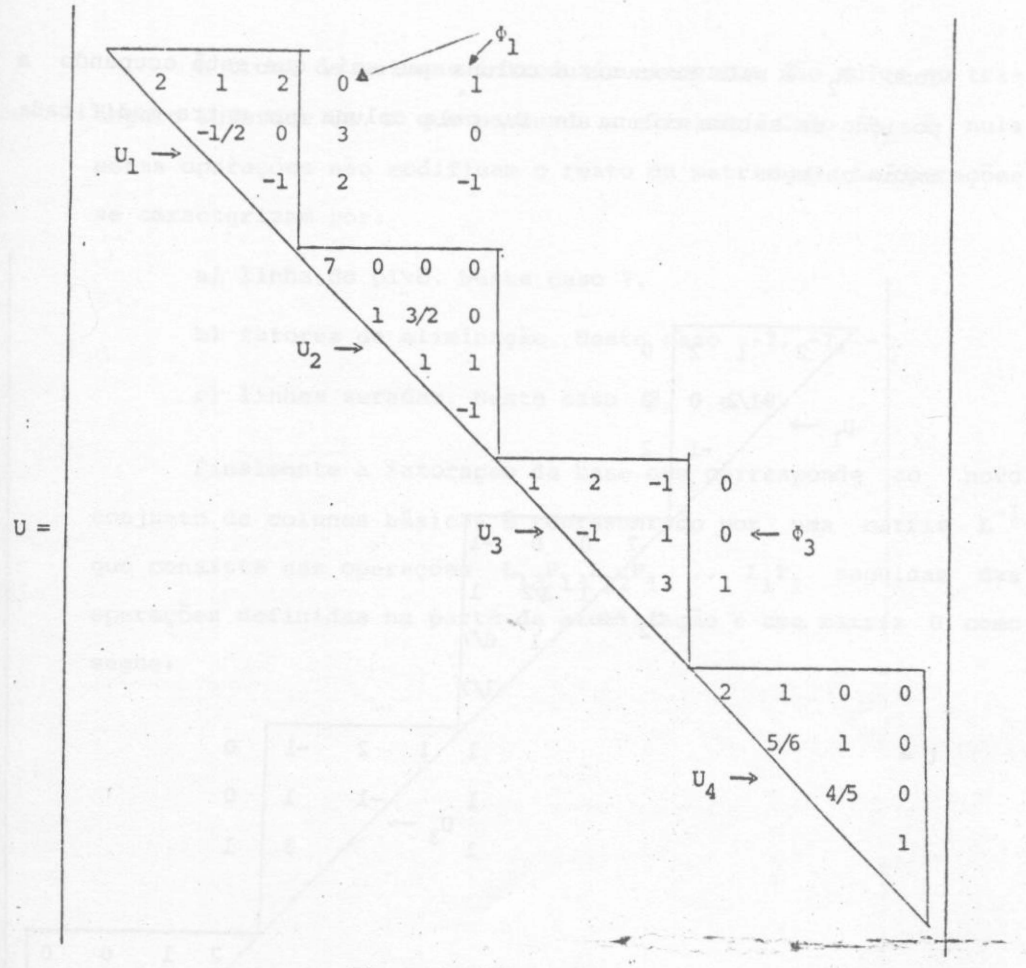


Figura I.15

Se aplicamos todas as operações efetuadas sobre U à coluna que entra na base obtemos

$$(0,0,0,-1,1,,6/7,1/7,1,1,1,0,0,0,0)'$$

Como Φ_2 é nula trocamos a coluna que sai, que está ocupando a posição da sétima coluna de U, pela coluna que entra modificada, Então obtemos

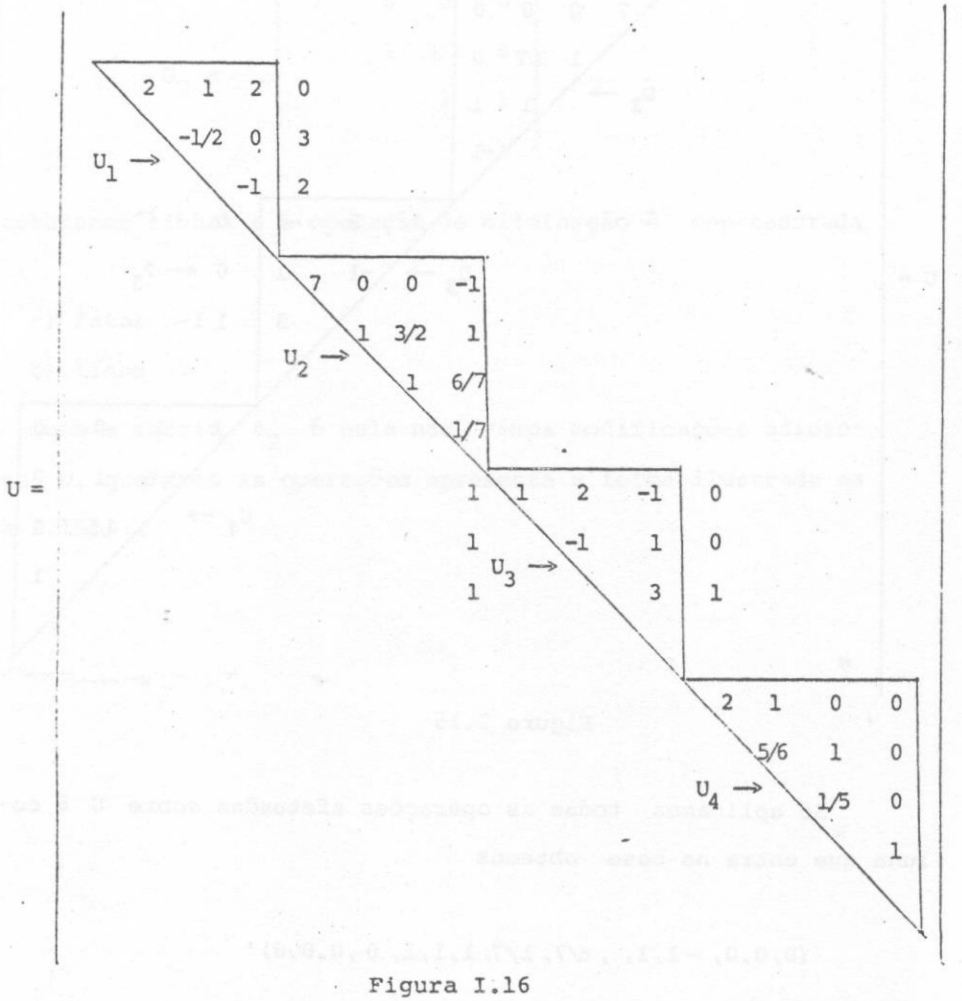


Figura I.16

A troca de colunas introduziu elementos não nulos no triângulo inferior de U que passamos a eliminar. Como Φ_2 é nula estas operações não modificam o resto da matriz U. As operações se caracterizam por:

a) linha do pivô. Neste caso 7.

b) fatores da eliminação. Neste caso -7, -7, -7.

c) linhas zeradas. Neste caso 8, 9 e 10.

Finalmente a fatoração da base que corresponde ao novo conjunto de colunas básicas é representado por uma matriz L^{-1} que consiste nas operações $L_{13}P_{13}L_{12}P_{12} \cdots L_1P_1$ seguidas das operações definidas na parte da atualização e uma matriz U como segue:

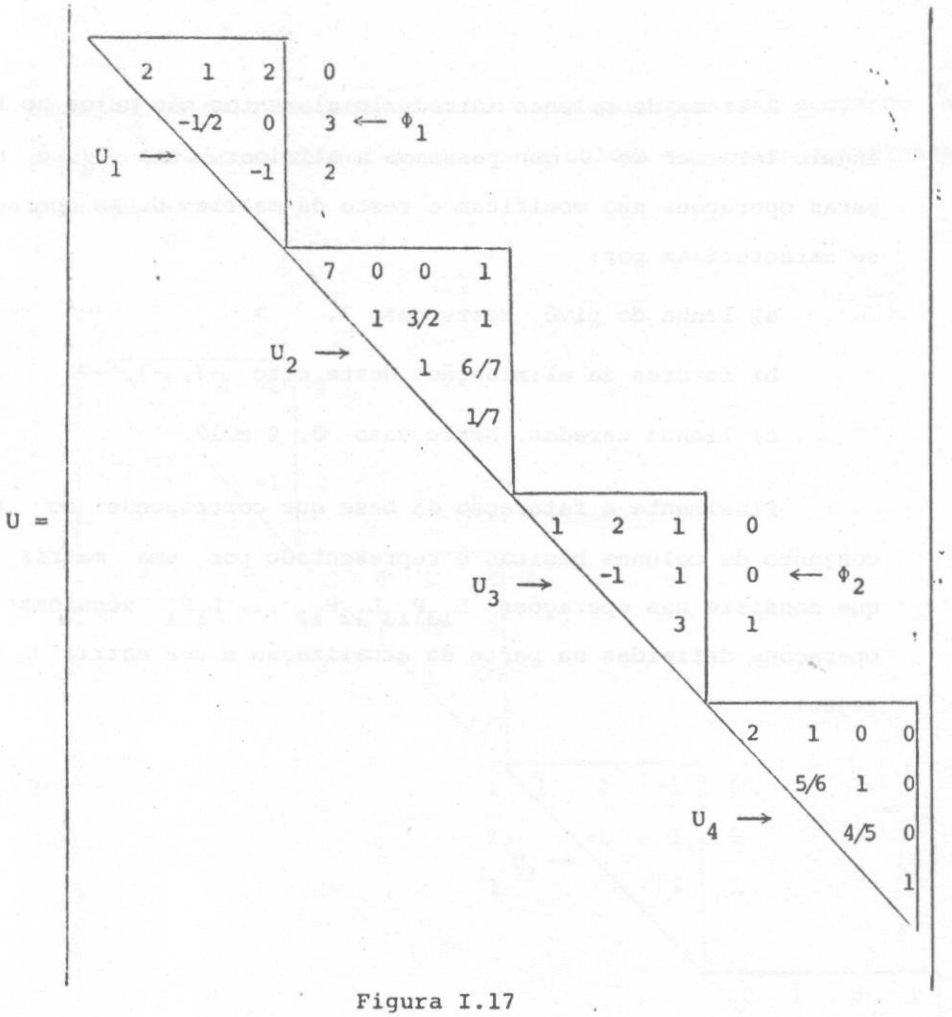


Figura I.17

Finalmente a matriz Q que representa todas as permuta - ções de colunas feitas sobre a B inicial é:

$$Q = \begin{vmatrix} 1 & 0 & 0 & 0 & \cdots & & & & & & & & & & \\ 0 & 0 & 0 & 1 & \cdots & & & & & & & & & & \\ 0 & 0 & 1 & 0 & \cdots & & & & & & & & & & \\ 0 & 0 & 0 & 0 & 1 & \cdots & & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 1 & \cdots & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots & & & & & & & \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \cdots & & & & & & \\ & & & & & & & & 1 & & & & & & \\ & & & & & & & & & 1 & & & & & \\ & & & & & & & & & & 1 & & & & \\ & & & & & & & & & & & \ddots & & & \\ & & & & & & & & & & & & 1 & & \end{vmatrix}$$

No capítulo seguinte veremos detalhadamente os aspectos computacionais relacionados com este algoritmo.

CAPÍTULO II

PROGRAMA COMPUTACIONAL PARA PROGRAMAÇÃO LINEAR DINÂMICA

1. INTRODUÇÃO

Neste capítulo consideramos o projeto e desenvolvimento de rotinas para cálculos matriciais envolvendo matrizes com estrutura escada, necessárias à implementação do algoritmo proposto no capítulo anterior.

Desenvolvemos um programa com 27 subrotinas. Uma delas SIMESC (Simplex escada) controla a chamada às restantes. A SIMESC é chamada uma única vez pelo programa principal onde são introduzidos os dados do problema.

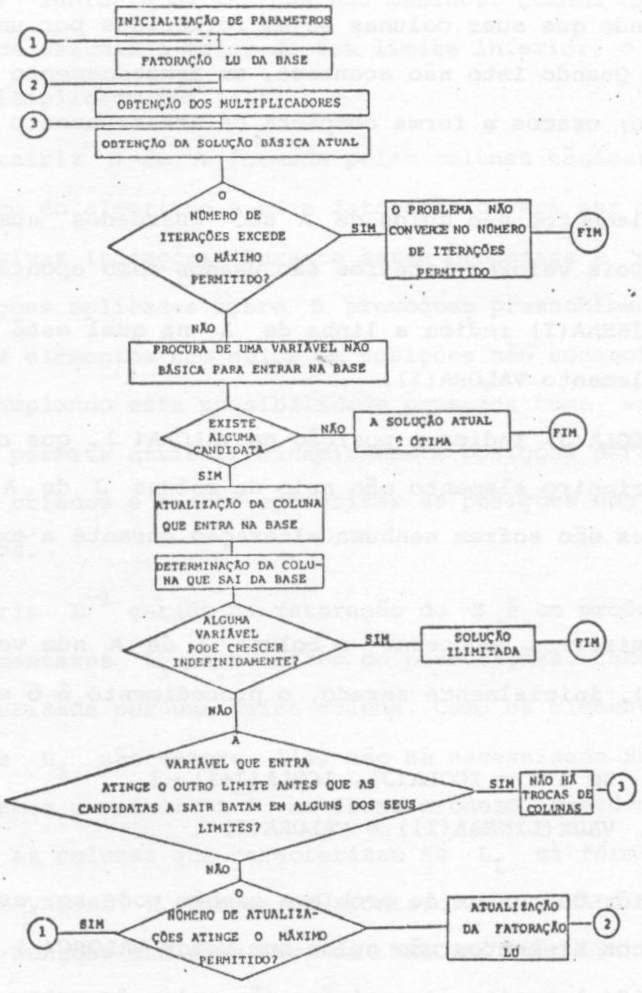
O programa implementado contempla a possibilidade da existência de atrasos tanto no vetor de estados quanto no de controles. Neste caso as equações dinâmicas do problema são:

$$x(t+1) = \sum_{j=0}^{NATRAS} \{A_j(t)x(t-j) + B_j(t)u(t-j)\}$$

onde NATRAS indica o máximo atraso. A única consequência da introdução dos atrasos é que as matrizes do problema têm o triângulo inferior mais denso (menos esparso). O método de resolução não sofre alterações.

2. DIAGRAMA DE BLOCOS DO ALGORITMO

O diagrama de blocos salientando os passos principais do algoritmo é apresentado a seguir:



3. TÉCNICAS DE ARMAZENAMENTO UTILIZADAS

As colunas da matriz A de restrições devem estar disponíveis para a implementação do processo iterativo do Método Simplex. Em muitos casos a matriz A obedece a uma lei de formação, possibilitando que suas colunas sejam fornecidas por uma rotina de geração. Quando isto não acontece, um armazenamento explícito é necessário; usamos a forma compacta de armazenamento de A por colunas.

Os elementos não nulos de A são guardados num vetor VALORA( ). Dois vetores inteiros são usados como apontadores:

a) LINHAA(I) indica a linha de A na qual está situado o elemento VALORA(I);

b) ICOLA(J) indica a posição de VALORA( ), que contém o primeiro elemento não nulo da coluna J de A.

Estes vetores não sofrem nenhuma alteração durante a execução do algoritmo.

Se quisermos recuperar a coluna J de A num vetor auxiliar VAUX( ), inicialmente zerado, o procedimento é o seguinte:

```
      DO 1 I = ICOLA(J), ICOLA(J+1) - 1
    1 VAUX(LINHAA(I)) = VALORA(I) .
```

O vetor de custos do problema também pode ser esparso e armazenamos os elementos não nulos num vetor VALORC( ). Um vetor inteiro ICOLC( ) aponta às posições não nulas do vetor de custos.

ICOLC(I) indica que posição ocupa o elemento VALORC(I) no vetor de custos. Analogamente definimos VALORV( ) e ICOLV( ) para o vetor de recursos do problema.

Um vetor INDB( ) de dimensão (m+n)·T, indicará a cada iteração o subconjunto de índices correspondentes às colunas básicas. INDNB( ) indicará os índices não básicos. Quando uma variável não básica assumir o valor do seu limite inferior, o índice dela será multiplicado por (-1).

A submatriz B de A formada pelas colunas básicas é fatorada no começo do algoritmo e esta fatoração deverá ser atualizada nas sucessivas iterações. Durante estes processos é provável que as operações aplicadas sobre B provoquem preenchimento ou eliminação de elementos não nulos em posições não conhecidas a priori. Contemplando esta possibilidade pensamos numa estrutura de dados que permita atribuir dinamicamente posições para os elementos novos criados e também aproveitar as posições dos elementos eliminados.

A matriz L^{-1} gerada na fatoração de B é um produto de matrizes elementares L_i, e matrizes de permutações. Cada matriz L_i é caracterizada por uma única coluna. Como os elementos das diagonais das L_i são sempre 1's, não há necessidade de guardá-los. Armazenamos a matriz L^{-1} na forma produto, guardando sucessivamente as colunas que caracterizam as L_i na forma compacta num vetor VALORL( ). Usamos os apontadores LINHAL( ) e ICOLL( ) com funções análogas as de VALORA( ), LINHAA( ) e

ICOLA( ) em A. Quando as fatorações são modificadas as novas matrizes elementares são arquivadas na continuação das anteriores.

Os vetores VALORL( ) e LINHAL( ) tem um contador, KL, que aponta sempre para a primeira posição livre neles. Um contador KICOL é usado para o mesmo fim em relação a ICOLL( ).

Um vetor IPERM( ), indicará se houve permutação de linhas no processo de fatoração. Se IPERM(I) = I para I = 1,2,...,(m+n).T-1, significa que não foi necessária nenhuma permutação de linhas. Se IPERM(I) = J significa que a linha I foi permutada com a linha J. As posições de IPERM( ) situadas após a (m+n).T-ésima posição são usadas para:

a) determinar se houve permutação de linhas durante as eliminações das matrizes H_i ;

b) Verificar se a L_i a ser considerada corresponde a:

- triangularização de uma matriz H_i ;
- eliminações feitas após permutar colunas de U_i e U_j com $i \neq j$;
- eliminações feitas após trocar definitivamente a coluna que sai da base pela coluna que entra na iteração atual.

A forma como as informações acima são registradas é ilustrada no exemplo apresentado no próximo ítem. Para IPERM( ) temos um contador KIPER.

O vetor IBALO( ) registrará as permutações de colunas.

Se IBALO(I) = J, significa que a coluna que inicialmente estava no J-ésimo lugar em B, encontra-se agora no I-ésimo lugar. Se quisermos saber qual é o índice em A da I-ésima coluna de B fazemos:

J = IBALO(I)

K = INDB(J)

K é o índice procurado.

Como foi visto no capítulo anterior, a matriz U tem a configuração representada na figura II.1.

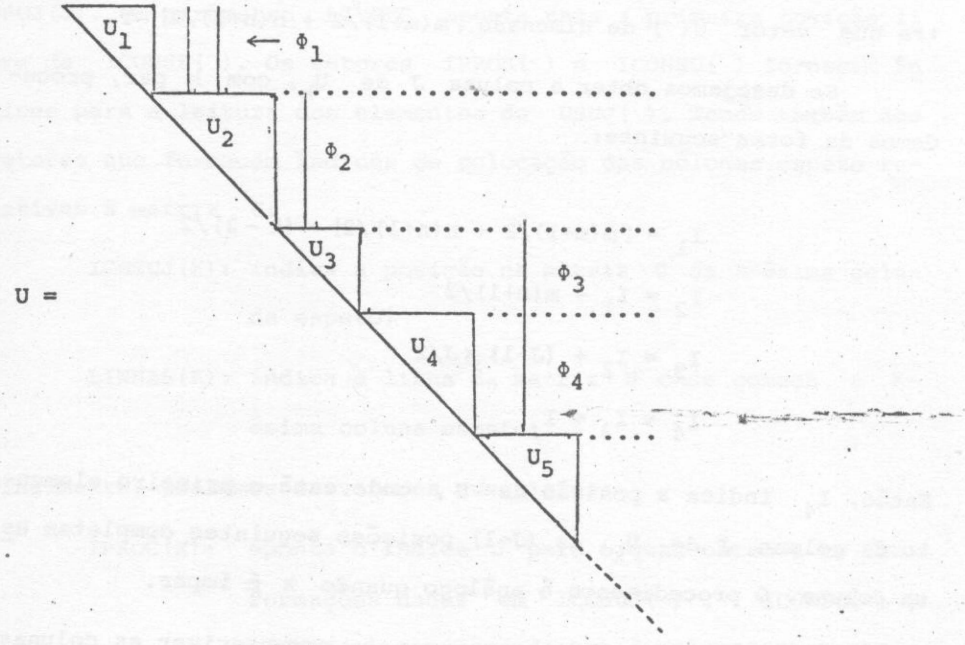


Figura II.1.

isto é, U é constituída de matrizes triangulares superiores $U_1, U_2, \ldots, U_{2T}$, com dimensões m,n,m,n,...,m,n respectivamente, e de algumas colunas isoladas que são as colunas sobrantes. A partir de agora usaremos para estas colunas a denominação de colunas 'espeto' (spikes). Lembramos que a parte das colunas espeto cujas linhas são as correspondentes às linhas de U_i, constituem a matriz Φ_i. Para armazenar a matriz U guardamos separadamente as partes correspondentes às matrizes U_i (i = 1,2,...,2T) incluindo os zeros, e as partes correspondentes às colunas espeto. As matrizes U_i são armazenadas por colunas, uma após a outra num vetor U( ) de dimensão [m(m+1)/2 + n(n+1)/2]·T.

Se desejamos obter a coluna J de U_k, com k par, procedemos da forma seguinte:

$$I_1 = [m(m+1)/2 + n(n+1)/2] \times (k-2)/2$$

$$I_2 = I_1 + m(m+1)/2$$

$$I_3 = I_2 + (J-1) \times J/2$$

$$I_4 = I_3 + 1$$

Então, I_4 indica a posição de U, onde está o primeiro elemento da coluna J de U_k. As (J-1) posições seguintes completam essa coluna. O procedimento é análogo quando k é ímpar.

Precisamos de vários vetores para caracterizar as colunas espeto. O primeiro deles, USUJ( ), contêm os valores de cada coluna, incluindo os zeros, a partir da primeira linha da Φ_i onde

a coluna começa. O vetor IPROS( ), indica em que posição de USUJ( ) está o elemento próximo daquele que estamos apontando. Se IPROS(I) = J, significa que o elemento situado na linha seguinte à linha correspondente a USUJ(I) está armazenado em USUJ(J). (Não necessariamente é J = I + 1). Quando apontamos ao último elemento de uma coluna espeto, digamos USUJ(K), definimos IPROS(K)=0 para indicar que a coluna acabou. A primeira posição livre em USUJ( ) é registrada numa variável chamada LIVRES. Um vetor ICONSU( ) aponta para as posições em USUJ( ) onde começam as colunas espeto. Por exemplo, se ICONSU(K) = J, significa que a K-ésima coluna espeto tem seu primeiro elemento armazenado em USUJ(J). Um parâmetro LIVREC aponta para a primeira posição livre de ICONSU( ). Os vetores IPROS( ) e ICONSU( ) fornecem índices para a leitura dos elementos de USUJ( ). Temos também dos vetores que fornecem índices de colocação das colunas espeto relativas à matriz U:

ICOSUJ(K): indica a posição na matriz U da K-ésima coluna espeto;

LINHAS(K): indica a linha da matriz U onde começa a K-ésima coluna espeto;

Finalmente, usaremos os vetores auxiliares:

IPROC(K): aponta o índice J para o qual obtemos as informações dadas em ICOSUJ( ), ICONSU( ), LINHAS( ) correspondentes à coluna espeto mais

próxima a direita da K-ésima. Se temos I colunas espeto no total definimos IPROC(I) = 0.

IANT(K): análogo ao vetor IPROC( ), aponta ao índice correspondente à coluna espeto mais próxima a esquerda da K-ésima.

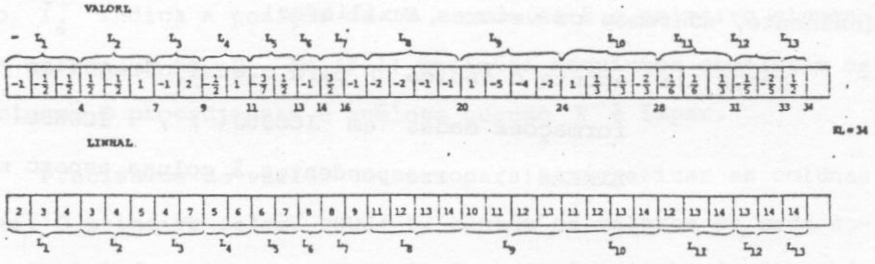
O parâmetro LIVREC, definido antes, também aponta para a posição livre destes vetores.

IPRIMC é uma variável que indica em que posição dos vetores definidos acima, está a informação da primeira coluna espeto de U. IULT tem significado análogo a IPRIMC, para a última coluna espeto de U.

4. EXEMPLO NUMÉRICO

Mostraremos aqui como alguns vetores descritos no ítem anterior são montados para o exemplo do Capítulo I.

Inicialmente, veremos a representação da matriz L^{-1}, armazenada na forma produto, através das matrizes elementares L_i.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ICOLL	1	4	7	9	11	13	14	16	20	24	28	31	33	34	

KICOL = 15

Os vetores IPERM( ) e IBALO( ) resultam:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
IPERM	1	2	3	5	7	7	7	10	9	10	11	12	13	

KIPER = 14

IBALO	1	2	4	3	5	6	7	8	9	10	11	12	13	14

Finalmente, apresentamos os vetores que representam a matriz U. No vetor U( ) colocamos sucessivamente as colunas das U_i , obtemos:

U

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
2	1	-1	1	0	$-\frac{1}{2}$	-1	1	7	0	0	1	1	0	$\frac{3}{2}$	1	1	2	-1	1	1	3	2	1	$\frac{5}{6}$	0	1	$\frac{4}{5}$	0	0	0	1

USUJ

1	2	3	4	5	6	7	8	9	10
2	-1	0	0	2	3	0	0	1	

LIVRES = 10

IPROS

2	3	0	5	6	0	8	9	0	

	1	2	3	4
ICONSU	1	4	7	

IPRIMC = 1

IULT = 3

LIVREC = 4

	1	2	3	4
ICOSUJ	4	5	11	
LINHAS	1	1	8	
IPROC	2	3	0	
IANT	0	1	2	

Os vetores e demais parâmetros apresentados acima são obtidos na fatoração inicial de matriz básica.

Vejamos agora como as informações dadas nesses vetores podem ser utilizadas. Se quizermos, por exemplo, aplicar as operações associadas a L_{10}, precisamos localizar os elementos que a

definem no vetor VALORL( ). Para isso usamos as seguintes informações do vetor ICOLL( );

$$ICOLL(10) = 24 \quad e \quad ICOLL(11) = 28 .$$

Deduzimos que existem 4 valores não nulos que caracterizam L_{10}. Esses valores estão em VALORL( ) a partir da posição 24. São:

24	25	26	27
1	$\frac{1}{3}$	$-\frac{1}{3}$	$-\frac{2}{3}$

As posições que ocupam estes elementos no vetor que caracteriza L_{10} estão em LINHAL( ) a partir da posição 24,

24	25	26	27
11	12	13	14

Vejamos agora como reconstruir colunas da matriz U.

Por exemplo, se procuramos a terceira coluna de U_3, representada na figura I.13, fazemos inic[illegible]

$$I_1 = \{3 \times (3+1)/2 + 4 \times (4+1)/2\} \times (3-1)/2 = 16$$

$$I_2 = 16 + (3-1) \times 3/2 = 19$$

$$I_3 = 19 + 1 = 20.$$

Então na posição 20 do vetor U, temos o primeiro elemento da coluna procurada, que por ser a terceira coluna duma U_i , neste caso U_3, consta de três elementos, que são:

20	21	22
1	1	3

Exemplificamos a seguir como reconstruir a segunda coluna espeto.

Começamos observando que IPRIMC = 1, o que significa que os dados da primeira coluna espeto estão na posição 1 dos vetores associados a este tipo de coluna. Portanto, IPROC(1) aponta ao índice que nos interessa. Neste caso IPROC(1) = 2, o que significa que as informações correspondentes à coluna desejada estão na posição 2 dos vetores que a caracterizam:

ICONSU(2) = 4 indica que os elementos da 2ª coluna espeto estão guardados em USUJ( ), a partir da posição 4.

USUJ(4) = 0.

Para sabermos a posição do elemento seguinte consultamos o vetor IPROS( ). Assim,

IPROS(4) = 5

diz que esse elemento é

USUJ(5) = 2.

Analogamente

IPROS(5) = 6

diz que o elemento seguinte a USUJ(5) é

USUJ(6) = 3

e

IPROS(6) = 0

diz que a coluna espeto acabou.

As demais características desta coluna espeto são dadas por:

ICOSUJ(2) = 5

diz que a 2^{a} coluna espeto é parte da 5^{a} coluna de U.

LINHAS(2) = 1

diz que o primeiro elemento desta coluna, USUJ(4) = 0, está na primeira linha de U

IANT(2) = 1

diz que os dados da coluna espeto imediatamente anterior a esta coluna estão em IPROC(1), ICOSUJ(1), etc;

IPROC(2) = 3

dá as mesmas informações que IANT( ), mas para a coluna espeto imediatamente posterior. Com isto temos os dados necessários para a coluna espeto exemplificada.

Analisamos agora as informações fornecidas pelos vetores IPERM( ) e IBALO( ).

IPERM(5) = 7

diz que P_5 consiste em permutar as linhas 5 e 7.

IPERM(6) = 7

diz que P_6 é a operação de permutação das linhas 6 e 7.

Como IPERM(8) = 10 temos que P_8 consiste em permutar as linhas 8 e 10.

IPERM(7) = 7

diz que P_7 = I.

Observe-se, que de forma análoga a P_7, as demais posições de IPERM( ) indicam que não houve outras permutações de linhas.

Vamos agora supor que o conjunto de índices básicos é

INDB =	1	2	4	5	7	10	11	13	15	16	18	20	21	22

então

IBALO(3) = 4

diz que a terceira coluna da base é INDB(4) = 5, ou seja a 5ª de A.

Como IBALO(4) = 3

é

INDB(3) = 4

temos que a 4ª coluna da base é a quarta de A.

Para as outras posições de IBALO( ) temos

IBALO(J) = J

significando que a J-ésima coluna da base corresponde a posição em A dada por INDB(J).

Vamos exemplificar como são registradas as mudanças nos vetores quando a fatoração é atualizada. Continuemos com o exemplo do Capítulo I.

Enumeramos a seguir as operações feitas naquele exemplo durante o processo de atualização.

1) Permutação das colunas 2 e 3 de U_1 obtendo-se H_1.

2) Permutação das linhas 2 e 3 (escolha do pivô).

3) Pivoteamento com

- fator 0
- linha zerada 3.

4) Permutação da coluna 3 de U_1 com a 1 de U_2.

5) Pivoteamento com
   - linha de pivô 3
   - fator -1
   - linha zerada 4.

6) Permutações de U_2 que resultam em H_2.
   Coluna 1 de U_2 (4 de U) passa para 4 de U_2 (7 de U) e as colunas 2,3,4 de U_2 (5,6,7 de U) passam para 1,2, 3 de U_2 (4,5,6 de U).

7) Permutação das linhas 4 e 5 (escolha de pivô).

8) Pivoteamento com
   - fator 1/7
   - linha zerada 5.

9) Permutação das linhas 5 e 6 (escolha do pivô).

10) Pivoteamento com
    - fator 0
    - linha zerada 6.

11) Pivoteamento sem permutação prévia
    - fator -1
    - linha zerada 7.

12) Pivoteamento após troca definitiva da coluna que sai pela que entra na base
    - linha do pivô 7

- fatores -7, -7, -7
- linhas zeradas 8, 9, 10.

Antes de começar com as operações definimos alguns parâmetros que serão usados:

M = 3 , N = 4 - dimensões das U_i com i ímpar e i par, respectivamente.

ICOS = 2 - posição da coluna que sai na matriz básica.

IRBL = 2 - posição relativa da coluna que sai em U_i . Neste caso 2^{a} de U_1.

Podemos calcular a linha na matriz U que corresponde à última linha de U_1

$$IUBL = ICOS + M - IRBL = 2 + 3 - 2 = 3.$$

Como estamos em U_1 é IRBL = 2 o primeiro elemento que será pivô na triangularização de H_1 estará na posição no vetor U dada por

$$IPIV = IRBL * (IRBL + 1)/2 + IRBL .$$

Neste caso $IPIV = 2 \times 3/2 + 2 = 5$.

Este elemento está na linha ICOS = 2 da matriz U. Definimos LIN = 2.

Agora podemos começar as operações nesta ordem.

Comparamos $|U(IPIV)|$ e $|U(IPIV + 1)|$.

Neste caso $|U(5)| = 0 < |U(6)| = \frac{1}{2}$ portanto permutamos

$$D = U(IPIV)$$

$$U(IPIV) = U(IPIV + 1)$$

$$U(IPIV + 1) = D .$$

Agora o vetor U( ) ficou:

$$U = (2,\ 1,\ -1,\ 1,\ -\tfrac{1}{2},\ 0,\ 1\ , \ldots\ldots\ldots\ldots)$$

(1 2 3 4 5 6 7 → segue como antes)

Esta permutação a registramos fazendo

$$IPERM(14) = -1$$

e fizemos a operação 2).

O pivoteamento de 3) vem dado pelo fator

$$-U(IPIV + 1)/U(IPIV)$$

Neste caso $-U(6)/U(5) = -0/5 = 0$.

A linha zerada é $LIN + 1 = 3$.

(Esta eliminação apesar de ter fator 0, é registrada, pois não estamos considerando a esparsidade dentro das U_i).

Registramos estas informações em

$$VALORL(34) = 0$$

$$LINHAL(34) = 3.$$

A linha do pivô das L_i correspondentes ao processo de Bartels-Golub é sempre a linha que está imediatamente acima da linha zerada. Os dados registrados definem L_{14} e L_{15} será armazenada a partir de KL = 35, fazemos então ICOLL(15) = 35.

Como estamos operando sobre a última coluna de U_1 definimos

IPERM(15) = IUBL = 3

que avisa pelo fato de ser $3 \neq -1$ e $3 \neq 0$ que a triangularização de H_1 acabou e da a linha do pivô da próxima L_i, ou seja L_{15}. Devemos aplicar P_{14} sobre a coluna que sai da base, para isso a colocamos num vetor auxiliar VAUX( ), ocupando as duas primeiras posições, zerando as posições restantes

$$\text{VAUX} = (\underset{\downarrow}{1}\ ,\ \underset{\downarrow}{-1}\ ,0,0,\ldots\ldots)$$

(U(2) sobre 1, U(3) sobre -1)

Aplicamos P_{14} sobre VAUX( ) obtendo:

$$\text{VAUX} = (1\ ,0\ ,-1\ ,0,\ \ldots\ )$$

Agora passamos a realizar as operações de 1) sem perigo de perder informação. Retrocedemos com os valores U(4) e U(5) para as posições 2 e 3 de U( ) obtendo

$$\begin{array}{l} 1\ \ 2\ \ \ 3\ \ 4\ \ \ 5\ \ 6 \longrightarrow \text{segue como antes} \\ U = (2,\ 1,\ -\frac{1}{2}\ ,1,\ -\frac{1}{2}\ ,0\ ,\ldots\ldots\ldots) \end{array}$$

Registramos esta permutação de colunas fazendo

IBALO(2) = 3

IBALO(3) = 2 .

Agora aplicamos a permutação de linhas e L_{14} sobre Φ_1 que consta das duas primeiras colunas espeto, que estão armazenadas em USUJ( ) que fica como segue:

USUJ = (2, 0, -1, 0, 3, 2, 0, 0, 1)

e não há necessidade de outras modificações nos vetores o parâmetros correspondentes às colunas espeto. A primeira coluna de Φ_1 tem um elemento não nulo na linha IUBL = 3 fazendo com que a coluna que sai, que deveria estar na 3ª coluna de U, seja permutada com esta primeira coluna espeto que está na quarta coluna de U. Este dado está em ICOSUJ(1) = 4. Esta permutação (operação 4) é registrada fazendo:

IBALO(3) = 4

IBALO(4) = 2 .

A coluna que sai passará agora a ocupar o lugar da 1ª coluna de U_2. Fazemos então IRBL = 1.

A 1ª coluna de U_2 possui um único elemento não nulo

abaixo da linha IUBL = 3, que desejamos eliminar antes de levar esta coluna para U_1. O pivô desta última eliminação está na posição 3 de USUJ( ), ou seja é

USUJ(3) = -1.

O elemento a ser zerado é o primeiro de U_2 cuja posição em U( ) é obtida assim:

M × (M+1)/2 + 1 = 3 × 4/2 + 1 = 7.

Então o fator desta eliminação é:

-U(7)/USUJ(3) = -(-1/-1)= -1.

e registramos

VALORL(35) = -1

LINHAL(35) = 4 .

Estes valores junto com IPERM(15) = 3 caracterizam L_{15}. L_{16} começará então na posição dada por ICOLL(16) = 36.

A eliminação L_{15} deve ser aplicada sobre todas as colunas espeto a direita da 1ª coluna de U_2 que tenham um elemento não nulo na linha IUBL = 3. Varrendo IPROC( ) e usando as informações de LINHAS( ), ICONSU( ), USUJ( ) e IPROS( ) detectamos todas essas colunas. Neste caso só temos uma que é a 5ª coluna

da matriz U ($2^{\underline{a}}$ coluna espeto). O valor nesta coluna correspondente à linha do pivô é USUJ(6) = 2. Aplicar L_{15} a esta coluna consiste em fazer:

$$U(8) = U(8) + VALORL(35) \times USUJ(6)$$

$$U(8) = 1 + (-1) \times 2 = -1 .$$

O vetor U( ) ficou:

$$U = (2,\ 1,\ -\tfrac{1}{2},\ 1,\ -\tfrac{1}{2},\ 0,\ 1,\ -1,\ 7,\ \ldots\ldots\ldots\ldots)$$

1 2 3 4 5 6 7 8 9 → segue como antes

Aplicamos agora L_{15} à coluna de saída que está em VAUX, obtendo:

$$VAUX = (1,0,-1,1,0\ ,\ldots\ldots).$$

Passamos agora a rearrumar os vetores e redefinir alguns parâmetros antes de continuar com as operações que seguem. Estas operações correspondem a um novo processo do tipo Bartels-Golub em U_2. A coluna que sai estará na matriz U na posição dada agora por ICOS = 4. Ela ocupa a primeira coluna de U_2, portanto IRBL = 1. As posições 1, 2, 3 de USUJ( ) passam a ocupar as posições 4, 5, 6 de U( ); temos então:

$$U = (2,\ -1,\ -\tfrac{1}{2},\ 2,\ 0,\ -1,\ 1,\ -1,\ 7,\ \ldots\ldots\ldots\ldots)$$

1 2 3 4 5 6 7 8 9 → segue como antes

As posições 1, 2, 3 de VAUX( ) passam a ocupar as posições 1, 2, 3 de USUJ( ):

USUJ = (1, 0, -1, 0, 3, 2, 0, 0, 1).

Os outros vetores e parâmetros não são modificados.

O novo valor IUBL é obtido como:

IUBL = ICOS + N - IRBL = 4 + 4 - 1 = 7

e dá a linha da matriz U na qual está a última linha de U_2.

O processo para eliminar a matriz H_2 é análogo ao exposto para H_1, portanto, não o repetimos em detalhe aqui.

As operações correspondentes ficam registradas nos vetores associados como segue:

IPERM(16) = -1 , VALORL(36) = $\frac{1}{7}$, LINHAL(36) = 5, ICOLL(17) = 37
IPERM(17) = -1 , VALORL(37) = 0 , LINHAL(37) = 6 , ICOLL(18) = 38
IPERM(18) = 0 , VALORL(38) = -1 , LINHAL(38) = 7 , ICOLL(19) = 39
IPERM(19) = 7 .

O vetor U( ) resultante após as permutações que dão H_2 e as operações registradas acima é:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ⟶

U = (2, 1, -1, 2, 0, -1, 7, 0, 1, 0, $\frac{3}{2}$, 1, 0, 0, 1, -1, ...)

As permutações de colunas feitas estão representadas em:

IBALO(4) = 5

IBALO(5) = 6

IBALO(6) = 7.

Como a coluna que sai estará colocada na posição da última coluna de U_2 teremos

IBALO(7) = 2.

Como acima da submatriz U_2 , existem colunas espeto, estas permutações devem modificar alguns dos vetores que as representam.

Neste caso basta redefinir

ICONSU = (4, 1, 7)

ICOSUJ = (4, 7, 11).

Neste exemplo a matriz Φ_2 é nula e podemos trocar definitivamente a coluna que sai da base pela que entra. A coluna que entra, guardada num vetor auxiliar VAUX(2), devidamente modificada por todas as operações feitas até aqui é:

$$\begin{matrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \text{VAUX2} = (& 0, & 0, & 0, & -1, & 1, & \frac{6}{7}, & \frac{1}{7}, & 1, & 1, & 1, & 0, & 0, & 0, & 0). \end{matrix}$$

Os elementos VAUX2(8), VAUX2(9) e VAUX2(10) devem ser zerados se pretendemos colocar este vetor no lugar da coluna que sai. Registramos agora esta eliminação fazendo:

VALORL(39) = -7 LINHAL(39) = 8

VALORL(40) = -7 LINHAL(40) = 9

VALORL(41) = -7 LINHAL(41) = 10

e definimos ICOLL(20) = 42.

A linha do pivô já está em IPERM(19) = 7.

As posições 4, 5, 6 e 7 de VAUX2 passam a formar a última coluna de U_2 portanto ocuparão em U( ) as posições 13, 14, 15 e 16, temos:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

U (2, 1, -1, 2, 0, -1, 7, 0, 1, 0, $\frac{3}{2}$, 1, -1, 1, $\frac{6}{7}$, $\frac{1}{7}$ ⟶)

Como as posições 1, 2 e 3 de VAUX2 são nulos, perdemos nesta troca uma coluna espeto. Para registrar esta perda fazemos

IPROC = (3, 0, 0)

IANT = (0, 0, 1)

e definimos LIVREC = 2.

Isto significa que os elementos que estão na 2ª posição dos vetores ICONSU( ), ICOSUJ( ), LINHAS( ), IPROC( ) e IANT( )

não interessam mais e podemos usar essa posição para armazenar dados de alguma nova coluna espeto que possa surgir numa outra iteração. Analogamente as posições 1,2,3 de USUJ( ) e IPROC( ) não nos interessam mais e fazemos LIVRES = 1.

Para ilustrar como registraríamos a introdução de uma coluna espeto nova, suponhamos que a matriz U atualizada com essa nova coluna corresponde à matriz apresentada na Figura II.2.

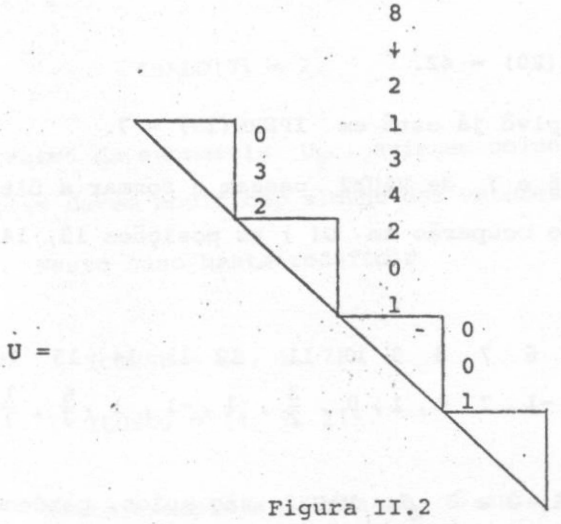


Figura II.2

Isto é, uma coluna espeto foi criada na coluna 8 de U, a partir da primeira linha. Os valores dessa coluna espeto devem ser guardados em USUJ( ). Em relação a USUJ( ), temos que LIVRES = 1, portanto usaremos inicialmente essa posição. Os vetores USUJ( ) e IPROS( ) atualizados são:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
USUJ	2	1	3	0	3	2	0	0	1	4	2	0	1	

IPROS	2	3	10	5	6	0	8	9	0	11	12	13	0	

tem-se também que LIVRES = 14.

Note-se que os elementos da nova coluna espeto não estão todos em sequência em USUJ( ). Quando chegamos a USUJ(3) temos que IPROS(3) = 10 é a coluna contínua na posição 10 de USUJ( ).

Para os demais vetores e parâmetros, tem-se

IPRIMC = 1 IULT = 3 .

Como LIVREC = 2 usamos a posição 2 dos vetores restantes para caracterizar esta coluna. Então, fazemos

ICONSU	4	1	7

ICOSUJ	4	8	11

LINHAS	1	1	8

IPROC	2	3	0

IANT	0	1	2

definindo em seguida

LIVREC = 4.

5. O PROGRAMA

5.1. FATORAÇÃO DA BASE

O processo de fatoração de B é realizado através de cinco subrotinas. Detalhamos aqui os passos essenciais de cada uma delas.

a) DELUST (chamada pela subrotina SIMESC).

Para $j = 1,2,\ldots,2T$, esta subrotina faz as seguintes operações:

i) Determina o número de colunas básicas candidatas a formação de U_j ;

ii) Para cada coluna entre as obtidas em i) cria os elementos correspondentes a L^{-1} e U. Para isso chama a subrotina CRIALU ;

iii) Determina quais são as colunas sobrantes entre as determinadas em i) quando U_j foi completada, caso elas existam;

iv) Quando uma coluna sobrante é aceita para formar parte de U_j , chama a subrotina CRIASU para gerar os elementos dos vetores que caracterizam as colunas espeto ;

v) Define os parâmetros LIVRES, LIVREC, KICOL, KL, KIPER, IPRIMC e IULT.

b) CRIALU (chamada pela subrotina DELUST)

Realiza as seguintes operações:

i) Quando a coluna testada é sobrante aplica sobre ela as L_k anteriores que a modificam. (Não necessariamente são todas). As operações feitas neste passo não usam as posições estruturalmente nulas, com exceção das que estão nas colunas espeto ;

ii) Chama a subrotina AMAXT para escolher o pivô entre os elementos da coluna correspondente as linhas de U_j ;

iii) Quando nenhum elemento dos pesquisados em ii) é aceito como pivô, a coluna é rejeitada. Se ela não for já uma coluna sobrante, é acrescentada a este conjunto de colunas;

iv) Se a coluna é aceita cria os elementos de VALORL( ), ICOLL( ), LINHAL( ) e U( ) correspondentes ;

v) Define o vetor IBALO( ).

c) CRIASU (chamada também pela DELUST).

Esta rotina cria os elementos que caracterizam as colunas espeto.

d) AMAXT (chamada pela subrotina CRIALU).

Escolhe o pivô para cada coluna ou avisa que a coluna foi rejeitada.

e) CRIANT (chamada pela SIMESC).

Cria o vetor IANT( ).

Um parâmetro EXSUJ toma valor 1 se existem colunas espe to no fim da fatoração da matriz B. Caso contrário EXSUJ = 0.

5.2. OBTENÇÃO DOS MULTIPLICADORES

O vetor de multiplicadores VMULT( ) é obtido através da resolução do sistema (1).

(1) $$B'VMULT = C_B$$

onde C_B é o vetor de custos básicos.

Como B esá fatorada na forma

$$BQ = LU$$

e

$$Q' = Q$$

temos

$$QB' = U'L'$$

então

$$QB'VMULT = QC_B$$

e

$$U'L'VMULT = QC_B .$$

Assim, (1) pode ser reduzidó à resolução de (2) e (3)

(2) $\qquad U'y = QC_B$

(3) $\qquad L'VMULT = y \quad$ ou $\quad VMULT = (L^{-1})'y$.

No programa implementado QC_B é obtido, aplicando sobre C_B as permutações registradas no vetor IBALO( ). Três subrotinas são utilizadas na resolução de (2) e (3): MULT1, MULT12 e UTRINF.

a) MULT1 (chamada pela SIMESC).

Esta subrotina resolve o sistema (2).

A matriz associada ao sistema (2), U' , é triangular inferior com a configuração na figura II.3.

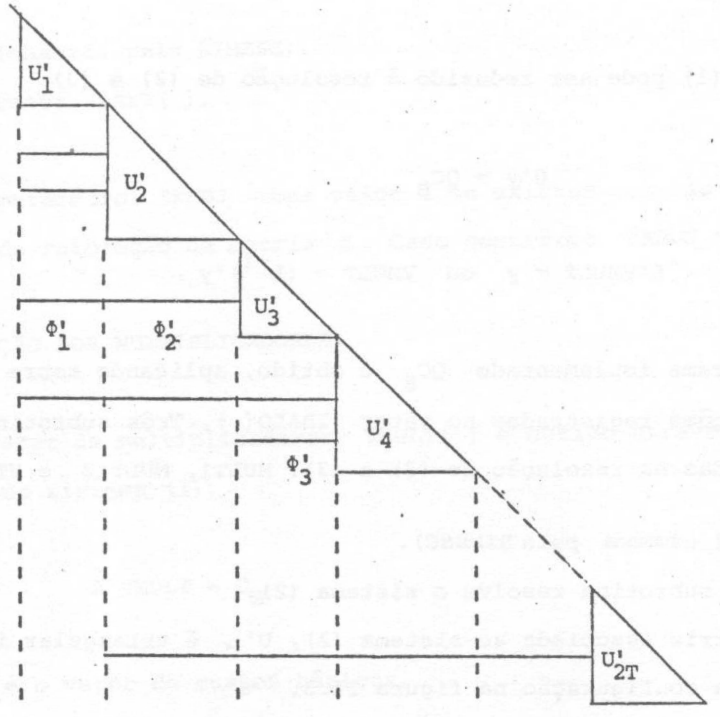


Figura II.3

O esquema para resolver um sistema $U'y = C_B$ com esta matriz pode ser resumido nos passos i) e ii) a seguir, repetidos para $j = 1,2,\ldots,2T$.

i) chama-se a subrotina UTRINF que resolver $U'_j y_j = (C_B)_j$ onde y_j e $(C_B)_j$ indicam as componentes dos vetores y e C_B correspondentes a U'_j. Se $j = 2T$ o procedimento está terminado. Caso contrário, se $j < 2T$, executamos ii).

ii) Se EXSUJ = 0 fazemos $j = j + 1$ e vamos a i). Se EXSUJ = 1 analisamos se existem colunas espeto associadas a U'_{j+1}. Em caso negativo fazemos $j = j + 1$ e vamos a i). Em caso afirmativo fazemos o produto escalar entre essas colunas espeto e y_j e modificamos $(C_B)_{j+1}$, de acordo.

Fazemos $j = j + 1$ e vamos a i). Nesta parte usamos IPRIMC, USUJ( ), IPROS( ), ICOSUJ( ), ICONSU( ), LINHAS( ) e IPROC( ) para identificar as colunas espeto que interessam junto com seus elementos.

b) MULTI2 (chamada pela SIMESC).

Calcula VMULT = $(L^{-1})' \bar{y}$.

Lembramos que

$$L^{-1} = \overbrace{L_{(m+n)T+k} P_{(m+n)T+k} \cdot \ldots}^{(1)} \cdot \overbrace{L_{(m+n)T-1} P_{(m+n)T-1} \cdot \ldots \cdot L_1 P_1}^{(2)}$$

onde as operações dentro de (1) indicam modificações de uma fatoração inicial e as correspondentes a (2) são as próprias operações dessa fatoração inicial.

Então, como $P'_i = P_i$ temos

$$(L^{-1})' = P_1 L'_1 P_2 L'_2 \ldots P_{(m+n)T-1} L'_{(m+n)T-1} \ldots P_{(m+n)T+k} L'_{(m+n)T+k} \cdot$$

As matrizes L_i' são elementares com uma linha não trivial. Esta linha é exatamente a coluna de L_i cujos valores e apontadores estão armazenados em VALORL( ), LINHAL( ), ICOLL( ) e IPERM( ). Quando $(L^{-1})'$ é aplicada sobre um vetor, a leitura dos vetores mencionados acima deve ser feita de trás para frente. KICOL e KIPER indicam quais são as últimas posições de ICOLL( ) e IPERM( ), respectivamente. KL indica qual é a última posição de VALORL( ) e LINHAL( ).

5.3. OBTENÇÃO DA SOLUÇÃO BÁSICA

Seja v o vetor de recursos do problema e S_B a solução básica associada à base B, temos que

$$BS_B = v \tag{4}$$

como

$$L^{-1}BQ = U \tag{5}$$

e

$$Q^{-1} = Q$$

resulta

$$L^{-1}B = UQ \tag{6}$$

de (4) e (6) deduzimos que

$$UQS_B = L^{-1}v. \tag{7}$$

Então a sequência de operações para obter S_B é:

- Aplicar L^{-1} sobre v
- Resolver $UQS_B = L^{-1}v$.

Chamaremos simplesmente S ao vetor QS_B. Realizamos os passos acima através de três subrotinas: ALCOLU, AUCOLU e UTRISP.

a) ALCOLU (chamada pela SIMESC).

Esta rotina aplica a sucessão de operações de L^{-1} sobre o vetor v.

b) AUCOLU (chamada pela SIMESC).

Esta rotina resolve um sistema triangular superior com as características da figura II.1 deste capítulo.

Para j=2T,2T-1,...,2,1 ela executa as operações i) e ii) a seguir:

i) Chama UTRISP, subrotina que resolve $U_j S_j = (L^{-1}v)_j$, onde S_j e $(L^{-1}v)_j$ indicam as componentes dos vetores S e $L^{-1}v$ correspondentes a U_j. Se j=1 terminou o procedimento. Caso contrário, se j > 1, vai-se para ii).

ii) Se EXSUJ = 0 faz j = j-1 e vai para i). Se EXSUJ=1 analisa se existem colunas espeto sobre as colunas de U_j. Em caso negativo faz j=j-1 e vai para i). Em caso afirmativo usando os valores obtidos em i) para S_j modifica $(L^{-1}v)_{j-1}$. Faz j = j-1 e vai para i).

Nesta parte usamos IULT, USUJ( ), IPROS( ), ICOSUJ( ), ICOSU( ), LINHAS( ) e IANT( ) para identificar as colunas espeto que interessam e os seus elementos.

Para obter $S_B = Q^{-1}S$ utilizamos o vetor IBALO( ).

Cabe aqui um esclarecimento acerca da resolução de um sistema triangular superior,

$$\begin{vmatrix} u_{11} & u_{12} & \cdots & u_{1m} \\ & u_{22} & & u_{2m} \\ & & \ddots & \\ & & u_{jj} & u_{jm} \\ & & & \ddots \\ & & & u_{mm} \end{vmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \\ x_j \\ \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \\ b_j \\ \\ b_m \end{bmatrix}$$

Para aproveitar o armazenamento por colunas deste sistema, procede-se da forma seguinte:

Calculamos

$$x_m = b_m/u_{mm} .$$

Modificamos o termo a direita da igualdade

$$\tilde{b}_i = b_i - u_{im}x_m \quad , \quad i = 1,2,\ldots,m .$$

Ficamos assim com um sistema triangular superior de dimensão

(m-1), cujas colunas são as (m-1) primeiras da matriz inicial. Repetimos o processo até obter x_1. A rotina UTRISP utiliza este procedimento de cálculo.

Se considerarmos que a matriz U completa é triangular superior esparsa com estrutura escada, é claro que fora da escada o procedimento descrito é feito exclusivamente com as colunas espeto.

5.4. ESCOLHA DA VARIÁVEL A ENTRAR NA BASE

Seja 'k' o índice de uma variável não básica. A condição de otimalidade para essa variável é:

$$C_k - \text{VMULT'}A^k \leq 0 \quad \text{se} \quad x_k = \overline{x}_k$$

$$C_k - \text{VMULT'}A^k \geq 0 \quad \text{se} \quad x_k = \underline{x}_k .$$

onde C_k é o custo associado a variável x_k.

A subrotina VAENBA escolhe entre as variáveis que não verificam as condições de otimalidade aquela para a qual $|C_k - \text{VMULT'}A^k|$ toma o máximo valor. Outros esquemas de seleção podem ser adotados, dependendo das características de cada problema (Fourer, 1983).

5.5. OBTENÇÃO DA COLUNA ATUALIZADA

Se A^S é a coluna que entrará na base, devemos achar $\hat{A}^S$ tal que

$$B\hat{A}^S = A^S \tag{8}$$

como em II.5.3 deduzimos que (1) é equivalente a

$$UQ\hat{A}^S = L^{-1}A^S \tag{9}$$

A sequência de operações para obter $\hat{A}^S$ é:

- Aplicar L^{-1} sobre A^S.
- Resolver $Uz = L^{-1}A^S$ onde $z = Q\hat{A}^S$
- Calcular $\hat{A}^S = Q^{-1}z$.

O esquema é praticamente o mesmo usado na obtenção da solução básica atual do ítem II.5.3. A única diferença é que a esparsidade das colunas de A é aproveitada. Quando aplicamos L^{-1} sobre A^S, as L_i que correspondem a períodos anteriores ao associado com A^S, não são aplicadas.

5.6. DETERMINAÇÃO DA COLUNA QUE SAI DA BASE.

É feita da maneira usual através da subrotina VASABA.

5.7. ATUALIZAÇÃO DA FATORAÇÃO LU DA BASE.

Gostaríamos de enfatizar o fato de que o armazenamento esparso, o aproveitamento da esparsidade nos cálculos matriciais e a preservação da esparsidade durante estes cálculos, resultam em programas mais complexos do que aqueles que não contemplam essas características. É precisamente nesta parte do programa que isto se torna mais evidente.

A atualização LU é feita através de treze subrotinas:

a) DADSUJ (chamada pela SIMESC).

Retorna à subrotina SIMESC com as seguintes informações:

i) Se a coluna que sai da base é coluna espeto ou não ;

ii) O número de colunas espeto existentes acima de U_i , sendo U_i a submatriz de U, onde está colocada a coluna que sai da base;

iii) Os índices que caracterizam as colunas espeto imediatamente anterior e posterior a coluna de saída;

iv) A posição em U da última coluna de U_i ;

b) DPPBGL (chamada pela SIMESC).

Localiza em U a posição do primeiro pivô usado para eliminar

a matriz H_i ;

c) ZEMATH (chamada pela SIMESC).

i) Efetua as eliminações na matriz H_i ;

ii) Cria os novos elementos de VALORL( ), LINHAL( ), ICOLL( ), IPERM( ), que representam as operações feitas em i).

d) BGVAUX (chamada pela SIMESC).

Aplica as operações feitas em ZEMATH à coluna que sai da base e à atualização da coluna que entra ($\tilde{A}^S$).

e) PERUZI (chamada pela SIMESC).

i) Permuta as colunas de U_i de acordo com o processo de Bartels-Golub;

ii) Redefine IBALO( ) de acordo com as permutações feitas em i).

f) PERSUJ (chamada pela SIMESC).

Redefine os vetores que caracterizam as colunas espeto acima de U_i, quando estas existem, de acordo com as permutações feitas em PERUZI.

g) BGUSUJ (chamada pela SIMESC).

Esta rotina é chamada somente quando existem colunas espeto (EXSUJ = 1). Neste caso,

i) Aplica as operações feitas em ZEMATH sobre as colunas espeto à direita da última coluna de U_i ;

ii) Analisa se entre essas colunas existe alguma com elemento não nulo na linha que corresponde a última linha de U_i. Em caso afirmativo, devolve o índice característico desta coluna espeto. Em caso negativo avisa que não existe coluna espeto com essa propriedade.

As próximas cinco subrotinas são chamadas somente quando existe a coluna espeto procurada em g) - ii), acima. Neste caso esta coluna será permutada com a coluna que sai da base que atualmente se encontra na posição da última coluna de U_i. A coluna que irá ocupar este lugar tem elementos não nulos em algumas linhas abaixo da linha correspondente à última de U_i. Uma parte destes elementos estão representados numa coluna espeto e outra em U_j com $j > i$. O trabalho a ser realizado é repartido entre as subrotinas que seguem:

h) ZESUJ1 (chamada pela SIMESC).

i) Efetua as eliminações da coluna citada acima, na parte que corresponde à coluna espeto;

ii) Cria os elementos de VALORL( ), LINHAL( ) correspondentes as operações feitas em i).

i) ZESUJ2 (chamada pela SIMESC).

i) Efetua as eliminações da mesma coluna que em ZESUJ1, mas na parte correspondente a U_j ;

ii) Cria os elementos de VALORL( ), LINHAL( ), ICOLL( ) correspondentes as operações feitas em i).

j) LTROCA (chamada pela SIMESC).

i) Aplica sobre a coluna de saída as operações feitas em ZESUJ1 e ZESUJ2;

ii) Aplica sobre a coluna que entra na base já modificada as operações feitas em ZESUJ1 e ZESUJ2.

k) TTROCA (chamada pela SIMESC).

i) Redefine os elementos que caracterizam as colunas espeto, de acordo com as mudanças resultantes da última permutação de colunas;

ii) Redefine os elementos de U( ). Partes de i) e ii) são feitas numa subrotina denominada TRVAUX;

iii) Redefine IBALO( ).

ℓ) TRODEF (chamada pela SIMESC).

i) Faz a troca definitiva da coluna que sai da base pela que entra;

ii) Redefine os vetores que caracteizam as colunas espeto quando houve alguma modificação causada pela troca anterior;

iii) Redefine o vetor U( );

iv) Realiza, quando é necessário, as operações para triangularizar a matriz U depois da troca de colunas ;

v) Cria os elementos de VALORL( ), LINHAL( ), ICOLL( ) correspondentes as operações feitas em iv).

5.2. ROTINA PRINCIPAL

A rotina principal SIMESC realiza essencialmente o seguinte trabalho:

i) Inicializa todos os parâmetros;

ii) Reordena os elementos de INDB( ) na ordem crescente;

iii) Chama as subrotinas DELUST, CRIANT, MULTI1, MULTI2, VAENBA, ALCOLU, AUCOLU, VASABA;

iv) Redefine INDB( ) e INDNB( );

v) Decide se refatoriza a matriz básica atual ou atualiza a fatoração disponível. Se refatoriza, volta a iii). Se atualiza prepara os parâmetros necessários para rotinas de atualização;

vi) Chama sucessivamente essas rotinas;

vii) Imprime as mensagens finais.

6. EXPERIÊNCIAS COMPUTACIONAIS

O programa foi testado em uma série de problemas cujas características indicamos a seguir. Os principais parâmetros dos problemas são:

n - número de restrições dinâmicas por período, também é o número de variáveis de estado por período.

m - número de restrições não dinâmicas por período.

r - número de controles por período.

T - número total de períodos considerados.

O primeiro teste foi feito para um problema com $m = 6$, $n = 6$ e $r = 12$ para vários valores de T. A tabela II.6.1 mostra os resultados de uma fase 1 para diferentes valores de T. Além de T, as entradas da tabela representam

D - densidade da matriz de restrições

NR - número total de restrições

NV - número total de variáveis

P.R. - posições de memória ocupadas por variáveis reais.

P.I. - posições de memória ocupadas por variáveis inteiras.

NATUAL - número máximo de atualizações permitido, ou seja, cada NATUAL atualizações a matriz e refatorizada.

ITER - número de iterações até o ótimo.

C.P.U. - tempo de CPU em segundos por iteração

FASE 1.

T	D	NR	NV	PR	PI	NATUAL	ITER	CPU
12	1.33%	144	432	3890	4151	50	156	0.36
12	1.33%	144	432	-	-	0	156	0.5
24	0.67%	288	864	7504	6982	50	307	1
24	0.67%	288	864	-	-	0	307	1.7
36	0.33%	432	1728	12246	10946	50	458	2.13
36	0.33%	432	1728	-	-	0	458	3.78
96	0.13%	1152	4608	32414	27871	50	1213	13

TABELA II.6.1.

A tabela II.6.2 mostra os resultados para a fase 2

FASE 2.

T	NR	NV	PR	PI	NATUAL	ITER	CPU
12	144	288	4546	4592	50	47	0.15
12	144	288	-	-	0	47	0.32
24	288	576	7576	7118	50	47	0.34
24	288	576	-	-	0	47	0.83
36	432	1296	11758	10508	50	47	0.61
36	432	1296	-	-	0	47	1.75
96	1152	3456	29078	24588	50	47	3
96	1152	3456	-	-	0	47	8.93

TABELA II.6.2.

O segundo teste é a aplicação do algoritmo a um problema de planejamento da operação de um sistema hidroelétrico, discutido em detalhe no capítulo III. Neste problema temos m=1, n=2, r = 6 , T = 42. A tabela II.6.3. mostra os resultados da fase 2.

T	D	NR	NV	PR	PI	NATUAL	ITER	CPU
42	1%	126	336	3678	3247	10	25	0.3
42	1%	126	336	-	-	0	18	0.8

CAPÍTULO III

EXPERIÊNCIA COMPUTACIONAL COM UM PROBLEMA LINEAR

1. INTRODUÇÃO

Discutimos neste capítulo a aplicação do algoritmo e programa apresentados nos capítulos anteriores a um problema de planejamento da operação de um sistema hidroelétrico situado no médio São Francisco.

2. DESCRIÇÃO DO PROBLEMA

Uma descrição física do sistema e a formulação dum modelo matemático para o problema em estudo encontra-se no trabalho de Cardoso(1981). O sistema representado na figura III.1 é formado por quatro usinas, sendo duas com reservatório, Sobradinho e Moxotó; as outras usinas, Paulo Afonso I-II-III(P.A.I-II-III) e Paulo Afonso IV(P.A.IV) são a fio d'agua

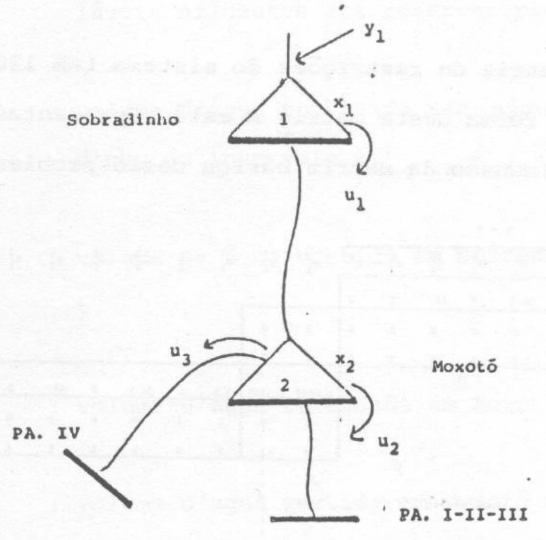


Figura III.1 Sistema CHESF no médio São Francisco

Obtém-se uma solução factível inicial para a etapa 1 , considerando que os reservatórios vertem toda vazão afluente.

Da simples observação das equações dinâmicas (1) e (2) , nota-se que os coeficientes das variáveis do problema nestas equações são 1, -1 ou 0. A partir dos dados para A_i, E_i e C_i que estão no trabalho citado de Cardoso, obtivemos a seguinte equação de balanço energético, já considerada a compatibilização das unidades das variáveis do problema:

$$(11)\quad 0.0073x_1^t+0.1871x_2^t+7.8u_1^t+34.3u_2^t+37u_3^t-354.7+\Delta^t = d^t$$

A equação(11) substitui as equações(6), (9) e (10). A unidade usada para x_i^t, u_i^t é $10^6 m^3$. Para Δ^t e d^t adotou-se Mw médio.

Os coeficientes na equação(11) estão coerentes com estas unidades.

A matriz de restrições do sistema tem 126 linhas e 378 colunas. A forma desta matriz A está representada na figura III.2.

A dimensão da matriz básica deste problema é 126x126.

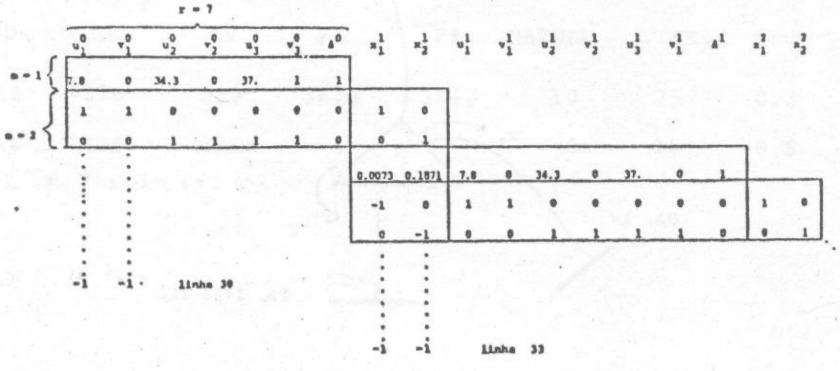


Figura III.2

As equações dinâmicas que correspondem à conservação d'agua nos reservatórios são:

$$(1)\quad x_1^{t+1} = x_1^t + y_1^t - u_1^t - v_1^t$$

$$(2)\quad x_2^{t+1} = x_2^t + y_2^t + u_1^{t-\tau} + v_1^{t-\tau} - u_2^t - u_3^t - v_2^t - v_3^t$$

$$t = 0, 1, \ldots, T-1$$

onde

x_1^t, x_2^t : volumes d'agua armazenada em Sobradinho e Moxotó, respectivamente, no intervalo de tempo t.

y_1^t, y_2^t : volumes d'agua correspondente as vazões não controláveis afluentes aos reservatórios no intervalo t.

u_1^t : volume d'agua turbinada nas máquinas de Sobradinho no intervalo t.

v_1^t : volume de água vertido em Sobradinho no intervalo t.

u_2^t : volume d'agua turbinada em Moxotó e P.A. I-II-III.

v_2^t : volume d'agua vertida em Moxotó e P.A. I-II-III.

u_3^t : volume d'agua turbinada em P.A. IV

v_3^t : volume d'agua vertida em P.A. IV

τ : número de intervalos correspondente ao tempo de transporte da onda d'agua entre Sobradinho e Moxotó.

T : instante final do horizonte de estudo.

Os volumes, turbinagens e vertimentos estão sujeitos a limites:

(3) $\underline{x}_i \leq x_i^t \leq \overline{x}_i$ $\quad i=1, 2; \ t=1,\ldots, T$

(4) $\underline{u}_i \leq u_i^t \leq \overline{u}_i$ $\quad i=1, 2, 3; \ t=0,\ldots, T-1$

(5) $v_i^t \leq 0$ $\quad i=1, 2, 3; \ t=0,\ldots T-1$

Uma outra equação representa o balanço energético do sistema

(6) $h_1^t + h_2^t + \Delta^t = d^t$ $\quad t=0,\ldots, T-1$

onde

(7) $h_1^t = g_1(x_1^t, u_1^t, v_1^t)$

(8) $h_2^t = g_2(x_2^t, u_2^t) + g_3(u_2^t, v_2^t) + g_4(x_2^t, u_3^t)$

d^t : demanda de energia no intervalo t.

Δ^t : corte de carga ou geração térmica.

g_1, g_2, g_3, g_4 funções de geração de energia correspondentes às usinas de Sobradinho, Moxotó, P.A. I-II-III e P.A. IV respectivamente.

As funções g_i são não lineares. No trabalho citado foram usadas as aproximações lineares representadas nas equações (9) e (10).

$$(9)\quad h_1^t = A_1 u_1^t + E_1 x_1^t + C_1$$

$$(10)\quad h_2^t = A_2 u_2^t + A_3 u_3^t + E_2 x_2^t + C_2$$

onde A_i, E_i e C_i são constantes.

Deseja-se coordenar a operação semanal deste sistema de modo que as necessidades de energia sejam atendidas apenas com geração hidráulica, ou seja, $\Delta^t = 0$ em (6), para todos os intervalos t considerados. Todas as restrições de operação devem ser atendidas e a reserva de energia, que neste sistema é a água armazenada em Sobradinho, deve ser poupada ao máximo.

Consideram-se duas semanas de operação, divididas em 42 intervalos de 8 horas. O tempo de viagem d'agua entre os reservatórios de Sobradinho e Moxotó, é de 3 dias que resultam num retardo de 9 períodos, ou seja $\tau = 9$ em (2).

O problema é resolvido em duas etapas:

A primeira, denominada fase térmica, consiste em verificar se é factível atender toda a demanda sem geração térmica ou corte de de carga. Em termos das variáveis do modelo matemático, isto significa que procuramos uma solução factível de nosso problema com $\Delta^t = 0$, para todos os intervalos t considerados. Resolvida positivamente a primeira etapa a segunda consistirá em procurar entre todas as soluções factíveis da primeira etapa aquela que maximize o volume de água armazenada em Sobradinho no instante final.

A formulação matemática para a primeira etapa é:

$$\min \quad J = \sum_{t=0}^{41} \Delta^t$$

sujeito às restrições (1), (2), (3), (4), (5), (6), (9) e (10) e

$\Delta^t \geq 0$

dados $x_1^0,\ x_2^0$

$y_1^t,\ y_2^t \qquad t = 0, 1, \ldots, 41$

$u_1^{t-9} + v_1^{t-9} \qquad t = 0, 1, \ldots, 8$

$d^t \qquad t = 0, 1, \ldots, 41$

3. EXPERIÊNCIA NUMÉRICA

Vamos fazer algumas considerações sobre a estabilidade numérica do algoritmo proposto quando aplicado a este problema. Como m=1, o pivô da primeira linha de cada período está totalmente determinado. Se acontecer que as colunas associadas à x_1^t e u_3^t para o mesmo t, estejam presentes numa base, pode acontecer que o pivô seja o valor 0.0073. O quociente entre o coeficiente de u_3^t e este pivô é :

$$3 \quad /0.0073 = 5068.4931$$

se fizermos os cálculos com oito dígitos significativos. Este valor, razoàvelmente grande, repetido através de vários períodos desestabiliza numéricamente o algoritmo. Como exemplo, consideremos uma submatriz que corresponda a um único período nessa situação:

$$B = \begin{vmatrix} 0.0073 & 0 & 37 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Seja $b' = (370073., -10^4, 10^4)$

Se para resolver Bx=b fatoramos B sem permutar a primeira linha, teremos:

$$U = \begin{vmatrix} 0.0073 & 0 & 37 \\ & 1 & 5068.4931 \\ & & 1 \end{vmatrix}$$

A matriz L^{-1} consiste numa única matriz elementar L_1 representa

da pelo vetor:

$$(1,\ 136.9863,\ 0)'$$

Então temos

$$L^{-1}b = (370073.,50684930,\ 10^4)'$$

e resolvendo

$$Ux = L^{-1}b$$

obtemos

$$x' = (10^4,\ -1,\ 10^4)$$

mas a solução de

$$Bz = b \quad \text{é} \quad z' = (10^4,\ 0,\ 10^4)$$

No problema estudado, esta desestabilização se manifestou através da apárição de soluções básicas que não verificam alguma das restrições de limites(3), (4) ou (5). Para verificar se a causa desta infactibilidade era realmente a presença desse quociente, retomamos uma solução básica correspondente a uma iteração anterior àquela em que aparece a primeira infactibilidade. A partir desta solução básica factível fizemos um certo número de iterações "não esparsas", quer dizer, liberamos a escolha do pivô em prejuízo da preservação da estrutura. Desta maneira não aconteceu a infactibilidade confirmando nossa hipótese. Finalizadas as iterações não esparsas retomamos novamente o algoritmo esparso. Assim conseguimos convergir para uma solução ótima desta primeira etapa do problema, mas com o sacrifício da esparsidade.

4. MODIFICAÇÕES PARA MELHORAR A ESTABILIDADE

Para evitar o problema descrito em III.3. pensamos em duas opções:

a) controlar a entrada das variáveis não básicas, eliminando quando possível a presença dos coeficientes desestabilizadores.

b) modificar a equação de balanço energético.

Como inicialmente é de nosso interesse testar a eficiência do algoritmo proposto, em problemas estáveis optamos pela alternativa b). Para isso usamos uma outra forma de linearizar as funções de geração. Consideramos a geração média de cada usina para os volumes variando entre os limites inferior e superior isto é:

$$\bar{g}_i(u) = \int_{\underline{x}_i}^{\bar{x}_i} g_i(x_i, u)\, dx_i$$

as funções $\bar{g}_i(u)$ não lineares em u foram aproximadas por uma função linear das turbinagens utilizando um ajuste de 1000 pontos entre $\underline{u}$ e $\bar{u}$. As funções de geração $g_i(x_i, u)$ consideradas são os polinomios utilizados no trabalho de Lyra, Friedlander e Gercmel (1982).

Com estas mudanças a equação (11) foi substituída por:

$$(12)\quad 8.26u_1^t + 32.98u_2^t + 34.39u_3^t + \Delta^t = d^t$$

Usando a equação de balanço energético(12) o algoritmo convergiu sem nenhum problema intermediário de infactibilidade . O programa foi rodado no computador VAX/785 com sistema operacional VMS, utilizando o compilador Fortran 77.

Uma solução ótima para a primeira etapa foi obtida em 195 iterações, demorando em média 1 segundo de C.P.U por iteração.

Com a solução da etapa 1, passamos à resolução da segunda etapa, denominada fase hidráulica. Neste caso, o objetivo foi a maximização do volume d'agua que permanece em Sobradinho no fim do horizonte de estudo.

ETAPA 2: Considerando a nova função objetivo e a inexistência de geração térmica ou corte de carga tem-se:

$$\max J = x_1^{42}$$

sujeito a (1), (2), (3), (4), (5) e(13)

onde

$$(13)\quad 8.26u_1^t + 32.98u_2^t + 34.39u_3^t = d^t$$

(note-se que as variáveis Δ^t não figuram mais nas equações).
O algoritmo convergiu ao ótimo em 20 iterações usando em média , 0,6 segundos de C.P.U por iteração.

Os valores mínimo e máximo considerados para x_1^t são res-

pectivamente 19054 e 19484(em $10^6 m^3$). O valor ótimo obtido com este modelo é

$$x_1^{42} = 19466.27$$

A tabela III.1 representa a quantidade de energia gerada em cada período pela solução ótima obtida na etapa 2, considerando como funções de geração os polinomios usados no artigo de Lyra, Friedlander e Geromel(1982).

As duas primeiras entradas desta tabela fornecem os valores correspondentes a:

1) As Linearizações dos Polinomios(Estes valores coincidem em todos os dígitos impressos com os valores das demandas dadas).
2) Os polinomios de geração.

A terceira entrada dessa tabela representa as diferenças entre esses valores, ou em outras palavras, o erro cometido quando as aproximações lineares são usadas. Observar que o erro máximo é de 1,8%, que é aceitável quando se leva em conta outras aproximações e a qualidade das informações disponíveis.

PERÍODO	LINEAR	POLINOMIOS	ERRO
1	1808.000	1802.896	5.104
2	2190.000	2176.776	13.224
3	2463.000	2460.971	2.029
4	1879.000	1867.349	11.651
5	2276.000	2257.687	18.313
6	2559.000	2535.734	23.266
7	1816.000	1813.029	2.971
8	2200.000	2179.214	20.786
9	2474.000	2454.790	19.210
10	1844.000	1842.357	1.643
11	2234.000	2223.565	10.435
12	2512.000	2498.796	13.204
13	1911.000	1900.925	10.075
14	2315.000	2298.671	16.329
15	2604.000	2578.589	25.411
16	1835.000	1825.159	9.841
17	1956.000	1940.364	15.636
18	2413.000	2391.635	21.365
19	1738.000	1728.922	9.078
20	1571.000	1552.008	18.992
21	2176.000	2155.323	20.677
22	1859.000	1847.709	11.291
23	2253.000	2234.861	18.139
24	2533.000	2522.551	10.449
25	1906.000	1887.156	18.844
26	2309.000	2265.320	43.680
27	2596.000	2557.843	38.157
28	1864.000	1842.926	21.074
29	2258.000	2234.418	23.582
30	2539.000	2500.360	38.640
31	1833.000	1825.781	7.219
32	2220.000	2208.721	11.279
33	2497.000	2470.465	26.535
34	1833.000	1836.941	-3.941
35	2220.000	2224.773	-4.773
36	2497.000	2504.400	-7.400
37	1894.000	1899.759	-5.759
38	2019.000	2025.139	-6.139
39	2491.000	2498.397	-7.397
40	1698.000	1703.163	-5.163
41	1535.000	1538.300	-3.300
42	2126.000	2130.571	-4.571

Tabela III.1 Energia gerada e erro de aproximação para a solução ótima da etapa 2 do problema.

5. COMENTÁRIOS

É interessante observar que a duração média de cada iteração e o número de iterações da etapa 1 são significativamente maiores que os da etapa 2. Como as dimensões e caracterísi-

cas dos sistemas lineares resolvidos em cada caso são as mesmas, suspeitamos que o tempo adicional por iteração na etapa 1, é consumido em sua maior parte pela rotina que faz a escolha da variável que entra na base em cada iteração. O número de iterações poderia ser reduzido dando prioridade para entrar na base às variáveis que ocasionem incrementos maiores na função objetivo. Algumas idéias neste sentido, para este problema particular estão no trabalho de Carneiro da Silva(1984).

Fourer (1983) tem propostas gerais para matrizes com estrutura escada para esta parte do método Simplex. Em trabalho futuro pretendemos estudar a incorporação de técnicas similares ao nosso programa.

CAPÍTULO IV

PROGRAMAÇÃO NÃO LINEAR DINÂMICA

1. INTRODUÇÃO

Neste capítulo descrevemos um algoritmo de programação não linear para problemas de grande porte caracterizados por restrições lineares com estrutura escada. Estamos particularmente interessados na resolução de problemas onde o número de variáveis que aparecem não linearmente na função objetivo seja pequeno em relação ao número total de variáveis.

Os problemas a considerar podem ser representados como

$$(1) \qquad \min_{x \in E^n} f(x) = F(x^N) + c'x$$

sujeito a

$$(2) \qquad Ax = b$$

$$(3) \qquad \underline{x} \leq x \leq \overline{x}$$

onde $A \in E^{m \times n}$, $m \leq n$, $b \in E^m$ e $c \in E^n$.

Partimos o vetor x na forma $x' = (x^L, x^N)'$ onde as componentes de x^L e x^N aparecem de forma linear e não linear, respectivamente, na função objetivo. Note-se que A e c operam sobre todas as componentes de x. Assumimos que $F(x^N)$ é uma função de classe C^1 na região de factibilidade com gradiente $\nabla F(x^N)$.

O algoritmo que apresentamos está baseado no método de gradiente reduzido (Wolfe, 1962). Mais especificamente, está baseado na implementação desenvolvida por Murtagh e Saunders (1978). Este método estende o conceito de solução básica em programação linear permitindo que mais de m variáveis estejam estritamente entre os seus limites.

Devido à estreita ligação do método que apresentamos com a programação linear, várias partes do algoritmo descrito nos capítulos I e II podem ser incorporados na implementação deste método.

A partição de x e $f(x)$ nas suas partes lineares e não lineares é de importância prática, mas para descrever o algoritmo que usaremos é conveniente considerar as variáveis e a função globalmente. Chamamos a nossa função objetivo simplesmente $f(x)$ e denotamos o seu gradiente $\nabla f(x)$.

2. VARIÁVEIS SUPERBÁSICAS

Quando tratamos com problemas de programação linear,

sabemos que a existência de uma solução ótima implica na existência de uma solução ótima básica. Se o problema não for linear não podemos mais esperar que a solução ótima seja uma solução básica. Mas se o número de variáveis que entram não linearmente no problema é pequeno, parece razoável supor que existirá uma solução ótima "quase" básica. Fazendo uma generalização simples (Murtagh e Saunders, 1978) é introduzido o conceito de variáveis superbásicas. Para isso define-se uma partição da matriz de restrições da seguinte forma

$$Ax = \begin{array}{|c|c|c|} \multicolumn{1}{c}{m} & \multicolumn{1}{c}{s} & \multicolumn{1}{c}{n-m-s} \\ \hline B & S & N \\ \hline \end{array} \begin{bmatrix} x_B \\ x_S \\ x_N \end{bmatrix} = b \quad .$$

As colunas da matriz B formam uma base de E^m, como no método Simplex, $S \in E^{m \times s}$ com $0 \leq s \leq n-m$, e N é a matriz formada pelas colunas restantes de A. As variáveis associadas x_B, x_S, x_N são chamadas básicas, superbásicas e não básicas respectivamente. As variáveis básicas e superbásicas podem tomar qualquer valor entre os limites dados. O nome de superbásicas foi escolhido por Murtagh e Saunders (1978) para salientar o papel destas variáveis na escolha de direções de deslocamento. Estas variáveis podem ser movidas em qualquer direção, e em consequência, as variáveis básicas são forçadas a mudar de maneira que a factibilidade de (2) seja mantida.

A expectativa de que as soluções ótimas de problemas com

poucas variáveis não lineares sejam 'quase' básicas é confirmada pelo teorema que enunciamos a seguir:

TEOREMA 1 (Jain). Se um problema de programação não linear possui t variáveis que aparecem não linearmente, e se o problema tem solução ótima; então existe uma solução ótima do problema com no máximo t variáveis superbásicas.

O conceito de variáveis superbásicas é utilizado na implementação do pacote computacional MINOS (Murtagh e Saunders, 1977).

3. DESLOCAMENTOS CONSERVANDO AS RESTRIÇÕES ATIVAS

Suponhamos a existência de um ponto factível x, onde as variáveis estão divididas em básicas, superbásicas e não básicas.

$$x' = (x_B , x_S , x_N)' .$$

Então x satisfaz

$$Bx_B + Sx_S + Nx_N = b \tag{4}$$

$$x_N = d \tag{5}$$

onde d_i vale $\underline{x}_i$ ou $\bar{x}_i$.

Queremos nos deslocar para um ponto y onde as variáveis não básicas fiquem inalteradas. Em outras palavras, queremos que o subconjunto de restrições ativas de (3) continue o mesmo e com as variáveis não básicas nos mesmos valores. Então, também as variáveis básicas e superbásicas podem continuar sendo as mesmas, e y deve satisfazer

$$By_B + Sy_S + Ny_N = b \tag{6}$$

$$y_N = d \,. \tag{7}$$

Subtraindo (4) de (6) e (5) de (7) e definindo-se $\Delta x = y - x$ temos

$$B\Delta x_B + S\Delta x_S + N\Delta x_N = 0 \tag{8}$$

$$\Delta x_N = 0 \,. \tag{9}$$

Multiplicando (8) por B^{-1} obtemos que (8) e (9) equivalem a:

$$\Delta x_B + B^{-1}S\Delta x_S = 0 \tag{10}$$

$$\Delta x_N = 0 \tag{11}$$

Assim, as condições que deve satisfazer um deslocamento que conserve as restrições ativas de (3) são:

(12) $$\Delta x_B = -B^{-1}S\Delta x_S$$

(13) $$\Delta x_N = 0 \ .$$

A equação (12) mostra como é determinado o deslocamento das variáveis básicas quando as variáveis superbásicas são movimentadas. Note-se a analogia com o método de gradiente reduzido.

4. CÁLCULO DO GRADIENTE REDUZIDO EM RELAÇÃO ÀS VARIÁVEIS SUPERBÁSICAS.

A partir de (4) deduzimos que:

(14) $$x_B = B^{-1}b - B^{-1}Sx_S - B^{-1}Nx_N$$

(15) $$x_N = d \ .$$

Como x_N está fixo, os valores de x_B dependem dos valores de x_S. Ou seja, temos:

(16) $$h(x_S) = f(B^{-1}b - B^{-1}Sx_S - B^{-1}Nd, x_S, d) = f(x_B, x_S, x_N).$$

Então

(17) $$\nabla_{x_S} h(x_S) = \nabla_{x_S} f(x_B, x_S, x_N) - (B^{-1}S)' \nabla_{x_B} f(x_B, x_S, x_N).$$

5. DESCRIÇÃO GERAL DO ALGORITMO.

Dados x^k, um ponto que satisfaz (2) e (3), $\varepsilon > 0$, e uma partição de $A = [B,S,N]$. Seja s o número de variáveis superbásicas.

A cada iteração, o algoritmo executa os seguintes passos:

PASSO 1 - Calcula $\nabla f(x^k)$

PASSO 2 - Resolve $B'\lambda^k = \nabla_{x_B} f(x^k)$

PASSO 3 - Calcula $\nabla_{x_S} h(x_S^k) = \nabla_{x_S} f(x^k) - S'\lambda^k$

Se $s = 0$ ou $\|\nabla_{x_S} h(x_S^k)\| < \varepsilon$ vai ao passo 8.

Em caso contrário ainda existe uma direção de deslocamento factível na região dada por (14) e (15) que provoca um decréscimo em $f(x)$. Vai ao passo 4.

PASSO 4 - Determina uma direção de decréscimo para $h(x_S^k)$, ou seja Δx_S^k tal que $\langle \Delta x_S^k, \Delta h_{x_S}(x_S^k) \rangle < 0$.

PASSO 5 - Determina a direção de deslocamento correspondente às variáveis básicas resolvendo

$$B\Delta x_B^k = -S\Delta x_S^k$$

A direção de deslocamento total é Δx^k ,

$$\Delta x^{k'} = (\Delta x_B^k \ , \ \Delta x_S^k \ , \ 0)' \ .$$

PASSO 6 - Calcula α^1 e α^2 tais que

$$\alpha^1 = \max(\alpha / x_B^k + \alpha \Delta x_B^k \in [\underline{x}_B \ , \bar{x}^B])$$

$$\alpha^2 = \max(\alpha / x_S^k + \alpha \Delta x_S^k \in [\underline{x}_S \ , \bar{x}^S])$$

Define $\alpha^* = \min(\alpha^1 \ , \alpha^2)$.

PASSO 7 - Procura $\gamma \in [0, \alpha^*]$ tal que $f(x^k + \gamma \Delta x^k) \leq f(x^k)$ ou seja, $h(x^S + \gamma \Delta x_S^k) \leq h(x_S^k)$.
Faz $x^{k+1} = x^k + \gamma \Delta x^k$.
Se $\gamma < \alpha^*$, faz $k = k+1$ e vai ao passo 1.
Se $\gamma = \alpha^*$, significa que uma variável básica ou uma superbásica atingiu um limite. Se foi uma superbásica, ela é declarada não básica. Redefine S e N, faz $s = s-1$, $k = k+1$ e vai ao passo 1.
Se foi uma básica ela é declarada não básica. Redefine B,S,N e faz $s = s-1$, $k = k+1$ e vai ao passo 1.

PASSO 8 - Chega-se a este passo quando $\| \nabla_{x_S} h(x_S^k) \| < \varepsilon$ ou $s = 0$.
Assim, resta verificar as condições de otimalidade para as variáveis não básicas, isto é,

$$C^k_{N,j} \begin{cases} \leq 0 & \text{se } x^k_{N,j} = \bar{x}_{N,j} \\ \geq 0 & \text{se } x^k_{N,j} = \underline{x}_{N,j} \end{cases}$$

onde $C^k_{N,j} = (\nabla_{x_N} f(x^k) - N'\lambda^k)_j$.

Chamamos aos $C^k_{N,j}$ custos reduzidos por analogia com o Método Simplex. Se os custos reduzidos de todas as variáveis não básicas satisfazem as condições de otimalidade acima, x^k é uma solução ótima do problema. Em caso contrário o algoritmo escolhe uma variável não básica cujo custo reduzido não satisfaz a otimalidade, e a declara super básica. Redefine S,N faz s = s+1 e vai ao passo 4.

Observe-se que a cada iteração é definido um subproblema que consiste em minimizar a função objetivo restrita a uma variedade linear afim cuja dimensão coincide com o número de variáveis superbásicas. Quando este problema é resolvido passamos a resolver outro subproblema similar cuja variedade linear afim correspondente é obtida acrescentando uma variável não básica ao conjunto de variáveis superbásicas, o que aumenta em 1 a dimensão da variedade linear afim resultante. O Teorema de Jain enunciado em IV.2 nos diz que se o número de variáveis que intervém não linearmente é pequeno podemos esperar que as dimensões das variedades afins destes subproblemas também o sejam.

Por outro lado, quando estamos perto da solução ótima é

de se esperar que as restrições de limites ativas sejam as que correspondem à solução ótima, portanto a estratégia do algoritmo é boa no sentido de que conserva estas restrições.

6. ESPECIALIZAÇÃO PARA O PROBLEMA DE RESTRIÇÕES COM ESTRUTURA ESCADA.

Na resolução de problemas dinâmicos com restrições lineares e função objetivo não linear usando o algoritmo descrito em IV.5 é possível incorporar as técnicas descritas nos capítulos I e II.

Os sistemas lineares que precisam ser resolvidos a cada iteração têm as mesmas características que no caso linear discutidos nos capítulos I e II. No caso do sistema linear que obtém os multiplicadores λ^k (Passo 2) a diferença reside só no fato de que em lugar do vetor de custos básicos constantes tem-se $\nabla_{x_B} f(x^k)$ que deve ser calculado a cada iteração. O sistema linear que determina o deslocamento das variáveis básicas em função das superbásicas tem as mesmas características que o sistema resolvido no caso linear para achar as soluções básicas.

Para resolver estes sistemas também fatoramos a matriz B na forma LU.

Quando uma variável básica é trocada por uma superbásica, precisamos modificar a fatoração LU como no caso linear. A

variável superbásica que substitui a variável básica tem uma coluna associada em S que deve ser atualizada quando uma troca de colunas é feita. A atualização desta coluna é feita resolvendo um sistema linear com as mesmas características do sistema que aparace no caso linear para a atualização de colunas.

Mencionamos a seguir os passos do algoritmo descrito em IV.5 nos quais incorporamos as técnicas específicas para matrizes com estrutura escada.

PASSO 2 - Para a resolução do sistema $B'\lambda^k = \nabla_{x_B} f(x^k)$ usamos as rotinas MULTI1, MULTI2 e UTRINF (descritas no capítulo II).

PASSO 5 - O sistema $B\Delta x_B^k = -S\Delta x_S^k$ é resolvido mediante as rotinas ALCOLU, AUCOLU e UTRISP (descritas no capítulo II).

PASSO 7 - Quando é feita uma troca de colunas entre B e S usamos o esquema para atualizar a fatoração LU de B introduzido no capítulo I e as rotinas que o implementam apresentadas no capítulo II. A coluna de S que passa para B é atualizada resolvendo o sistema linear correspondente mediante as rotinas ALCOLU, AUCOLU e UTRISP.

7. O PROGRAMA

Comentaremos neste ítem apenas as rotinas não descritas anteriormente.

A rotina principal é chamada GRST e o seu trabalho essencial consiste em:

i) Inicializar parâmetros;

ii) Reordenar os índices básicos na ordem crescente;

iii) Chamar a DELUST para fatorar a matriz B;

iv) Chamar a rotina GRFOBJ que calcula o gradiente e o valor da função objetivo. Nesta rotina deve ser explorado o fato de ter poucas variáveis envolvidas não linearmente;

v) Chamar MULTI1, MULTI2 para obter λ^k ;

vi) Calcular o gradiente em relação às variáveis superbásicas. Se este gradiente for "quase nulo", chamar a VAENBA para procurar uma variável não básica que não verifique as condições de otimalidade;

vii) Calcular direções de decréscimo nas variáveis superbásicas. Isto é feito usando o método do gradiente conjugado a partir do gradiente reduzido superbásico;

viii) Chamar ALCOLU e AUCOLU para obter o deslocamento correspondente às variáveis básicas;

ix) Determinar α^* ;

x) Chamar à subrotina BUSCA que procura γ tal que $f(x^k + \gamma\Delta x^k) \leq f(x^k)$;

xi) Quando $\gamma = \alpha^*$ avisar qual é a variável que atingiu um limite;

xii) Se em xi) a variável que bateu num limite for básica, procurar uma coluna superbásica em condições de substituir a que sai de B. Chamar ALCOLU e AUCOLU para atualizar essa coluna;

xiii) Se for preciso trocar colunas entre B e S, chamar as subrotinas enumeradas no capítulo II que correspondem à atualização da fatoração LU de B.

A menos de VALORC( ) no qual é armazenado o $\nabla f(x^k)$ a cada iteração, todos os vetores apresentados no capítulo II são usados neste programa com o mesmo significado. Um vetor INDSB( ) indicará o conjunto de índices correspondentes às variáveis superbásicas.

A subrotina de busca unidimensional BUSCA utiliza o seguinte procedimento:

Seja $\theta = \alpha^*$

i) Considera os pontos

$$(x^k, f(x^k)) \quad e \quad (x^k + \theta\Delta x^k, f(x^k + \theta\Delta x^k))$$

e os gradientes $\nabla f(x^k)$ e $\nabla f(x^k + \theta\Delta x^k)$

interpola uma cúbica para estes dados (Luenberger,1984).

Seja γ o parâmetro tal que $f(x^k + \gamma\Delta x^k)$ minimize a cúbica interpolada.

ii) Se $f(x^k + \gamma\Delta x^k) \leq f(x^k)$ retorna a GRST e faz $x^{k+1} = x^k + \gamma\Delta x^k$.

Em caso contrário faz $\theta = \gamma$ e refaz a interpolação, isto é, volta para i).

8. EXPERIÊNCIA COMPUTACIONAL

O programa foi aplicado ao problema de planejamento da operação de um sistema hidroelétrico discutido no capítulo III.

Lembramos que as funções de geração de energia são originalmente funções não lineares que, no capítulo III, foram linearizadas.

As funções de geração foram aproximadas por

$$\bar{g}_i(u) = \int_{\underline{x}_i}^{\bar{x}_i} g_i(x_i,u)dx \quad . \tag{18}$$

Esta aproximação é aceitável para o problema em questão e evita eventuais problemas de estabilidade (devidos aos coeficientes de x_i) quando $\bar{g}_i(u)$ aparece nas restrições do problema (capítulo V).

A função de geração total de energia para o sistema é dada por

$$(19) \qquad \bar{g}(u) = -0.00853u_1^2 + 8.3u_1 + 32.98u_2 + 34.39u_3 \ .$$

A formulação matemática da etapa 1 (fase térmica) é dada por

$$(20) \qquad \min \sum_{t=0}^{41} (\bar{g}(u^t) - d^t)^2$$

sujeita a

$$(21) \qquad x_1^{t+1} = x_1^t + y_1^t - u_1^t - v_1^t$$

$$(22) \qquad x_2^{t+1} = x_2^t + y_2^t + u_1^{t-\tau} + v_1^{t-\tau} - u_2^t - u_3^t - v_2^t - v_3^t$$
$$t = 0,1,\ldots,41$$

$$(23) \qquad \underline{x}_i \leq x_i^t \leq \bar{x}_i \quad ; \quad i = 1,2 \ ; \ t = 1,2,\ldots,42$$

$$(24) \qquad \underline{u}_i \leq u_i^t \leq \bar{u}_i \quad ; \quad i = 1,2,3 \ ; \ t = 0,1,2,\ldots,41$$

$$(25) \qquad v_i^t \geq 0 \quad ; \quad i = 1,2,3 \ ; \quad t = 0,1,\ldots,41$$

O significado de todas estas variáveis é o mesmo que no capítulo III. Nesse capítulo, obtivemos na resolução da etapa 1 uma solução ótima, considerando como equação de balanço energético uma

aproximação linear de (19) dada por

$$(26) \qquad \bar{\bar{g}}(u) = 8.26u_1 + 32.98u_2 + 34.39u_3 \ .$$

Vamos aproveitar aquela solução e colocar o problema da seguinte maneira:

$$(27) \qquad \min \sum_{t=0}^{41} (\bar{g}(u) - d^t)^2$$

sujeito a (21), (22), (23), (24) e (25).

Colocado desta forma o problema fica somente com restrições dinâmicas. Isto significa que se m é o número de equações não dinâmicas do problema por período, neste caso $m = 0$. O programa implementado não contempla essa possibilidade (as mudanças necessárias para isso não são muitas e serão feitas futuramente); então para testá-lo consideramos o problema

$$\min \sum_{t=0}^{41} (\bar{g}(u^t) - d^t)^2$$

sujeito a (21), (22), (23), (24), (25) e

$$(27) \qquad \bar{\bar{g}}(u^t) + e^t = d^t \quad \text{com} \quad e^t \quad \text{irrestrito.}$$

Uma solução inicial factível é a solução ótima obtida na etapa 1 do capítulo III.

Rodamos este problema no computador VAX/785 com sistema operacional VMS usando o compilador FORTRAN 77. Uma solução ótima foi obtida em 126 iterações demorando em média 0.7 segundos de CPU por iteração.

No capítulo seguinte vamos considerar a equação de balanço energético dada por (19), como uma restrição. Isto nos coloca na situação de resolver um problema dinâmico de otimização com restrições não lineares.

CAPÍTULO V

PROGRAMAÇÃO NÃO LINEAR DINÂMICA COM RESTRIÇÕES NÃO LINEARES

1. INTRODUÇÃO

Neste capítulo vamos estudar a aplicação das técnicas desenvolvidas neste trabalho para programação linear dinâmica a problemas onde algumas restrições são não lineares.

Muitos problemas reais dão origem a formulações matemáticas onde aparecem poucas variáveis envolvidas não linearmente em proporção ao número total de variáveis. Também é frequente que o número de restrições não lineares seja pequeno em relação ao total de restrições.

Mostramos a viabilidade de um algoritmo para resolver problemas não lineares dinâmicos com estas catacterísticas utilizando as técnicas dos capítulos I e II.

O algoritmo proposto se enquadra na categoria dos métodos de gradiente reduzido generalizado (G.R.G.). O primeiro método G.R.G. foi apresentado por Abadie e Carpentier (1965,1969). Seguiram-se os trabalhos de Abadie (1978), Lasdon e outros (1978) e Sargent e Murtagh (1973).

Na perspectiva de aplicar o método G.R.G. a problemas de

controle ótimo (dinâmicos) com restrições não lineares surgiram os trabalhos de Abadie (1972), Abadie e Bichara (1973), Facó (1977) e Drud (1976, 1985). No artigo de Drud (1985) é apresentado um programa computacional CONOPT para resolver problemas dinâmicos com o seguinte modelo

$$\min \sum_{t=1}^{T} x_j^t ,$$

sujeito a

$$h^t(x^t, x^{t-1}, \ldots, x^{t-\tau}) = b^t \quad , \; t = 1, \ldots, T$$

$$\underline{x} \leq x^t \leq \overline{x} \; , \; t = 1, \ldots, T$$

onde $x^t \in E^n$, h^t é uma função não linear, $b^t \in E^m$, x_j^t é a j-ésima componente de x^t, τ é o máximo atraso permitido e T é o instante final do horizonte de tempo considerado. Este artigo é extremamente interessante porque apresenta discussões detalhadas da implementação feita para todos os passos essenciais de um G.R.G. Os diferentes códigos G.R.G. se caracterizam pela implementação particular realizada para esses passos, por exemplo, fatoração das matrizes na resolução dos sistemas lineares, método iterativo usado para resolução dos sistemas de equações não lineares, etc.

Um programa computacional foi implementado no computador

VAX/785 com sistema operacional VMS e compilador FORTRAN 77. Este programa foi aplicado na resolução da etapa 2 do problema de otimização da operação de um sistema hidroelétrico, discutido nos capítulos III e IV.

2. O MÉTODO G.R.G.

No intuito de facilitar a compreensão do método proposto, expomos neste ítem as idéias básicas do método G.R.G.

Seja o problema

$$\min f(x) \tag{1}$$

sujeito a

$$h(x) = b \tag{2}$$

$$\underline{x} \leq x \leq \overline{x} \tag{3}$$

onde $x \in E^n$, $h(x)$ e $f(x)$ são funções de classe $\mathbb{C}^1$, $b \in E^m$, $m \leq n$.

Seja x um vetor que satisfaça (2) e (3). Supõe-se a existência de uma partição de x,

$$x' = (x_B \; , x_S \; , x_N)'$$

tal que

a) x_B tem m componentes, x_S tem s componentes com $0 \leq s$ e x_N consta das n-m-s componentes restantes.

b) $\underline{x}_B < x_B < \bar{x}_B$,

$\underline{x}_S < x_S < \bar{x}_S$,

e $x_N = d$.

onde $d_i = \underline{x}_i$ ou $d_i = \bar{x}_i$.

c) a matriz jacobiana $\nabla_{x_B} h(x_B, x_S, x_N)$ de dimensão m×m é não singular.

Analogamente ao caso de restrições lineares chamamos as variáveis x_B básicas, as x_S superbásicas e as x_N não básicas.

No caso linear, onde as restrições (2) são da forma $Ax = b$ e $A = [B,S,N]$, a matriz $\nabla_{x_B} h(x)$ coincide com B. A hipótese de B ser uma base, permite no caso linear, obter facilmente x_B em função de x_S e x_N como vimos no capítulo anterior.

No caso de restrições não lineares, a hipótese c) garante que numa vizinhança de $x' = (x_B, x_S, x_N)'$ existe uma função $\tilde{h}(x_S)$ tal que $x_B = \tilde{h}(x_S)$. (Teorema das funções implícitas). Então,

localmente teremos que

$$f(x) = f(\tilde{h}(x_S), x_S, d) = \tilde{f}(x_S) \quad (4)$$

O gradiente reduzido com respeito às variáveis x_S (deixando fixas as variáveis x_N) é obtido como segue: (quando não haja dúvida sobre qual é o argumento das funções utilizamos x ou (x_B, x_S, x_N) indistintamente).

$$\nabla_{x_S} \tilde{f}(x_S) = [\nabla_{x_S} \tilde{h}(x_S)]' \nabla_{x_B} f(x) + \nabla_{x_S} f(x) \quad (5)$$

como $h(x_B, x_S, x_N) = b$ e x_N é fixo temos:

$$\nabla_{x_B} h(x) \nabla_{x_S} \tilde{h}(x_S) + \nabla_{x_S} h(x) = 0 \quad (6)$$

$$\nabla_{x_S} \tilde{h}(x_S) + [\nabla_{x_B} h(x)]^{-1} \nabla_{x_S} h(x) = 0 \quad (7)$$

ou

$$\nabla_{x_S} \tilde{h}(x_S) = - [\nabla_{x_B} h(x)]^{-1} \nabla_{x_S} h(x) \quad (8)$$

$$\nabla_{x_S} \tilde{f}(x_S) = \nabla_{x_S} f(x) - [(\nabla_{x_B} h(x))^{-1} \nabla_{x_S} h(x)]' \nabla_{x_B} f(x) \quad (9)$$

O procedimento para resolver problemas com restrições não lineares é análogo ao seguido no caso linear, no sentido de que escolhidas direções de deslocamento para as variáveis x_S, ficam determinadas as direções correspondentes às variáveis dependentes x_B. A diferença é que embora os movimentos de x_S sejam lineares, os movimentos resultantes de x_B não são lineares, devendo satisfazer as restrições (2).

Computacionalmente, isto é obtido deslocando x para $x + \Delta x$ de maneira linear dentro do espaço tangente à superfície definida por $h(x) = b$. Ou seja, se

$$\Delta x' = (\Delta x_B, \Delta x_S, 0)'$$

então

$$\nabla_{x_B} h(x) \Delta x_B = - \nabla_{x_S} h(x) \Delta x_S \quad (10)$$

Este movimento dentro do espaço tangente resulta num ponto $x + \Delta x$ que não necessariamente verifica as restrições $h(x) = b$. Em consequência um procedimento para retornar à região de factibilidade dessas restrições é necessário.

Enumerámos a seguir os passos essenciais de um algoritmo G.R.G. com decomposição LU.

Dados uma solução inicial factível x^0 e $\varepsilon > 0$, fazemos $k = 0$ e iniciamos o processo iterativo expresso na sequência de passos abaixo.

PASSO 1 - Calcular $\nabla h(x^k)$ e $\nabla f(x^k)$

PASSO 2 - Partir $x^k = (x_B^k, x_S^k, x_N^{k\prime})$ de modo que

a) $x_B^k \in E^m$, $x_S^k \in E^s$, $0 \leq s$, $x_N^k \in E^{n-m-s}$

b) $\underline{x}_B < x_B^k < \overline{x}^B$

c) $\underline{x}_S < x_S^k < \overline{x}^S$

d) $B^k = \nabla_{x_B} h(x^k)$ é não singular.

Analogamente, chamamos S^k à matriz $\nabla_{x_S} h(x^k)$ e N^k à matriz $\nabla_{x_N} h(x^k)$.

PASSO 3 - Fatorar $B^k = L^k U^k$.

PASSO 4 - Resolver

$$B^{k\prime} \lambda^k = \nabla_{x_B} f(x^k) \ .$$

PASSO 5 - Calcular o gradiente reduzido

$$\nabla_{x_S} \tilde{f}(x_S^k) = \nabla_{x_S} f(x^k) - S^{k\prime} \lambda^k \qquad (9)$$

Se $\|\nabla_{x_S} \tilde{f}(x_S^k)\| \leq \varepsilon$ significa que x_S^k é um mínimo da função objetivo restrita às variáveis superbásicas atuais. Então vai-se ao Passo 11.

Se $\|\nabla \tilde{f}_S(x_S^k)\| > \varepsilon$ vai-se ao Passo 6.

PASSO 6 - Definir uma direção de deslocamento para x_S^k, Δx_S^k a partir do gradiente reduzido calculado no Passo 5.

PASSO 7 - Calcular

$$\alpha_1 = \max\{\alpha / x_S^k + \alpha \Delta x_S^k \in [\underline{x}_S, \bar{x}^S]\}.$$

PASSO 8 - Achar $\gamma \in [0, \alpha_1]$ tal que

$$\tilde{f}(x_S^k + \gamma \Delta x_S^k) \leq \tilde{f}(x_S^k)$$

PASSO 9 - Resolver

$$B^k \Delta x_B^k = -S^k \Delta x_S^k \quad (11)$$

PASSO 10 - Definir $x_S^{k+1} = x_S + \gamma \Delta x_S^k$

A partir de $\tilde{x}_B^k = x_B^k + \gamma \Delta x_B^k$, mediante algum método corretivo, obter x_B^{k+1} tal que

a) $h(x_B^{k+1}, x_S^{k+1}, x_N^k) = b$

b) $f(x_B^{k+1}, x_S^{k+1}, x_N^k) \leq f(x^k)$

c) $\underline{x}_B < x_B^{k+1} < \bar{x}^B$

Chamamos $x^{k+1} = (x_B^{k+1}, x_S^{k+1}, x_N^k)$

Este passo consiste essencialmente na resolução de um sistema de equações não lineares por algum método iterativo, por exemplo o método de Newton (Luenberger,1984)

Fazer $k = k+1$, ir ao Passo 1.

PASSO 11- Verificar as condições de otimalidade .

$C_{N,j}^k \leq 0$ se $x_{N,j}^k = \bar{x}_{N,j}$

$C_{N,j}^k \geq 0$ se $x_{N,j}^k = \underline{x}_{N,j}$

onde $C_{N,j}^k = (\nabla_{x_N} f(x^k) - N^{k'}\lambda^k)_j$

Por analogia ao caso linear chamamos de custos reduzidos aos $C_{N,j}^k$. Se todos os custos reduzidos não básicos satisfazem à otimalidade x^k é uma solução ótima do problema. Caso contrário escolhemos uma variável não básica que não verifique a condições de otimalidade para entrar no conjunto de variáveis superbásicas. Redefinimos S e N , fazemos $s = s+1$ e vamos ao Pas-5.

Note-se que nos passos 4 e 9 os sistemas lineares que devem

ser resolvidos são os característicos do Método Simplex.

3. ESPECIALIZAÇÃO DO G.R.G. A PROBLEMAS DINÂMICOS

Quando falamos em problemas com restrições não lineares dinâmicas queremos dizer que a matriz jacobiana $\nabla h(x)$ das restrições dos problemas tem estrutura escada.

Da inspeção dos passos do algoritmo do ítem anterior observa-se que no Passo 3 podemos usar as rotinas de fatoração LU específicas para matrizes escada apresentadas neste trabalho.

Os sistemas lineares dos Passos 4 e 9, podem ser resolvidos com as rotinas utilizadas no caso linear.

O Passo 11 é executado usando a rotina VAENBA, discutida no capítulo II.

No Passo 1 é preciso calcular a matriz jacobiana das restrições no ponto x^k. Isto deve ser feito mediante uma subrotina que aproveite o fato de que o número de variáveis que aparecem não linearmente é pequeno.

As colunas da matriz jacobiana correspondente ao ponto inicial são armazenadas da mesma forma que a matriz de restrições do caso linear. A cada iteração esta matriz deve ser recalculada, tomando o cuidado de modificar somente os elementos que correspondem às variáveis que intervêm não linearmente nas restrições.

Para melhor clareza da exposição os Passos 8, 9 e 10 do algoritmo G.R.G. foram enumerados independentemente. Na prática as rotinas que os executam, BUSCA.GRG, VOLFAC e RESID estão interligadas através de um subprocesso iterativo que descrevemos aqui.

Suponhamos ter efetuado até o Passo 7 (inclusive) do G.R.G. e estarmos prontos para executar o Passo 8, ou seja, achar $\gamma \in [0,\alpha_1]$ tal que

$$\tilde{f}(x_S^k + \gamma\Delta x_S^k) \leq \tilde{f}(x_S^k) \ .$$

O processo de busca unidimensional é executado da forma seguinte:

Fazemos incialmente $\theta = \alpha_1$

i) Testamos se

$$\tilde{f}(x_S^k + \theta\Delta x_S^k) \leq \tilde{f}(x_S^k) \tag{11}$$

Para isto é preciso calcular

$$\tilde{f}(x_S^k + \theta\Delta x_S^k) = f(\tilde{h}(x_S^k + \theta\Delta x_S^k),\ x_S^k + \theta\Delta x_S^k) \tag{12}$$

mas para obter (12) é necessário resolver

$$h(x_B\ ,\ x_S^k + \theta\Delta x_S^k\ ,\ x_N) \ = \ b \ . \tag{13}$$

Para resolver o sistema não linear (13) utilizamos a rotina de resolução de sistemas lineares própria para o Passo 10, com ponto inicial

$$\tilde{x}_B = x_B^k + \theta \Delta x_B^k \tag{14}$$

Obtido x_B que satisfaz a equação (13) calculamos (12) e verificamos (11).

Se (11) não é satisfeito, vamos a ii).

Caso contrário, se (11) é satisfeito testamos se $x_B \in [\underline{x}_B, \overline{x}_B]$.

Em caso negativo fazemos $\theta = \theta/2$ e voltamos a i)

Em caso afirmativo pesquisamos se alguma componente de x_B atingiu um dos limites. Se nenhuma variável básica bateu num limite fazemos $\gamma = \theta$.

Se alguma variável básica bateu num limite a declaramos não básica e procuramos uma variável superbásica para substituí-la. Atualizamos a coluna associada à variável superbásica achada com as mesmas rotinas usadas para este fim no caso linear.

Atualizamos a fatoração $L^k U^k$ de B^k, usando também as rotinas próprias do caso linear.

ii) Chegamos a este passo se

$$\tilde{f}(x_S^k + \theta \Delta x_S^k) > \tilde{f}(x^k) \ .$$

Interpolamos uma cúbica entre $(x^k , \tilde{f}(x^k))$ e $(x_S^k + \theta\Delta x_S^k , \tilde{f}(x_S^k + \theta\Delta x_S^k))$ da mesma forma que no capítulo IV. Se μ e o parâmetro que minimiza esta cúbica fazemos $\theta = \mu$ e vá-mos a i).

A resolução do sistema não linear (13) é feita usando um método de Newton modificado. Queremos resolver o sistema (15),

(15) $h(x_B , x_S , x_N) = b$ com $(x_S , x_N)'$ dado.

Então (15) é um sistema de m equações com m incógnitas. O mé-todo de Newton modificado para resolver este sistema é descrito a seguir.

Dados um valor inicial para x_B , seja $x_B^o = \tilde{x}_B$ onde $\tilde{x}_B$ verifica (14), $\varepsilon > 0$, o passo inicial consiste em fazer $j = 0$ e calcular

$$\nabla_{x_B} h(\tilde{x}_B , x_S , x_N) = \tilde{B}$$

Em seguida, tem-se a sequência de cálculo abaixo:

i) Resolver $\tilde{B}\Delta x_B^j = -h(x_B^j , x_S , x_N)$

ii) Fazer $x_B^{j+1} = x_B^j + \Delta x_B^j$, $j = j+1$

iii) Se $\|h(x_B^j , x_S , x_N)\| \leq \varepsilon$, x_B^j é uma solução de (15). Caso contrário ir ao Passo i).

Note-se que em i) $\tilde{B}$ não depende de j, quer dizer, usamos o jacobiano com respeito às variáveis básicas calculado no ponto inicial $\tilde{x}_B$. O sistema de i) também tem estrutura escada e as rotinas discutidas no capítulo II são usadas na sua resolução.

4. EXPERIÊNCIA COMPUTACIONAL

No capítulo IV resolvemos a etapa 1 (fase térmica) do problema de planejamento da operação de um sistema hidroelétrico apresentado no capítulo III. Obtivemos uma solução para o seguinte problema:

$$\min \sum_{t=0}^{41} (\bar{g}(u^t) - d^t)^2 \tag{16}$$

sujeito a

$$x_1^{t+1} = x_1^t + y_1^t - u_1^t - v_1^t \tag{17}$$

$$x_2^{t+1} = x_2^t + y_2^t + u_1^{t-\tau} + v_1^{t-\tau} - u_2^t - v_2^t - u_3^t - v_3^t \tag{18}$$

$$i = 1,2 \; ; \; t = 0,1,\ldots,41$$

$$\underline{x}_i \leq x_i \leq \bar{x}^i \quad , \quad i = 1,2 \; ; \; t = 1,2,\ldots,42 \tag{19}$$

$$\underline{u}_i \leq u^t \leq \bar{u}^t \quad , \quad i = 1,2,3 \; ; \; t = 0,1,\ldots,41 \tag{20}$$

(21) $$v_i^t \geq 0 \quad , \quad i = 1,2,3 \; ; \; t = 0,1,\ldots,41$$

onde

(22) $$\bar{g}(u) = -0.00853u_1^2 + 8.3u_1 + 32.98u_2 + 34.39u_3 \; .$$

Queremos resolver agora a etapa 2 (fase hidráulica) utilizando a função (22) como função de geração total de energia. Lembramos que a etapa 2 consiste em maximizar o volume de água de Sobradinho no instante final. O problema é formulado da seguinte forma:

$$\max \; x_i^{42}$$

sujeito a (17), (18), (19), (20), (21), (22) e

(23) $$\bar{g}(u^t) = d^t \; .$$

Este problema foi rodado no computador VAX/785 com sistema operacional VMS usando o compilador FORTRAN 77. Uma solução ótima foi obtida em 25 iterações usando em média 2 segundos de C.P.U. por iteração.

CONCLUSÕES

Neste trabalho mostramos como pode ser eficientemente aproveitada a estrutura escada das matrizes associadas às restrições em problemas de otimização.

Introduzimos um novo método de atualização das fatorações LU de uma matriz com essa estrutura e implementamos um programa computacional (SIMESC) que consiste na adaptação do Método Simplex para problemas lineares cujas restrições têm estrutura escada. Este programa foi testado em alguns problemas. Esta experiência computacional preliminar aponta para boas perspectivas de utilização da metodologia proposta.

Mostramos também a viabilidade da implementação de algoritmos do tipo gradiente reduzido e gradiente reduzido generalizado que utilizem as técnicas desenvolvidas. Algumas aplicações e experiências computacionais neste sentido são apresentadas.

A construção destes programas computacionais nos levou a conviver com muitos detalhes que são essenciais na preparação de pacotes de otimização, como MINOS, (Murtagh e Saunders, 1977) e MPSX (Benichou e outros, 1977).

O programa implementado, SIMESC, pode ainda ser aperfeiçoado. Vários assuntos de pesquisa podem ser considerados nesta linha.

i) Técnicas de escalamento dos coeficientes das matrizes de restrições;

ii) Estudo do comportamento do algoritmo em relação à propagação de erros de arredondamento com o intuito de definir parâmetros e tolerâncias variáveis;

iii) Extensão do algoritmo de maneira a considerar a esparsidade dentro dos blocos da escada;

iv) Métodos de cálculo dos custos reduzidos (pricing) específicos para estrutura escada.

REFERÊNCIAS BIBLIOGRÁFICAS

ABADIE, J. e CARPENTIER, J. (1965). Généralisation de la méthode du gradient réduit de Wolfe au cas de contraintes non-linéaires, Note HR 6678, Elétricité de France, Paris.

ABADIE, J. e CARPENTIER, J. (1969). "Generalization of the Wolfe reduced-gradient method to the case of nonlinear constraints", em Optimization (R. Fletcher, ed.), pag. 37-49, Academic Press, London and New York.

ABADIE, J. e GUIGOU, J. (1970). "Numerical experiments with the GRG method", em Integer and Nonlinear Programming (J. Abadie, ed.), pag. 529-536, North-Holland, Amsterdam.

ABADIE, J. (1972). "Application of the GRG algorithm to optimal control problems", em Nonlinear and Integer Programming (Abadie, J. ed.) pag. 191-211, North-Holland, Amsterdam.

ABADIE, J. e BICHARA, M. (1973). Resolution numérique de certains problèmes de commande optimale, R.A.I.R.O., 7, V-2, pag. 77-105.

ABADIE, J. (1978). "The GRG method for nonlinear programming" , em Design and Implementation of Optimization Software (H. J. Greenberg, ed.), pag. 335-362, Sijthoff and Noordhoff, Netherlands.

AONUMA, T. (1978). A two-level algorithm for two-stage linear programs, Journal of the Operations Research Society of Japan 21, pag. 171-187.

ARMENTANO, V.A. (1979). "Programação linear dinâmica" , Tese de Mestrado, DEE - FEC - UNICAMP.

BARTELS, R.H., e GOLUB, G.H. (1969). The simplex method using LU decomposition, Comm. ACM 12, pag. 266-268.

BARTELS, R.H. (1971). A stabilization of the simplex method, Numerishe Mathematik 16, pag. 414-434.

BENICHOU, M., GAUTHIER, J.M., HENTGES, G. e RIBIÈRE, G. (1977). The efficient solution of large-scale linear programming problems - some algorithmic techniques and computational results, Mathematical Programming 13, pag. 280-322.

CARDOSO, M.A. (1981) "Um modelo de programação linear para a operação semanal de um parque hidroelétrico", Tese de Mestrado, Depto. de Matemática Aplicada, IMECC-UNICAMP.

CARNEIRO DA SILVA, M.C. (1984). "Modelo de Otimização para a operação hidroenergética da cascata do São Francisco", Tese de Mestrado, DEE - FEC - UNICAMP.

COLVILLE, A.R. (1968). A comparative study on nonlinear programming codes, Report Nº 320-2949, IBM, New York. Scientific Center.

DANTZIG, G.B. (1949). Programming of Interdependents Activities, II, Econométrica, (July-October).

DANTZIG, G.B. (1955). Upper bound, secondary constraints and block triangularity in linear programming. Econométrica 23, pag. 174-183.

DANTZIG, G.B. e WOLFE, P. (1960). Decomposition Principle for linear programs, Operations Research 8, pag. 101-111.

DANTZIG, G.B. (1963). "Compact Basis triangularization for the Simplex Method", em Recent Advances in Mathematical Programming, (R.L. Graves and P. Wolfe (eds), pag. 125-132, Mc Graw-Hill Book Co., New York.

DANTZIG, G.B. (1973). Solving staircase linear programs by a nested block-angular method, Technical Report 73-1, Dept. of Operations Research, Stanford University.

DRUD, A. (1976). "Optimization in large partly nonlinear systems" em Optimization Techniques. Modeling and optimization in

the service of Man, Part 2, Lecture Notes in Computer Science, vol. 41, (J.Cea, ed.) pag. 312-329, Springer-Verlag, Berlin, Heidelberg, New York.

DRUD, A. (1985). Conopt: A GRG code for large sparse dynamic non-linear optimization problems, Mathematical Programming 31, pag. 153-191.

FACÓ, J.L.D. (1977). Commande Optimale des systèmes dynamiques non linéaires à retards avec contraintes d'inéglités sur l'état et la commande, Comunicação GSC-06/77, Dpto. de Engenharia Elétrica, PUC, RJ.

FRIEDLANDER, A., MEDINA, E.L., TAVARES, H.M.F. (1982). A method for Solving Linear Dynamic Programming Problems, XI International Symposium on Mathematical Programming, Bonn, Alemanha.

FRIEDLANDER, A., MEDINA, E.L., TAVARES, H.M.F. (1982). Algoritmo para Programação Linear Dinâmica, IV Congresso Brasileiro de Automática, Campinas, Brasil.

FORREST, J.J.H. e TOMLIN, J.A. (1972). Updating triangular factors of the basis to mantain sparsity in the product form simplex method, Mathematical Programming, 2, pag. 263-278.

FORSYTHE, G.E. e MOLER, C.B. (1967). "Gaussian elimination and LU decomposition" em Computer Solution of Linear

algebraic systems, pag. 27-33, Prentice Hall, Inc., Englewood Cliffs, N.J.

FOURER, R. (1979). Sparse Gaussian elimination of staircase linear systems, Technical Report SOL 79-17, Systems Optimization Laboratory, Dept. of Operations Research, Stanford University.

FOURER, R. (1982). Solving staircase linear programs by the simplex method, 1: inversion, Mathematical Programming 23, pag. 274-313.

FOURER, R. (1983). Solving staircase linear programs by the simplex method, 2: Pricing, Mathematical Programming 25, pag. 251-292.

FOURER, R. (1984). Staircase Matrices and Systems, SIAM Review, vol. 26, Nº 1, pag. 1-70.

GILL, P.E. MURRAY, W. e WRIGHT, M.H. (1981). Practical Optimization, Academic Press Inc. (London).

GLASSEY, C.R. (1971). Dynamic linear programs for production scheduling, Operations Research 1, pag. 46-56.

GLASSEY, C.R. (1973). Nested decomposition and multi-stage linear programs. Management Science 20, pag. 282-292.

GRINOLD, R.C. (1972). Steepest ascent for large scale linear programs, SIAM Review 14, pag. 447-464.

HEESTERMAN, A.R.G. e SANDEE, J. (1965). Special Simplex Algorithm for linked problems, Management Science 11, pag.420-428.

HELLERMAN , E. e RARICK, O. (1971). Reinversion with the preassigned pivot procedure, Mathematical Programming 1, pag. 195-216.

HELLERMAN, E. e RARICK, D. (1972). "The partitioned preassigned pivot procedure (P^4)", em Sparse Matrices and their Applications (D.J. Rose and R.A. Willoughby, eds), pag. 67--76, Plenum Press, New York.

HO, J.K. e MANNE, A.S. (1974). Nested decomposition for Dynamic Models, Mathematical Programming 6, pag, 121-140.

LASDON, L.S. (1970). "Compact inverse methods", em Optimization Theory for Large Systems, pag. 304-318, The Macmillan Co, New York.

LASDON, L.S. e WAREN, A.D. (1978). "Generalized reduced gradient software for linearly and nonlinearly constrained problems", em Design and Implementation of Optimization Software (H.J. Greenberg,ed), pag.335-362, Sijthoff and Noordhoof, Netherlands.

LYRA, C., FRIEDLANDER, A. e GEROMEL, J.C. (1982)."Coordenação da Operação Energética no Médio São Francisco por um Método de Gradiente Reduzido", Matemática Aplicada e Computacional, vol. 1, nº 2, pag. 107-120.

LUENBERGER, D.G. (1984). Linear and Nonlinear Programming, Addison-Wesley Publishing Company, Reading, Massachusetts.

MARKOWITZ, H.M. (1957). The elimination form of the inverse and its application to linear programming, Management Science 3, pag. 255-269.

MURTAGH, B.A. e SAUNDERS, M.A. (1977). MINOS User's Guide, Report SOL 77-9, Department of Operations Research, Stanford University.

MURTAGH, B.A. e SAUNDERS, M.A. (1978). Large-scale linearly constrained optimization, Mathematical Programming 14, pag. 41-72.

MURTAGH, B.A. (1981). Advanced Linear Programming, Mc Graw - Hill Book Co., New York.

NAG Fortran Library Reference Manual (Mark 8) (1981). Numerical Algorithms Group Limited, Oxford, England.

ORCHARD-HAYS, W. (1968). Advanced Linear Programming Computing Techniques, Mc Graw-Hill, New York.

ORCHARD-HAYS, W. (1978a). "History of mathematical programming

systems" em Design and Implementation of Optimization Software, (H.J. Greenberg, ed), pag. 1-26, Sijthoff and Noordhoff, Netherlands.

ORCHARD-HAYS, W. (1978b). "Scope of mathematical programming software" em Design and Implementation of Optimization Software, (H.J. Greenberg, ed), pag.27-40, Sijthoff and Noordhoff, Netherlands.

ORCHARD-HAYS, W. (1978c). "Anatomy of a mathematical programming systems" em Design and Implementation of Optimization Software, (H.J. Greenberg, ed), pag. 41-102, Sijthoff and Noordhoff, Netherlands.

PEROLD, A. e DANTZIG, G.B. (1978). A basis factorization method for block triangular linear programs, Technical Report SOL 78-7, Systems Optimization Laboratory, Department of Operations Research, Stanford University.

PISSOLATO, E.L.M. (1982). "Método para resolver um problema de programação linear dinâmica", Tese de Mestrado, FEC - - UNICAMP.

PROPOI, A. e KRIVONOZHKO, V. (1978). The Simplex Method for Dynamic Linear Programs - Report RR-78-14 - Int. IASA, Laxenburg, Austria.

REID, J.K. (1975). A sparsity-exploiting variant of the Bartels--Golub decomposition for linear programming bases, Report CSS 20, Atomic Energy Research Establishment, Harwell,England.

REID, J.K. (1976). Fortran subroutines for handling sparse linear programming bases, Report R 8269, Atomic Energy Research Establishment, Harwell, England.

ROSEN, J.B. (1960). The gradient projection method for nonlinear programming, Part I - linear constraints, SIAM Journal of Applied Mathematics 8, pag. 181-217.

ROSEN, J.B. (1961). The gradient projection method for nonlinear programming, Part II - nonlinear constraints, SIAM Journal of Applied Mathematics 9, pag. 514-532.

SAUNDERS, M. (1976). "A fast stable implementation of the simplex method using Bartels-Golub updating" em Sparse Matrix Computations, (Bunch, J.R. e Rose, D.J. ed), pag. 213-226. Academic Press, New York.

WOLFE, P. (1962). The reduced-gradient method: manuscrito não publicado. The Rand Corporation.

WOLLMER, R.D. (1977). A substitute inverse for the basis of a staircase structure linear program, Mathematics of Operations Research 2, pag. 230-239.

WOOD, M.K. e DANTZIG G.B. (1949). Programming of interdependent Activities I, Econométrica (July-October).