

ALGORITMOS PARA RESOLUÇÃO DE  
SISTEMAS NÃO LINEARES  
ESPARSOS  
TARCÍSIO LATERZA LOPIES

Orientador

Prof. Dr. José Mário Martinez

Dissertação apresentada no Instituto  
de Matemática, Estatística e Ciência  
da Computação, UNICAMP, como requisi-  
to parcial para obtenção do título /  
de Mestre em Matemática Aplicada.

Maio, 1982

**UNICAMP**  
**BIBLIOTECA CENTRAL**

Aos meus pais.

## AGRADECIMENTOS

Ao Martínez, pela proposta desse trabalho, pelo apoio e orientação constantes.

A todos os amigos que, de uma forma ou de outra, me ajudaram a realizar esse trabalho.

À Maria do Carmo, por sua ajuda, carinho e apoio.

Ao Laboratório de Matemática Aplicada, em cujo contexto este trabalho foi realizado.

## ÍNDICE

INTRODUÇÃO.....	01
CAPÍTULO I - MÉTODO DE NEWTON E MÉTODOS QUASE-NEWTON PARA RESOLUÇÃO DE SISTEMAS NÃO LINEARES.....	02
Método de Newton.....	02
Métodos Quase-Newton.....	03
CAPÍTULO II - RESOLUÇÃO DE SISTEMAS LINEARES ATRAVÉS DE DE- COMPOSIÇÃO LU.....	07
Introdução.....	07
O Método.....	07
CAPÍTULO III - ARMAZENAMENTO DA DECOMPOSIÇÃO LU DE UMA MA - TRIZ ESPARSA.....	13
Matrizes a serem armazenadas.....	13
Armazenamento e operação das matrizes L, U, P e $R^{-1}$ .....	13
CAPÍTULO IV - ALGORITMOS PROPOSTOS.....	18
CAPÍTULO V - EXPERIÊNCIAS NUMÉRICAS E CONCLUSÕES.....	23
Experiências realizadas.....	23
Conclusões.....	25
CAPÍTULO VI - REFERÊNCIAS.....	27
APÊNDICE - DOCUMENTAÇÃO DA SUBROTINA SNLESP.....	29
Propósito.....	29
Parâmetros.....	29
Subrotinas providas pelo usuário.....	32
Alternativas de métodos.....	34
Tamanho do programa.....	35
Precisão.....	35
Acesso à subrotina SNLESP.....	35
Exemplo de programa principal.....	36
Data.....	39

## INTRODUÇÃO

Neste trabalho são apresentados dois algoritmos para resolução de sistemas não lineares esparsos.

No capítulo I os métodos mais conhecidos são descritos e discutidos.

No capítulo II um método para fatorização LU de matrizes esparsas é apresentado.

No capítulo III é explicado o armazenamento na memória do computador de matrizes esparsas e de sua decomposição LU.

O capítulo IV apresenta os dois algoritmos propostos para resolução de sistemas não lineares esparsos.

No capítulo V são apresentadas as experiências numéricas e conclusões. As experiências foram realizadas no computador PDP-10 da UNICAMP, e o programa foi feito em FORTRAN-10 e sua documentação, disponível no Laboratório de Matemática Aplicada encontra-se no apêndice deste trabalho.

I. MÉTODO DE NEWTON E MÉTODOS QUASE-NEWTON PARA RESOLUÇÃO DE SISTEMAS NÃO LINEARES

I.1 Método de Newton

O problema consiste em achar  $x^* \in \Omega \subset \mathbb{R}^n$ , tal que  $F(x^*)=0$ , onde:

$$\left\{ \begin{array}{l} F : \Omega \rightarrow \mathbb{R}^n \\ F = (f_1 \dots f_n)^T \\ f_i : \Omega \rightarrow \mathbb{R} \\ \Omega \text{ aberto} \\ F \in C^1(\Omega) \end{array} \right. \quad (1.1)$$

Dado  $x^k \in \Omega$ , uma aproximação da solução  $x^*$  de 1.1 e desenvolvendo-se por Taylor, obtemos :

$$\text{onde } \left\{ \begin{array}{l} F(x) \simeq F(x^k) + F'(x^k)(x - x^k) = L(x) \\ F'(x^k) = \begin{pmatrix} \frac{\partial f_i}{\partial x_j} & (x^k) \end{pmatrix} \end{array} \right. = \text{matriz jacobiana de } F \text{ no ponto } x^k$$

O método de Newton consiste em tomar como a - aproximação  $x^{k+1}$ , a solução de  $L(x)=0$ , ou seja :

$$\text{onde } \left\{ \begin{array}{l} x^{k+1} = x^k + \Delta x^k \\ F'(x^k) \Delta x^k = -F(x^k) \end{array} \right. \quad (1.2)$$

Se  $n$  é muito grande mas o jacobiano é esparsa, podemos permutar linhas e colunas de  $F'(x^k)$  a cada iteração, antes de resolver 1.2. Estas permutações objetivam reduzir o aparecimento de elementos não nulos durante o processo de eliminação de Gauss, necessário à resolução de 1.2. Uma técnica de permutação é explicada em [8].

Uma segunda opção seria permutar a matriz de estrutura esparsa  $E$  do jacobiano, antes de se iniciarem as iterações. A matriz  $E$  se define assim:

$$\left\{ \begin{array}{l} E = (E_{ij}) \\ F'(x) = (F'_{ij}) \\ E_{ij} = 0 \Leftrightarrow F'_{ij} = 0, \forall x \in \Omega \\ E_{ij} \neq 0 \text{ caso contrário} \end{array} \right.$$

As permutações aplicadas em E seriam aplicadas ao jacobiano calculado a cada iteração, antes de resolver 1.2. Uma terceira opção seria a utilização de métodos iterativos para resolver 1.2.

Cada iteração do método de Newton exige o cálculo do jacobiano e a resolução de um sistema linear. Uma tentativa de baratear cada iteração dá origem ao método de Newton com Refinamentos, que consiste em utilizar o mesmo jacobiano em p+1 iterações consecutivas, ou seja, a cada iteração do método de Newton seguem-se p subiterações onde o jacobiano não muda, logo não precisa ser calculado. O número ótimo de subiterações pode ser calculado usando noções teóricas [1,10,14] ou práticas [11].

## 1.2 MÉTODOS QUASE-NEWTON [2,4,5]

Estes métodos surgiram da tentativa de baratear o cálculo do jacobiano e de sua inversa, mas conservando as propriedades essenciais do método de Newton. Consiste em substituir o jacobiano por uma matriz  $B_{k+1}$ , não singular, que satisfaça :

$$B_{k+1} \Delta x^k = \Delta F^k = F(x^{k+1}) - F(x^k) \quad (1.3)$$

propriedade esta que seria satisfeita se F fosse linear e  $B_{k+1}$  o jacobiano verdadeiro. Nesse caso, 1.2 é substituído por :

$$\text{onde } \begin{cases} x^{k+1} = x^k + \Delta x^k \\ B_k \Delta x^k = -F(x^k) \end{cases}$$

O que diferencia um método quase-Newton de outro é a fórmula utilizada para calcular  $B_k$ , pois existem infinitas matrizes que satisfazem 1.3. As fórmulas mais conhecidas são :

- a fórmula de Broyden Boa [2,7,12,13,15]

$$B_{k+1} = B_k + (\Delta F^k - B_k \Delta x^k) (\Delta x^k)^T / (\Delta x^k)^T \Delta x^k \quad (1.4)$$

- a fórmula de Broyden Ruim [2,7,12,13]

$$B_{k+1} = B_k + (\Delta F_k - B_k \Delta x^k) (\Delta F^k)^T B_k / (\Delta F^k)^T B_k \Delta x^k$$

As principais características destas fórmulas são :

a) as matrizes  $B_{k+1}$  se obtêm através de atualizações de posto um à matriz  $B_k$ , ou seja,

$$B_{k+1} = B_k + \Delta B_k, \quad \Delta B_k = v w^T \quad (v, w \in R^n)$$

b) a estrutura esparsa de  $B_k$  não é conservada

c) a fórmula 1.4 é a única solução do problema de minimização :

$$\text{Min} \left\{ \| B - B_k \|_F : B \Delta x^k = \Delta F^k \right\}$$

onde  $\| \cdot \|_F$  é a norma de Frobenius

$$\| A \| = \sqrt{\sum_{i=1}^n \sum_{j=1}^n a_{ij}^2}$$

Outra fórmula desenvolvida por Broyden [3,6] conserva a estrutura esparsa do jacobiano e é apropriada para sistemas esparsos de grande porte. A fórmula é :

$$B_{k+1} = B_k + \Delta B_k \quad (1.5)$$

$$\Delta B_k = \sum_{i=1}^N \left[ (S_i \Delta x^k)^T (S_i \Delta x^k) \right] + e_i^T (\Delta F^k - B_k \Delta x^k) e_i (S_i \Delta x^k)^T$$

onde  $S_i$  são operadores de projeção "esparsa", que projeta  $\Delta x^k$  ortogonalmente (norma de Frobenius) nos subespaços  $Z_i$  ( $i=1, \dots, n$ ), onde



$$Z_i = \left\{ v \in \mathbb{R}^n : \begin{array}{l} e_j^T v = 0 \text{ para todo } j \text{ tal que} \\ F'_{1j} = 0, \text{ para todo } x \in \Omega \end{array} \right\}$$

$Z_i \subset \mathbb{R}^n$  é o subespaço que identifica a estrutura esparsa da  $i$ -ésima linha do jacobiano.

Então o subespaço  $Z \in \mathbb{R}^{n \times n}$  que identifica a estrutura esparsa do jacobiano é definido por :

$$Z = \left\{ A \in \mathbb{R}^{n \times n} : A^T e_i \in Z_i \text{ para } i=1, \dots, n \right\}$$

Na fórmula 1.5,  $(\cdot)^+$  é a inversa generalizada. Para um escalar  $a$ ,  $a^+ = 0 \Leftrightarrow a = 0$  e  $a^+ = a^{-1} \Leftrightarrow a \neq 0$

Características desta fórmula :

- as matrizes  $B_{k+1}$  são obtidas de  $B_k$  com um custo proporcional ao número de elementos distintos de zero de um membro de  $Z$
- as matrizes  $B_{k+1}$  são obtidas por atualizações de posto  $n$ .
- a estrutura esparsa do jacobiano é mantida.
- a fórmula 1.5 é a única solução de :

$$\begin{aligned} \text{Min } & \left\{ \|B - B_k\|_F : B \in Y \cap Z \right\} \\ \text{onde } Y = & \left\{ A \in \mathbb{R}^{n \times n} : A \Delta x^k = \Delta F^k \right\} \end{aligned}$$

A desvantagem desta fórmula é que não se obtém diretamente a inversa de  $B_{k+1}$  a partir de  $B_k$ . Sendo assim, a fórmula 1.3 é resolvida a cada iteração, fatorizando-se a matriz  $B_{k+1}$ .

J. E. Dennis e Earl S. Marwil [6] desenvolveram outra fórmula baseada na decomposição  $L_k U_k$  da matriz  $P_k B_k$ , sendo  $P_k$  a matriz de permutação de linhas resultante da estratégia de pivotamento parcial ;  $L_k \in \mathcal{L}_k$  sendo este último um subespaço afim de matrizes  $n \times n$  triangulares inferiores ;  $U_k \in \mathcal{U}_k$  um sub-

espaço  $n \times n$  de matrizes triangulares superiores.

Assume-se que  $\mathcal{L}_k$  e  $\mathcal{U}_k$  refletem a estrutura esparsa de  $P_k A = LU$  para  $A \in Z$ .  $P_k$  pode ser escolhida para afetar a esparsidade de  $\mathcal{L}_k$  e  $\mathcal{U}_k$ .

A cada iteração, a fórmula mantém  $L_k$  constante e atualiza  $U_k$  pela atualização esparsa de Broyden. Ou seja,

$$\begin{aligned} P_{k+1} &= P_k \\ L_{k+1} &= L_k \\ U_{k+1} &= U_k + \Delta U_k \end{aligned} \quad (1.6)$$

$$\Delta U_k = \sum_{i=1}^n \left\{ \left[ (S_i \Delta x^k)^T (S_i \Delta x^k) \right] + e_i^T w e_i (S_i \Delta x^k)^T \right\}$$

$$\text{Onde } \begin{cases} S_i \text{ são operadores de projeção "esparsa" em } \mathcal{U}_k \\ w = v_k - U_k \Delta x^k \\ v_k = L_k^{-1} \Delta F^k \end{cases}$$

$$\text{Neste caso } P_{k+1} B_{k+1} = L_{k+1} U_{k+1}.$$

Características desta fórmula :

- as matrizes  $U_{k+1}$  são obtidas de  $U_k$  com um custo proporcional ao número de elementos não nulos de um membro de  $\mathcal{U}_k$
- a estrutura esparsa da decomposição LU do jacobiano é mantida
- a fórmula 1.6 resolve :

$$\text{Min } \left[ \begin{array}{l} \|U - U_k\|_F : U \in \mathcal{U}_k \text{ é o ponto mais próximo} \\ \text{em } \mathcal{U}_k \text{ de } Y \end{array} \right]$$

- a fórmula 1.3 é resolvida a cada iteração sem a necessidade de se fatorizar a matriz  $B_{k+1}$ , apenas resolvendo dois sistemas triangulares.

## II. RESOLUÇÃO DE SISTEMAS LINEARES ATRAVÉS DE DECOMPOSIÇÃO LU

### II.1 Introdução

A decomposição LU de uma matriz quadrada não singular é uma técnica baseada na eliminação de Gauss e consiste em decompor a matriz em fatores triangulares, L (triang. inferior) e U (triang. superior), tendo um deles diagonal unitária. A decomposição é obtida através de operações entre linhas (fatoração por colunas) ou colunas (fatoração por linhas). No caso de sistemas não lineares, a matriz a ser decomposta é o jacobiano do sistema, que é mais simples de ser calculado por linhas, motivo pelo qual estudaremos a decomposição LU por linhas.

### II.2 O Método

Aplicam-se operações e permutações entre colunas da matriz de tal forma a reduzi-la a uma matriz triangular inferior. As permutações e operações entre colunas são realizadas através de pós-multiplicação por matrizes de permutação simples e por matrizes elementares, como será visto a seguir.

Uma matriz de permutação simples  $Q_p$  é uma matriz identidade com a p-ésima linha (ou coluna) permutada com alguma linha (ou coluna) j. Quando pré-multiplicada por uma matriz A, o resultado é a matriz A com as colunas p e j permutadas.

Exemplo :

$$Q_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad \begin{array}{l} \text{( colunas 2 e 4 permutadas)} \\ p = 2 \quad j = 4 \end{array}$$

$$A = \begin{pmatrix} 11 & 12 & 13 & 14 \\ 21 & 22 & 23 & 24 \\ 31 & 32 & 33 & 34 \\ 41 & 42 & 43 & 44 \end{pmatrix}$$

$$AQ_2 = \begin{pmatrix} 11 & 14 & 13 & 12 \\ 21 & 24 & 23 & 22 \\ 31 & 34 & 33 & 32 \\ 41 & 44 & 43 & 42 \end{pmatrix}$$

Uma matriz elementar  $T_p$  é uma matriz identidade com a parte à direita da diagonal da  $p$ -ésima linha preenchida com elementos escolhidos de tal maneira a zerar a parte à direita da  $p$ -ésima linha de uma matriz  $A$ .

Estes elementos são calculados pela fórmula :

$$T_{p,ij} = (-a_{ij} / a_{ii}) \quad \text{para } i = p \\ j = i+1, \dots, n$$

A matriz  $T_p$  deve ser pré-multiplicada pela matriz  $A$ . Exemplos:

$$A = \begin{pmatrix} 8 & 5 & 0 & 3 \\ 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \\ 1 & 7 & 6 & 2 \end{pmatrix}$$

$$T_1 = \begin{pmatrix} 1 & -5/8 & 0 & -3/8 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$AT_1 = \begin{pmatrix} 8 & 0 & 0 & 0 \\ 2 & 7/4 & 4 & 17/4 \\ 6 & 13/4 & 8 & 27/4 \\ 1 & 51/8 & 6 & 13/8 \end{pmatrix}$$

$$T_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -9/8 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$AT_3 = \begin{pmatrix} 8 & 5 & 0 & 3 \\ 2 & 3 & 4 & 1/2 \\ 6 & 7 & 8 & 0 \\ 1 & 7 & 6 & -19 \end{pmatrix}$$

As matrizes de permutações simples servem para escolher o elemento pivô. A escolha do pivô é feita através de dois parâmetros, TOLPIV e TOLABS, e pode ser explicada através do seguinte algoritmo :

- a)  $\ell = \text{Arg} \max_{k \geq p} |a_{pk}|$   
 Se  $|a_{p\ell}| < \text{TOLABS}$ , pare (matriz singular)
- b)  $m = p$
- c)  $m = m + 1$
- d) Se  $|a_{pm}| < \text{TOLABS}$  ou  $\left| \frac{a_{pm}}{a_{p\ell}} \right| < \text{TOLPIV}$ , vá para c)
- e)  $\text{PIVO} = a_{pm}$
- f)  $Q_p$  = matriz identidade com as colunas p e m permutadas

A fatoração realiza-se através da seguinte

fórmula :

$$AQ_1^T Q_2^T \dots Q_{n-1}^T = L$$

onde

- $n$  = ordem da matriz A
- $L$  = matriz triangular inferior
- $Q_p$  = matrizes de permutação simples, determinadas pela escolha do pivô
- $T_p$  = matrizes elementares, utilizadas para zerar a parte à direita da p-ésima linha transformada de A

Exemplo : (  $\text{TOLABS} = 1.E-10$  ;  $\text{TOLPIV} = 0.5$  )

$$A = \begin{pmatrix} 3 & 6 & 0 & 12 \\ 5/2 & 5 & 8 & 34 \\ 1 & 0 & 31/3 & 1 \\ 18/5 & 3 & 8/3 & 11 \end{pmatrix}$$

$p = 1$  ;  $\ell = 4$  ;  $m = 2$  ;  $\text{PIVO} = a_{1,2} = 6$

$$Q_1 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$AQ_1 = \begin{pmatrix} 6 & 3 & 0 & 12 \\ 5 & 5/2 & 8 & 34 \\ 0 & 1 & 31/3 & 1 \\ 3 & 18/5 & 8/3 & 11 \end{pmatrix}$$

$$T_1 = \begin{pmatrix} 1 & -1/2 & 0 & -2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$AQ_1 T_1 = \begin{pmatrix} 6 & 0 & 0 & 0 \\ 5 & 0 & 8 & 24 \\ 0 & 1 & 31/3 & 1 \\ 3 & 21/10 & 8/3 & 5 \end{pmatrix}$$

$$p = 2 ; \quad \ell = 4 ; \quad m = 4 ; \quad \text{PIVO} = a'_{2,4} = 24$$

$$Q_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$AQ_1 T_1 Q_2 = \begin{pmatrix} 6 & 0 & 0 & 0 \\ 5 & 24 & 8 & 0 \\ 0 & 1 & 31/3 & 1 \\ 3 & 5 & 8/3 & 21/10 \end{pmatrix}$$

$$T_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1/3 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$AQ_1 T_1 Q_2 T_2 = \begin{pmatrix} 6 & 0 & 0 & 0 \\ 5 & 24 & 0 & 0 \\ 0 & 1 & 10 & 1 \\ 3 & 5 & 1 & 21/10 \end{pmatrix}$$

$$p = 3 ; \quad \ell = 3 ; \quad m = 3 ; \quad \text{PIVO} = a'_{3,3} = 10$$

$$Q_3 = I = \text{Matriz Identidade } 4 \times 4$$

$$AQ_1T_1Q_2T_2Q_3 = \begin{pmatrix} 6 & 0 & 0 & 0 \\ 5 & 24 & 0 & 0 \\ 0 & 1 & 10 & 1 \\ 3 & 5 & 1 & 21/10 \end{pmatrix}$$

$$T_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1/10 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$AQ_1T_1Q_2T_2Q_3T_3 = \begin{pmatrix} 6 & 0 & 0 & 0 \\ 5 & 24 & 0 & 0 \\ 0 & 1 & 10 & 0 \\ 3 & 5 & 1 & 2 \end{pmatrix} = L$$

$$AQ_1T_1Q_2T_2Q_3T_3 = L$$

$$A = LT_3^{-1}Q_3^{-1}T_2^{-1}Q_2^{-1}T_1^{-1}Q_1^{-1}$$

$$\text{Mas } Q_p^{-1} = Q_p \text{ (f\u00e1cil verifica\u00e7\u00e3o)}$$

$$T_p^{-1} = \begin{pmatrix} 1 & 0 & \dots & \dots & 0 \\ 0 & 1 & \dots & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 1 & -T_{p,p,p} & \dots & -T_{p,p,n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & \dots & 1 \end{pmatrix} \quad \leftarrow \text{linha } p$$

$$\text{Logo } A = L (T_3^{-1}Q_3^{-1}T_2^{-1}Q_2^{-1}T_1^{-1}Q_1^{-1})$$

P\u00f3s-multiplicando os dois termos da igualdade por  $Q_1Q_2Q_3$ ,

obtemos :

$$AQ_1Q_2Q_3 = L (T_3^{-1}Q_3^{-1}T_2^{-1}Q_2^{-1}T_1^{-1}Q_1^{-1})Q_1Q_2Q_3 = LU$$

Fazendo as contas :

$$U = \begin{pmatrix} 1 & 2 & 0 & 1/2 \\ 0 & 1 & 1/3 & 0 \\ 0 & 0 & 1 & 1/10 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Assim, conseguimos encontrar L (triangular inferior) e U (triangular superior unitária) tais que  $AQ_1Q_2Q_3 = LU$ .

Generalizando, temos :

$$AQ_1 \dots Q_{n-1} = LU = L(T_{n-1}^{-1}Q_{n-1} \dots T_1^{-1}Q_1Q_1 \dots Q_{n-1})$$

Se matriz A, antes de ser fatorizada, for submetida a trocas de linhas e colunas, a fórmula fica :

$$A = P^{-1}LUQ_{n-1} \dots Q_1Q^{-1}$$

onde  $\left\{ \begin{array}{l} P \text{ é a matriz de permutação das linhas de A (matriz} \\ \text{identidade com linhas permutadas)} \\ Q \text{ é a matriz de permutação das colunas de A (matriz} \\ \text{identidade com colunas permutadas)} \\ PAQ \text{ é a matriz após a troca de linhas e colunas, an-} \\ \text{tes de ser fatorizada} \end{array} \right.$

Para resolvermos o sistema linear  $Ax=b$ , realizamos as seguintes etapas :

- a) resolva  $Ly=Pb$
- b) resolva  $Uz=y$
- c) faça  $x=Q_1Q_2 \dots Q_{n-1}z$



### III- ARMAZENAMENTO DA DECOMPOSIÇÃO LU DE UMA MATRIZ ESPARSA

#### III.1 Matrizes a serem armazenadas

Inicialmente temos a aproximação do jacobiano permutado  $PB_k Q$ . Após a decomposição temos  $B_k = P^{-1} L U Q_{n-1} \dots Q_1 Q^{-1}$ . Para a resolução de um sistema linear, precisamos de  $P$  e  $R$  ( $R = Q Q_1 \dots Q_{n-1}$ ). Para a decomposição, precisamos de  $P$  e  $Q$ .

Portanto, precisamos armazenar  $P^{-1}$  (ou  $P$ ),  $L$ ,  $U$ ,  $Q$  e  $R$  (ou  $R^{-1}$ ).

As matrizes de permutação têm uma inversa fácil de se calcular (é igual à transposta) implicitamente. Não é necessário armazenar a matriz  $B_k$ , pois a cada linha calculada desta se obtém a linha correspondente de  $L$  e  $U$ .

#### III.2 Armazenamento e operação das matrizes $L$ , $U$ , $P$ e $R^{-1}$

##### a) Armazenamento de $L$ e $U$

- Vetor real  $VEU$  de dimensão  $NMEDZ$  ( $n^2$  de elementos distintos de zero após a fatorização)

Armazena os valores das matrizes  $L$  e  $U$ , por linha, alternadamente (primeiro a  $U$ ).

Exemplo :

$$U = \begin{pmatrix} 1 & 3 & 2 \\ & 1 & 5 \\ & & 1 \end{pmatrix} \quad L = \begin{pmatrix} 4 & & \\ 1 & 3 & \\ 5 & 4 & 1 \end{pmatrix}$$

$$\text{VETLU} : ( \underbrace{3, 2, 1}_{1^a}, \underbrace{4}_{1^a}, \underbrace{5, 1}_{2^a}, \underbrace{1, 3}_{2^a}, \underbrace{1}_{3^a}, \underbrace{5, 4, 1}_{3^a} )$$

linha da U    linha da L    linha da U    linha da L    linha da U    linha da L

Obs : Os elementos de uma linha da L estarão ordenados por coluna de maneira crescente. Os elementos de uma linha da U virão em qualquer ordem, mas o último elemento sempre pertencerá à diagonal.

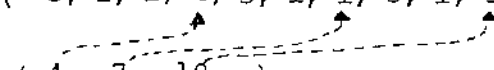
- Vetor inteiro APONTL de dimensão N

Aponta para o início de cada linha da L, no vetor VETLU

Exemplo :

$$\text{VETLU} : ( 3, 2, 1, 4, 5, 1, 1, 3, 1, 5, 4, 1 )$$

$$\text{APONTL} : ( 4, 7, 10 )$$



- Vetor inteiro APONTU de dimensão N

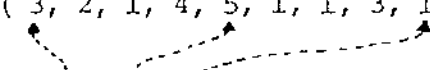
Aponta para o início de cada linha da U, no vetor VETLU

Exemplo :

$$\text{APONTL} : ( 4, 7, 10 )$$

$$\text{VETLU} : ( 3, 2, 1, 4, 5, 1, 1, 3, 1, 5, 4, 1 )$$

$$\text{APONTU} : ( 1, 5, 9 )$$



- Vetor inteiro COLUNS de dimensão NMEDZ

Indica a coluna a que pertence cada elemento do vetor

VETLU

Exemplo :

$$\text{APONTL} : ( 4, 7, 10 )$$

$$\text{APONTU} : ( 1, 5, 9 )$$

$$\text{VETLU} : ( 3, 2, 1, 4, 5, 1, 1, 3, 1, 5, 4, 1 )$$

$$\text{COLUNS} : ( \begin{matrix} \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ 2, & 3, & 1, & 1, & 3, & 2, & 1, & 2, & 3, & 1, & 2, & 3 \end{matrix} )$$

b) Armazenamento das matrizes  $P$ ,  $Q$ ,  $R$  e  $R^{-1}$

A matriz  $P$  é fornecida pelo usuário através do vetor inteiro  $P$ , de dimensão  $n$ . Neste caso,  $P(I)=J$  indica que a  $I$ -ésima linha de  $PB_kQ$  corresponde à  $J$ -ésima linha da matriz  $B_k$ .

Exemplo :

$$B_k = \begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix} \quad P = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} e_3^T \\ e_1^T \\ e_2^T \end{pmatrix}$$

$$PB_k = \begin{pmatrix} 31 & 32 & 33 \\ 11 & 12 & 13 \\ 21 & 22 & 23 \end{pmatrix}$$

Então o vetor  $P$  será : ( 3 , 1 , 2 )

A pré-multiplicação de um vetor  $b$  pela matriz  $P$  resultando no vetor  $c$  se fará pelo seguinte algoritmo :

Para  $I=1$  até  $N$  faça começo

$$IND = P(I) / DESLOC$$

$$C(I) = B(IND)$$

fim

Obs: Por motivos de maior compactação de dados, o conteúdo que especifica a matriz  $P$  está armazenado na metade esquerda de cada posição do vetor  $P$ . Para recuperá-lo, basta dividir a posição do vetor  $P$  pelo valor  $DESLOC$  ( no caso do programa implementado,  $DESLOC = 2^{17}$  ).

Exemplo :

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \quad Pb = \begin{pmatrix} b_2 \\ b_3 \\ b_1 \end{pmatrix}$$

Vetor P : ( 2, 3, 1 )

Seguindo o algoritmo :

$$I = 1 \ ; \ IND = 2 \ ; \ C(1) = B(2)$$

$$I = 2 \ ; \ IND = 3 \ ; \ C(2) = B(3)$$

$$I = 3 \ ; \ IND = 1 \ ; \ C(3) = B(1)$$

O usuário também fornece a matriz Q, através do vetor Q de dimensão n. Neste caso, Q(I)=J indica que a I-ésima coluna de  $PB_k Q$  corresponde à J-ésima coluna da matriz  $B_k$ .

Exemplo :

$$B_k = \begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix} \quad Q = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = (e_2 \ e_3 \ e_1)$$

$$PB_k = \begin{pmatrix} 12 & 13 & 11 \\ 22 & 23 & 21 \\ 32 & 33 & 31 \end{pmatrix}$$

Então o vetor Q será : ( 2, 3, 1 )

Após a fatoração LU da matriz  $B_k$ , o armazenamento das matrizes de permutação estará na seguinte situação :

Vetor P :  $\begin{cases} \text{lado direito - matriz R} \\ \text{lado esquerdo - matriz P} \end{cases}$

Vetor Q :  $\begin{cases} \text{lado direito - matriz R} \\ \text{lado esquerdo - matriz Q} \end{cases}$

A pós-multiplicação da matriz R por um vetor x resultando o vetor y se fará através de um dos seguintes algoritmos :

1º) Para I=1 até N faça começo

$$\text{IND} = \text{Q(I)} \text{ .AND. MASKRA}$$

$$\text{Y(IND)} = \text{X(I)}$$

fim

2º) Para I=1 até N faça começo

$$\text{IND} = \text{P(I)} \text{ .AND. MASKRA}$$

$$\text{Y(IND)} = \text{X(I)}$$

fim

Obs : A expressão  $\text{P(I)} \text{ .AND. MASKRA}$  serve para pegar o lado direito da I-ésima posição do vetor P ( no caso do programa implementado,  $\text{MASKRA} = 2^{17} - 1$  ).

Exemplo :

$$R = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = ( e_2 \ e_3 \ e_1 )$$

$$R^{-1} = R^T = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} = ( e_3 \ e_1 \ e_2 )$$

$$Rx = \begin{pmatrix} x_3 \\ x_1 \\ x_2 \end{pmatrix}$$

Vetor P : ( 2, 3, 1 ) - metade direita

Vetor Q : ( 2, 3, 1 ) - metade direita

Seguindo os algoritmos :

$$I = 1 \ ; \ \text{IND} = 3 \ ; \ \text{Y(2)} = \text{X(1)}$$

$$I = 2 \ ; \ \text{IND} = 1 \ ; \ \text{Y(3)} = \text{X(2)}$$

$$I = 3 \ ; \ \text{IND} = 2 \ ; \ \text{Y(1)} = \text{X(3)}$$

IV ALGORITMOS PROPOSTOS :

A fórmula 1.6 proposta por Dennis & Marwil se caracteriza pela decomposição LU da matriz  $B_k$  e pela atualização do fator que não tem diagonal unitária, a cada iteração do método. Se considerarmos que a fatorização é feita por linhas, o fator com diagonal unitária será a matriz triangular superior U. Neste caso, a fórmula 1.6 modificada para o caso de fatorização por linhas e generalizada para o caso da matriz  $B_k$  ser multiplicada à esquerda e à direita por matrizes de permutação, ficaria :

$$\begin{aligned}
 B_k &= P_k^{-1} L_k U_k R_k^{-1} \\
 P_{k+1} &= P_k \\
 R_{k+1} &= R_k \\
 U_{k+1} &= U_k \\
 L_{k+1} &= L_k + \Delta L_k
 \end{aligned} \tag{4.1}$$

$$\Delta L_k = \sum_{i=1}^n \{ ((S_i U_k y_k)^T (S_i U_k y_k)) + e_i^T w_i e_i (S_i U_k y_k)^T \}$$

$$\text{onde } \begin{cases} S_i & \text{são operadores de projeção "esparsa" em } \mathcal{U}_k \\ Y_k &= R_k^{-1} \Delta x^k \\ w_k &= P_k \Delta F^k - L_k U_k y_k \end{cases}$$

O primeiro algoritmo proposto é a atualização da matriz triangular que possui diagonal unitária (no caso da fatorização por linhas, é a U), fazendo com que esta matriz perca a unitariedade da sua diagonal nas iterações onde se realiza a atualização. Neste caso, o sub-espaço afim  $\mathcal{U}_k$  é formado pelas matrizes triangulares superiores nxn que refletem a estrutura esparsa de  $U_k$ , sem a restrição de diagonal unitária. A fórmula, então, seria idêntica à fórmula 1.6, generalizada para o caso de  $B_k$  ter linhas e colunas permutadas :

$$\begin{aligned}
B_k &= P_k^{-1} L_k U_k R_k^{-1} \\
P_{k+1} &= P_k \\
R_{k+1} &= R_k \\
U_{k+1} &= U_k + \Delta U_k \\
\Delta U_k &= \sum_{i=1}^n \{ ((S_i y_k)^T (S_i y_k)) + e_{i,k}^T v_i e_i (S_i y_k)^T \}
\end{aligned} \tag{4.2}$$

onde  $\left\{ \begin{array}{l} S_i \text{ são operadores de projeção "esparsa" em } \mathcal{U}_k \\ Y_k = R_k^{-1} \Delta x^k \\ v_k = L_k^{-1} P_k \Delta F^k - U_k Y_k \end{array} \right.$

O segundo algoritmo proposto é a alternância na atualização, ou seja, em algumas iterações se utiliza a fórmula 4.1 e, em outras, a fórmula 4.2. O critério para escolha entre modificar a L ou a U pode ser o mesmo adotado por L.S.Ochi e J.M.Martínez em [12,13], ou seja realiza-se a modificação que minimize  $\| B_{k+1} \Delta x^{k-1} - \Delta F^{k-1} \|$ .

Para a fórmula 4.1, temos :

$$\begin{aligned}
& \| B_{k+1} \Delta x^{k-1} - \Delta F^{k-1} \| = \\
& = \| P_k^{-1} L_{k+1} U_k R_k^{-1} x^{k-1} - F^{k-1} \| \\
& = \| P_k^{-1} (L_k + \Delta L_k) U_k R_k^{-1} \Delta x^{k-1} - \Delta F^{k-1} \| \\
& = \| P_k^{-1} L_k U_k R_k^{-1} \Delta x^{k-1} + P_k^{-1} \Delta L_k U_k R_k^{-1} \Delta x^{k-1} - \Delta F^{k-1} \| \\
& = \| B_k x^{k-1} - P_k^{-1} \Delta L_k U_k R_k^{-1} \Delta x^{k-1} - \Delta F^{k-1} \| \\
& = \| \Delta F^{k-1} - P_k^{-1} \Delta L_k U_k R_k^{-1} \Delta x^{k-1} - \Delta F^{k-1} \| \\
& = \| P_k^{-1} \Delta L_k U_k R_k^{-1} \Delta x^{k-1} \| \\
& = \| \Delta L_k U_k R_k^{-1} \Delta x^{k-1} \|
\end{aligned}$$

L.S.O

Para a fórmula 4.2, temos :

$$\left\| B_{k+1} \Delta x^{k-1} - \Delta F^{k-1} \right\| = \left\| L_k \Delta U_k R_k^{-1} \Delta x^{k-1} \right\|$$

Como este critério é muito trabalhoso, optou-se pela alternância simples, ou seja, se na iteração  $k$  houve atualização da  $L$ , então na iteração  $k+1$  haverá atualização da  $U$  e vice-versa.

Foi implementado um programa escrito em FORTRAN-10, no computador PDP-10 da UNICAMP, e sua documentação encontra-se no apêndice. O algoritmo deste programa é explicado a seguir :

- a) Inicializações
- b) Impressão da iteração
- c) Teste de convergência ou fracasso
- d) Decide :
 

}	1- fatorização
}	2- refatorização
}	3- atualização da $L$
}	4- atualização da $U$
}	5- conservação da fatorização
- e) Se a decisão foi 1 ou 2, acha fatorização  $LU$  do jacobiano. Se foi 3 ou 4, atualiza o fator correspondente através da fórmula 4.1 ou 4.2.
- f) Acha  $\Delta x^k$ ,  $x^{k+1}$ ,  $F(x^{k+1})$ ,  $\Delta F^k$  e  $\|F(x^{k+1})\|$
- g)  $k \leftarrow k + 1$
- h) Atualiza as variáveis  $XMIN$ ,  $FXMIN$  e  $NITMIN$
- i) Vá para b.

Explicações sobre cada passo :

a) Inicializações

Entre outras variáveis, inicializa :

$NFAILU=NIT=NITMIN=0$

$XK=X$

$FXK=FXMIN=F(XK)$

$NORMFK=NORMIN=\|FXK\|$

$DIST=DISIFX * NORMIN$



onde :  $\left\{ \begin{array}{l} \text{NIT} = \text{n}^{\circ} \text{ da itera\~{c}\~{a}o ( K )} \\ \text{NFATLU, NITMIN, X, NORMIN, FXMIN} - \text{Ver explica\~{c}\~{a}o no ap\~{e}ndice} \\ \text{DISTFX} - \text{Ver \u00edtem c.} \end{array} \right.$

b) Impress\~{a}o da itera\~{c}\~{a}o

Imprime-se, a cada IMPR itera\~{c}\~{a}o, o n\u00famero da itera\~{c}\~{a}o, o tipo da itera\~{c}\~{a}o e a norma de F(XK). Se IMPR=0, nada \u00e9 impresso. IMPR \u00e9 um par\u00e2metro de entrada, descrito no ap\u00eêndice.

c) Teste de converg\u00eancia ou fracasso :

Declara-se converg\u00eancia (CODERR=0) quando :

$$\text{NORMFK} < \text{EPSLON}$$

ou

$$\text{NORMFK} < \sqrt{\text{EPSLON}} \text{ e } \|\Delta x^k\| < \text{EPSLON} * (1 + \|\text{XK}\|)$$

Declara-se diverg\u00eancia ou fracasso (CODERR=5) quando  $\text{NORMFK} > \text{DIST}$

O programa tamb\u00e9m para (CODERR=4) se foi excedido o n\u00famero m\u00e1ximo de itera\~{c}\~{a}o ( $\text{NIT} > \text{NITMAX}$ ). CODERR \u00e9 um par\u00e2metro de sa\u00edda, descrito no ap\u00eêndice. EPSLON e NITMAX s\u00e3o par\u00e2metros de entrada, tamb\u00e9m descritos no ap\u00eêndice.

d) Decis\u00e3o sobre a matriz  $B_{k+1}$

Depende dos par\u00e2metros de entrada QUAL, FREQ e REFAT, exceto se NIT=0, neste caso a op\~{c}\~{a}o \u00e9 a fatoriza\~{c}\~{a}o. Nas outras itera\~{c}\~{a}o, os seguintes testes s\u00e3o feitos :

Se  $\text{MOD}(\text{NIT}, \text{REFAT})=0$  , refatoriza\~{c}\~{a}o

Se  $\text{MOD}(\text{NIT}+1, \text{FREQ})=0$ , atualiza\~{c}\~{a}o da L (se  $\text{QUAL}=1$  ou se  $\text{QUAL}=3$  e a \u00faltima atualiza\~{c}\~{a}o foi da U) ou da U (se  $\text{QUAL}=2$  ou se  $\text{QUAL}=3$  e a \u00faltima atualiza\~{c}\~{a}o foi da L).

Se nenhum dos testes foi satisfeito : conserva\~{c}\~{a}o da fatoriza\~{c}\~{a}o  
No caso de atualiza\~{c}\~{a}o da L ou da U, pode acontecer um crescimento excessivo de um dos elementos atualizados, ou a equa\~{c}\~{a}o da f\u00f3rmula 1.3 n\u00e3o ficar satisfeita. Neste caso, haver\u00e1 uma refatoriza\~{c}\~{a}o.

e)

f) Cálculo diversos vetores

O cálculo de DELTAX ( $\Delta x^k$ ) se faz através do seguinte algoritmo:

$$1) \text{ DELTAX} = - B_k^{-1} \cdot \text{FXK}$$

$$2) \text{ NORMDX} = \|\text{DELTAX}\|$$

$$3) \text{ Se } \text{NORMDX} > \text{DISTX} \text{ então faça } \begin{cases} \text{DELTAX} = \frac{\text{DELTAX}}{\text{NORMDX}} * \text{DISTX} \\ \text{NORMDX} = \text{DISTX} \end{cases}$$

O terceiro passo do algoritmo caracteriza o controle do passo, feito através do parâmetro de entrada DISTX.

Os outros cálculos são :

$$\text{XK} = x^{k+1} = \text{XK} + \text{DELTAX}$$

Acha F(XK)

$$\text{DELTA F} = \text{FXK} - F(\text{XK})$$

$$\text{FXK} = F(\text{XK})$$

$$\text{NORMFK} = \|\text{FXK}\|$$

g)

h) Atualizações

$$\text{Se } \|\text{FXK}\| < \text{NORMIN} \text{ então faça } \begin{cases} \text{NITMIN} = \text{NIT} \\ \text{XMIN} = \text{XK} \\ \text{FXMIN} = \text{FXK} \end{cases}$$

## V. EXPERIÊNCIAS NUMÉRICAS E CONCLUSÕES

### V.1 Experiências realizadas

Foram testadas as duas funções utilizadas em [3,6], que são :

$$1 - f_i(x) \begin{cases} = (3-k_1 x_i) x_{i+1} - x_{i-1} - 2x_{i+1}, & 1 < i < n \\ = (3-k_1 x_i) x_{i+1} - x_{i-1}, & i = n \\ = (3-k_1 x_i) x_{i+1} - 2x_{i+1}, & i = 1 \end{cases}$$

$$2 - f_i(x) = (k_1 - k_2 x_i^2) x_{i+1} - k_3 \sum_{j \in B_i} (x_j + x_j^2), \quad i=1, \dots, n$$

onde  $B_i = \{i_1, \dots, i_2\} - \{i\}$ , sendo  $i_1 = \max(1, i-r_1)$  e  $i_2 = \min(n, i+r_2)$ . Os parâmetros  $k_1, k_2, k_3$  variam para aumentar a não-linearidade do problema, enquanto que  $r_1$  e  $r_2$  determinam a largura da faixa da matriz jacobiano. O chute inicial foi  $x_i = -1$ ,  $i=1, \dots, n$ . O critério de convergência utilizado foi  $\|F\|_{\infty} < 10^{-6}$  (observe que não foi incluído o segundo teste de convergência, especificado no capítulo anterior). Foram rodados os 23 casos especificados em [3,6], e aqui reproduzidos na tabela I. Observe a existência de mais 3 casos, dois deles (24 e 25) se referindo à função 1, apenas variando o tamanho do problema. O caso 26 se refere à função abaixo :

$$3 - f_i(x) = x_i^2 + \sum_{j \in N_i} x_j - N_i - 1$$

onde  $N_i$  é um conjunto de  $N_i$  índices entre 1 e  $n$ , e diferentes de  $i$  gerados aleatoriamente para cada linha. Neste caso, o chute inicial foi  $x_i = 0.5$ ,  $i=1, \dots, n$ . O jacobiano inicial e de cada iteração de refatorização foi o analítico, fornecido pela rotina JACOB.

Na tabela II encontram-se os resultados das rodadas, sendo que a coluna BROYDEN ESPARSO e DENNIS-MARWIL foram tiradas de [6]. A coluna NEWTON indica o método de Newton, que se obtém fazendo

REFAT=1 ; a coluna NEWTON C/ REF. indica o método de Newton com refinamentos, que se obtém fazendo REFAT= $\infty$  e FREQ= $\infty$  ; a coluna MUDA SO L indica o método de Dennis-Marwil modificado (fórmula 4.1), que se obtém fazendo QUAL=1, FREQ=1 e REFAT= $\infty$  ; a coluna MUDA SO U indica o primeiro algoritmo proposto (fórmula 4.2), que se obtém fazendo QUAL=2 e FREQ e REFAT iguais ao caso anterior ; a coluna MUDA L E U indica o segundo algoritmo proposto, que é obtido fazendo QUAL=3 e FREQ e REFAT iguais ao caso anterior.

Os casos assinalados com \* foram aqueles que não convergiram.

Por não se dispor de uma rotina de pré-assinalamento de pivôs, adotou-se  $P(I)=I$  e  $Q(I)=I$  (matrizes de permutação iguais à identidade).

As colunas NIT, NEFATLU, MUTILL, MUTILU e EME indicam o número de iterações, número de fatorizações LU, número de elementos diferentes de zero na L e na U no pior caso (soma máxima) e o valor do parâmetro de entrada EME, respectivamente. Em nenhum dos casos rodados houve controle do passo e o valor do parâmetro TOLPIV foi  $10^{-3}$ . Finalmente, o valor do parâmetro DISTFX foi  $10^3$ . Houve permutações de colunas apenas no caso 26 e as refatorizações foram devidas a crescimento excessivo da L ou da U.

O programa teve quatro versões. A versão zero equivale ao primeiro programa desenvolvido. As versões seguintes apareceram devido à necessidade de diminuir o tempo de CPU no caso 25, principalmente, que demorava mais de 27 minutos de CPU para executar, quando se utilizava o segundo algoritmo proposto. A partir deste momento, ficou claro que um programa para problemas de grande porte não poderia ser codificado como se fosse um programa comum. Na versão zero, um vetor auxiliar era zerado a cada vez que se chamava a rotina JACOB, o que equivale a ter um código do tipo abaixo (considerando o caso 25) :

```

DO 10 I=1,6000
DO 10 J=1,6000
10 A(J)=0

```

Este conjunto de instruções leva 4 minutos de CPU para executar. No caso de se rodar o método de Newton, isto seria executado 4 vezes, o que daria 16 minutos só para zerar um vetor. Como este vetor era utilizado para armazenar uma linha do jacobiano, nas versões posteriores, o vetor era zerado uma vez a cada decomposição LU, a partir daí, apenas as posições preenchidas eram zeradas quando não eram mais necessárias. Algumas outras melhorias foram incorporadas, o que levou a uma economia de 7 minutos de CPU no caso da utilização do segundo método proposto, como pode ser visto na tabela III.

Nas versões zero e um, o teste para escolher o elemento pivô era feito em todos os elementos da linha. Nas versões posteriores, o teste foi feito apenas para os elementos contidos entre o primeiro e último diferente de zero da linha. Outra melhoria feita simultaneamente foi a eliminação da chamada das rotinas que realizam permutações no caso em que estas não acontecem, ou seja, no caso em que  $P(I)=I$  ou  $R(I)=I$ . Na última versão outras melhorias foram feitas, o que levou a um tempo final de 7 minutos de CPU, uma economia de 75%.

## V.2 Conclusões

Como pode ser visto na tabela III, o método de Newton foi o mais demorado de todos. Como pode ser visto na tabela II, o método de Broyden esparsa levaria mais tempo ainda, pois também envolve uma decomposição LU a cada iteração e, nos casos rodados, o cálculo do jacobiano é barato de se calcular. Resta-nos analisar os outros métodos. Os métodos DENNIS-MARWIL, MUDA SO L e MUDA SO U são praticamente iguais (nos casos rodados), com ligeira vantagem para MUDA SO U, que sempre ganha ou empata com os outros dois, exceto no caso 9, em que a L tem mais elementos diferentes de zero que a U.

O método MUDA L E U pareceu o mais instável de todos, mas é o que ganha na maior parte dos casos, o que leva a crer que

o desenvolvimento de um critério mais barato para se escolher entre mudar a L ou a U numa certa iteração deve compensar.

O método de Newton com refinamento levou um tempo de execução parecido com os 4 últimos citados anteriormente, mas falhou no caso 26, exatamente por não prever refatorização nas primeiras iterações.

TABELA I

CASO	FUNÇÃO	N	K <sub>1</sub>	K <sub>2</sub>	K <sub>3</sub>	R <sub>1</sub>	R <sub>2</sub>	N <sub>1</sub>
1	1	5	0.1	-	-	-	-	-
2	1	5	0.5	-	-	-	-	-
3	1	10	0.5	-	-	-	-	-
4	1	20	0.5	-	-	-	-	-
5	1	600	0.5	-	-	-	-	-
6	1	600	2	-	-	-	-	-
7	2	100	1	1	1	3	3	-
8	2	100	1	1	1	2	4	-
9	2	100	1	1	1	5	1	-
10	2	50	1	1	1	5	5	-
11	2	50	2	1	1	5	5	-
12	2	50	1	2	1	5	5	-
13	2	50	3	2	1	5	5	-
14	2	50	2	3	1	5	5	-
15	2	50	3	3	1	5	5	-
16	2	50	2	2	1	5	5	-
17	2	50	1	2	2	5	6	-
18	2	50	2	2	2	5	5	-
19	2	50	2	3	2	5	5	-
20	2	50	2	4	1	5	5	-
21	2	50	2	5	1	5	5	-
22	2	50	3	4	1	5	6	-
23	2	50	3	5	1	5	5	-
24	1	2000	0.5	-	-	-	-	-
25	1	6000	0.5	-	-	-	-	-
26	3	100	-	-	-	-	-	9

TABELA II

CASO	NEWTON	NEWTON C/ REF(NFATLU=1)	MUDA SÓ L		MUDA SÓ U		MUDA L e U		BROYDEN ESPARSO	DENNIS- - MARWIL		EME	MUTILL	MUTILU
	NIT=NFATLU	NIT	NIT	NFATLU	NIT	NFATLU	NIT	NFATLU	NIT=NFATLU	NIT	NFATLU			
1	3	8	6	1	5	1	6	1	5	5	1	10 <sup>3</sup>	9	9
2	3	7	5	1	5	1	5	1	4	5	1	10 <sup>3</sup>	9	9
3	3	11	6	1	6	1	5	1	5	6	1	10 <sup>3</sup>	19	19
4	4	14	7	1	6	1	6	1	5	6	1	10 <sup>3</sup>	39	39
5	4	15	7	1	6	1	6	1	5	6	1	10 <sup>3</sup>	1199	1199
6	4	15	9	1	7	1	6	1	7	8	1	10 <sup>3</sup>	1199	1199
7	4	10	7	1	6	1	6	1	6	7	1	10 <sup>3</sup>	394	394
8	4	10	7	1	6	1	6	1	6	7	1	10 <sup>3</sup>	297	490
9	4	12	6	1	7	1	6	1	6	7	1	10 <sup>3</sup>	585	199
10	3	9	7	1	6	1	6	1	6	7	1	10 <sup>3</sup>	285	285
11	4	13	8	1	8	1	6	1	7	8	1	10 <sup>3</sup>	285	285
12	4	13	9	1	8	1	7	1	8	8	1	10 <sup>3</sup>	285	285
13	4	21	6	2	11	1	6	2	9	12	1	10 <sup>3</sup>	285	285
13	4	21	12	1	11	1	9	1	9	12	1	10 <sup>4</sup>	285	285
14	4	21	6	2	7	2	9	1	11	11	1	10 <sup>3</sup>	285	285
14	4	21	11	1	7	2	9	1	11	11	1	10 <sup>4</sup>	285	285
14	4	21	11	1	11	1	9	1	11	11	1	10 <sup>5</sup>	285	285
15	5	25	15	1	12	1	11	1	11	13	1	10 <sup>3</sup>	285	285
16	4	17	8	2	10	1	9	1	9	10	1	10 <sup>3</sup>	285	285
16	4	17	10	1	10	1	9	1	9	10	1	10 <sup>4</sup>	285	285



TABELA II cont.

CASO	NEWTON	NEWTON C/ REF(NFATLU=1)	MUDA SÓ L		MUDA SÓ U		MUDA L e U		BROYDEN ESPARSO	DENNIS- - MARWIL		EME	MUTILL	MUTILU
	NIT=NFATLU	NIT	NIT	NFATLU	NIT	NFATLU	NIT	NFATLU	NIT=NFATLU	NIT	NFATLU			
17	4	15	12	2	10	1	17	2	8	10	1	10 <sup>3</sup>	285	285
17	4	15	25	1	10	1	18	1 *	8	10	1	10 <sup>3</sup>	285	285
18	4	14	11	1	9	1	11	1	8	9	1	10 <sup>3</sup>	285	285
19	4	16	10	1	9	1	8	1	9	10	1	10 <sup>3</sup>	285	285
20	4	25	8	2	12	1	8	2	12	13	1	10 <sup>3</sup>	285	285
20	4	25	13	2	12	1	10	1	12	13	1	10 <sup>4</sup>	285	285
20	4	25	13	1	12	1	10	1	12	13	1	10 <sup>5</sup>	285	285
21	5	28	15	2	7	2	10	2	13	14	1	10 <sup>3</sup>	285	285
21	5	28	15	1	13	1	25	2	13	14	1	10 <sup>4</sup>	285	285
22	5	29	11	2	11	2	8	2	13	14	1	10 <sup>3</sup>	285	285
22	5	29	15	1	14	1	11	1	13	14	1	10 <sup>4</sup>	285	285
23	5	33	17	1	9	2	9	2	13	16	1	10 <sup>3</sup>	285	285
23	5	33	17	1	16	1	18	2 *	13	16	1	10 <sup>4</sup>	285	285
23	5	33	17	1	16	1	21	1 *	13	16	1	10 <sup>5</sup>	285	285
24	4	15	7	1	6	1	6	1	-	-	-	10 <sup>3</sup>	3999	3999
25	4	15	7	1	6	1	6	1	-	-	-	10 <sup>3</sup>	11999	11999
26	3	13 *	4	2	4	2	4	2	-	-	-	10 <sup>3</sup>	3822	3833

TABELA III

VERSAO	CASO	NEWTON	NEWTON C/ REFINAMENTOS	MUDA SÓ L	MUDA SÓ U	MUDA L e U
0	5	85.3	24.8	23.6	23.3	23.5
3	5	20.5	8.5	8.9	8.6	8.5
0	6	86.2	25.4	26.5	25.5	25.4
3	6	21.5	8.9	9.3	8.8	8.1
0	24	-	-	224.3	232.4	225.3
2	24	-	-	79	76.6	77.3
3	24	170	53.8	54.4	52.7	56.8
0	25	-	-	-	-	1663
1	25	-	-	-	-	1220
3	25	1422	330	381	394	376
3	26	31	15	23	23	23

VI REFERÊNCIAS

1. R. P. Brent, "Some efficient algorithms for solving systems of nonlinear equations", SIAM J. Numer. Anal. , 10, 327-344 (1973)
2. C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations", Math. Comp. 19, 577-593 (1965)
3. \_\_\_\_\_, "The convergence of an algorithm for solving sparse nonlinear systems", Math. Comp. 25, 285-294 (1971)
4. J. E. Dennis, "A brief introduction to Quasi-Newton methods", Cornell University, Computer Science Department, TR 77-327 (1977)
5. \_\_\_\_\_ e J. J. Moré, "Quasi-Newton methods, motivation and theory", SIAM Review 19, 46-89 (1977)
6. \_\_\_\_\_ e E. S. Marwil, "Direct secant updates of Matrix factorizations", Technical Report Of Rice University, Department of Mathematical Sciences (1981)
7. D. M. Gay, "Some convergence properties of Broyden's method", SIAM J. Numer. Anal. 16, 623-630 (1979)
8. E. Hellerman and D. C. Rarick, "Reinversion with the preassigned pivot procedure", Math. Prog. 1, 195-216 (1971)
9. J. M. Martinez, "Algoritmos para resolução numérica de sistemas não lineares", Atas do XIII Colóquio Brasileiro de Matemática, 1981
10. \_\_\_\_\_, "Generalization of the methods of Brent and Brown for solving nonlinear simultaneous equations", SIAM J. Numer. Anal. 16, 434-448 (1979)

11. J. M. Martinez, "Solving nonlinear simultaneous equations with a generalization of Brent's method", BIT 20, 501-510 (1980)
12. \_\_\_\_\_ e L. S. Ochi, "Sobre dois métodos de Broyden", Mat. Apl. e Comput. , Vol. 1, Nº 2, 1982
13. L. S. Ochi, "Algoritmos de Broyden combinados para resolução de sistemas não lineares", Tese de Mestrado, Departamento de Matemática Aplicada, UNICAMP, 1981
14. A. N. Ostrowski, Solution of equations in Euclidean and Banach spaces, Academic Press, New York and London, 1973
15. M. J. D. Powell, "A hybrid method for nonlinear equations", Numerical methods for nonlinear equations, editado por P. Rabinovitz, Gordon and Breach, 1970

APÉNDICE

"SNLESP"

(Subrotina; Fortran-10)

1.-Propósito

Resolver um sistema não linear de equações algébricas :

$$f_1(x_1, \dots, x_n) = 0$$

$$\vdots$$

$$f_n(x_1, \dots, x_n) = 0$$

de preferência com jacobiano esparsos.

Utiliza-se métodos Quase-Newton com atualizações esparsas. De acordo com os parâmetros QUAL, FREQ e REFAT, pode-se utilizar o método de Newton, o método de Newton com refinamento, o método de Dennis & Moré (ligeiramente modificado) ou um dos métodos introduzidos por Lopes, através dos parâmetros acima. Pode-se utilizar também uma combinação destes métodos dando a estes parâmetros valores apropriados ( ver Ítem 5).

2.-Parâmetros

O comando de chamada de SNLESP deve ser do seguinte tipo:

```
CALL SNLESP (N, MEMOR, DISTX, DISTFX, EPSLON, TOLPIV,
* TOLABS, LME, NITMAX, IMPR, SAIDA, QUAL, FREQ, REFAT,
* P, Q, F, JACOB, X, TRAB, FXMIN, NIT, NITMIN, MUTIL,
* MUTILU, NFATLU, CODERR)
```

onde :

N - Número de funções e de variáveis do sistema

MEMOR - Dimensão do vetor TRAB. Deve ser pelo menos  $8N + 2(NL + NU) + 1$ , onde  $NL(NU) = N^{\circ}$  estimado de elementos diferentes de zero da  $L(U)$ , incluindo a diagonal.

- DISTX - Limite superior estimativo para a distância entre o ponto inicial e a solução ( norma suprema da diferença dos dois pontos).  
Serve para a realização de controle do passo. Ver referência no ítem 4.
- DISTFX - Limite superior para a razão entre as normas de  $F(x^{k+1})$  e  $F(x^0)$ . Serve para determinação de divergência. Ver referência no ítem 4.
- EPSLON - Precisão desejada na solução. O programa declarará convergência quando a norma suprema de  $F(x)$  for menor do que EPSLON ou quando, simultaneamente, a norma suprema de  $F(x)$  for menor que  $\sqrt{\text{EPSLON}}$  e a razão entre as normas supremas de  $\Delta x^k$  e  $x^k$  acrescida de um for menor que EPSLON. Ver referência no ítem 4.
- TOLPIV - Tolerância relativa de pivô, utilizada durante a decomposição LU do jacobiano. Ver referência no ítem 4.
- TOLABS - Tolerância absoluto de pivô, utilizada durante a decomposição LU do jacobiano. Ver referência no ítem 4.
- EME - Crescimento relativo máximo de cada elemento da L ou da U, durante uma atualização da fatorização do jacobiano.
- NIIMAX - Número máximo de iterações permitido.
- IMPR - Parâmetro de controle de impressão : se =0 , não imprime nada ; se  $\neq 0$ , imprime a norma de  $F(x_0)$  e diagnóstico de cada iteração múltipla de | IMPR | .
- SAIDA - Unidade de saída para impressão dos diagnósticos.
- QUAL - Controle de atualização da fatorização da aproximação do jacobiano. Se =1, atualiza só a L ; se =2, atualiza só a U ; se =3, atualiza ambas, alternadamente.
- FREQ - Frequência de atualização da fatorização. Atualiza a fatorização nas iterações múltiplas de FREQ. O fator a ser atualizado depende do parâmetro QUAL. O parâmetro REFAT tem prioridade sobre o parâmetro FREQ.

- REFAT - Frequência de refatorização. Se  $\text{MOD}(K+1, \text{REFAT})=0$ , então a fatorização anterior é esquecida e realiza-se nova decomposição LU do jacobiano, antes de se calcular  $\Delta x$ .
- P - Vetor especificando a matriz de permutação das linhas do jacobiano. Ver referência no item 4.
- Q - Vetor especificando a matriz de permutação das colunas do jacobiano. Ver referência no item 4.
- F - Subrotina externa fornecida pelo usuário, utilizada para calcular  $F(x)$ .
- JACOB - Subrotina externa fornecida pelo usuário, utilizada para calcular uma linha da matriz  $F'(x)$ .
- X - Ponto inicial (na entrada) e melhor aproximação obtida pelo algoritmo (na saída).
- TRAB - Vetor de trabalho de dimensão MEMOR.
- FXMIN - Contém os valores de  $F(X)$ , onde X é a melhor aproximação obtida pelo algoritmo.
- NIT - Número de iterações realizadas.
- NITMIN - Número da iteração onde se conseguiu a melhor aproximação da solução.
- MUTILL - Número de elementos armazenados na L.
- MUTILU - Número de elementos armazenados na U.
- NFATLU - Número de fatorizações LU realizadas.
- CODERR - Código de erro :
- 0 → significa convergência
  - 1 → indica erro nos parâmetros
  - 2 → indica singularidade no jacobiano
  - 3 → faltou memória - aumentar MEMOR
  - 4 → excedeu o nº máximo de iterações NITMAX
  - 5 → significa divergência

Parâmetros rotinas - F , JACOB

Parâmetros inteiros - N, MEMOR, NITMAX, IMPR, SAIDA, QUAL, FREQ, REFAT, P, Q, NIT, NITMIN, MUTILL, MUTILU, NFATLU, CODERR



Parâmetros reais - DISTX, DISTFX, EPSLON, TOLPIV, TOLABS, EME, TRAB, FXMIN, X

Parâmetros de entrada - N, ... , JACOB

Parâmetros de saída - FXMIN, NIT, NITMIN, MUTILL, MUTILU, NFATLU, CODERR

Parâmetros de entrada e saída - X

Parâmetros de trabalho - TRAB

Dimensões de parâmetros vetores - P(N), Q(N), X(N), TRAB(MEMOR), FXMIN(N)

### 3.-Subrotinas providas pelo usuário

A primeira subrotina é aquela especificada pelo parâmetro F, e tem como objetivo calcular as componentes  $f_i(x)$ , para um ponto dado. Sua forma deve ser :

```

SUBROUTINE F(N,X,FX)
DIMENSION X(N),FX(N)
:
:
RETURN
END

```

Ela será chamada em SNLESP que, portanto, fornecerá seus parâmetros de entrada N e X. Deve ser programada de tal maneira que a componente  $f_i(x)$  seja colocada na posição FX(I).

A segunda subrotina especificada pelo parâmetro JACOB, tem como objetivo calcular uma linha da matriz jacobiana num ponto X dado. Sua forma deve ser :

```

SUBROUTINE JACOB(N,LINHA,X,VALORS,COLS,NELEM)
INTEGER COLS(N)
REAL X(N),VALORS(N)
:
:
RETURN
END

```

onde :

- N - Dimensão do sistema  
 LINHA - Nº da linha (equação) do jacobiano desejada  
 X - Ponto dado  
 VALORS - Valores do jacobiano calculados nesta equação. Devem ser colocados apenas aqueles em que  $f_i(x_j)$  dependa de  $x_j$  ( $i=LINHA$ ), ou seja, apenas as posições que possam assumir valores não nulos para algum ponto X dado. Estes valores fornecidos devem estar compactados dentro do vetor VALORS, em ordem crescente de coluna.  
 COLS - Índices das variáveis (colunas) dos valores de VALORS.  
 NELEM - Nº de elementos preenchidos em VALORS ou COLS.

Exemplo :

$$\text{Seja o sistema } (n>1) : f_i(x) \begin{cases} = (3-0.5x_1)x_1 - x_{i-1} - 2x_{i+1} + 1 \text{ para } 1 < i < n \\ = (3-0.5x_1)x_1 - x_{i-1} + 1 \text{ para } i=n \\ = (3-0.5x_1)x_1 - 2x_{i+1} + 1 \text{ para } i=1 \end{cases}$$

neste caso o jacobiano (para  $n=3$ ) fica: 
$$\begin{pmatrix} 3-x_1 & -2 & 0 \\ -1 & 3-x_2 & -2 \\ 0 & -1 & 3-x_3 \end{pmatrix}$$

as posições  $(i,j)$  para  $j>i+1$  e  $j<i-1$  serão sempre nulas.

Neste caso as subrotinas F e JACOB serão :

```

SUBROUTINE F(N,X,FX)
DIMENSION X(N),FX(N)
DO 10 I=1,N
  FX(I) = (3. - X(I)*0.5)*X(I) + 1.0
  IF(I.NE.N) FX(I) = FX(I) - 2. * X(I+1)
  IF(I.NE.1) FX(I) = FX(I) - X(I-1)
10 CONTINUE
RETURN
END
```

```

SUBROUTINE JACOB (N,LINHA,X,VALORS,COLS,NELEM)
INTEGER COLS (N) ,C (3)
REAL X (N) ,VALORS (N) ,D (3)
DATA D / -1. , 0. , -2. /
D (2) = 3. - X (LINHA)
C (1) = LINHA - 1
C (2) = LINHA
C (3) = LINHA + 1
NELEM = 0
DO 10 K=1,3
  IF (LINHA.EQ.1.AND.K.EQ.1.OR.LINHA.EQ.N.AND.K.EQ.3) GOTO 10
  NELEM = NELEM + 1
  COLS (NELEM) = C (K)
  VALORS (NELEM) = D (K)
10 CONTINUE
RETURN
END

```

#### 4.-Origem

Ver

T.L.Lopes - "Algoritmos para resolução de sistemas não lineares  
esparsos " - Tese de Mestrado - IMEXC - UNICAMP

Orientador : J. M. Martínez

#### 5.-Alternativas de métodos

São fornecidos algumas combinações dos parâmetros QUAL,  
FREQ e REFAT e o método utilizado pelo programa em cada caso:

QUAL	FREQ	REFAT	METODO
qualquer	qualquer	1	Newton
qualquer	$\infty$	p	Newton c/ refina/
1	1	$\infty$	Dennis & Marwil (modificado)
2	1	$\infty$	Lopes (muda s $\tilde{o}$ a U)
3	1	$\infty$	Lopes (muda L e U, alternada/)
1	1	p	Dennis & Marwil com refatoriza $\tilde{c}$ oes a cada p itera $\tilde{c}$ oes

→ (p-1) subitera $\tilde{c}$ oes

#### 6.-Tamanho do programa

As rotinas utilizadas somam 799 linhas FORTRAN, das quais 197 s $\tilde{a}$ o coment $\tilde{a}$ rios. Ocupam 15K de mem $\tilde{o}$ ria no PDP-10 da UNICAMP, sem contar as dimens $\tilde{o}$ es de P,Q,X,FXMIN e TRAB.

#### 7.-Precis $\tilde{a}$ o

Simples.

#### 8.-Acesso $\tilde{a}$ subrotina SNLESP

O c $\tilde{o}$ digo FORTRAN da SNLESP e das rotinas por ela utilizadas est $\tilde{a}$ o gravadas na fita MA127. Para jog $\tilde{a}$ -las na sua  $\tilde{a}$ rea, os usu $\tilde{a}$ rios do PDP-10 da UNICAMP devem dar o seguinte comando :

```
FILE R MA127 SNLESP.FOR 4
```

Este comando ordena ao operador que monte a fita MA127 e jogue o arquivo SNLESP.FOR na  $\tilde{a}$ rea do usu $\tilde{a}$ rio. Assim que este arquivo apare $\tilde{c}$ a no diret $\tilde{o}$ rio, o usu $\tilde{a}$ rio pode dar o comando abaixo, para executar seu programa principal que est $\tilde{a}$  gravado no arquivo PRINC.FOR, por exemplo :

```
EXEC/F10 PRINC.FOR,SNLESP.FOR
```

Usuários externos à UNICAMP podem se comunicar com os responsáveis no endereço :

Laboratório de Matemática Aplicada  
IMECC - UNICAMP  
CP 1170  
13100 CAMPINAS SP  
BRASIL

#### 9.-Exemplo de programa principal

Seja o sistema especificado no ítem 3, para  $n=10$ . As sub-rotinas F e JACOB serão as mesmas descritas no ítem 3. Supondo-se  $x_0=(-1, \dots, -1)$ ,  $\text{EPSLON}=10^{-6}$ ,  $\text{QUAL}=3$ ,  $\text{FREQ}=1$  e  $\text{REFAT}=1000$  (método de Lopes, mudando a L e a U, alternadamente). Então, o programa principal poderia ser :

C  
C

```

PROGRAMA PARA CALCULO DE LA INTEGRAL DE LA FUNCIÓN
INTEGRAL DE F(X) EN EL INTERVALO [A, B] CON EL MÉTODO DE SIMPSON
DEBE SER: A(UB), B(A), F(X) = (X**3)
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
N=10
DIFIX=100,
EJESFIX=100,
EJESVAR=1.0001,
IQUADIV=3.0001,
IQUADIV=1.0001,
EMUL=1.00,
MIPRAX=100,
IMPR=1,
SALIDA=20,
QUAD=1,
PRGR=1,
DEF=1=100,
DO 10 I=1, N,
P(I)=1,
Q(I)=1,
A(I)=1.0,
CONTINUE,
CALL SQUAD(I, IQUAD, DIFIX, EJESFIX, EJESVAR, IQUADIV, EJESVAR, IQUAD,
1          MIPRAX, IQUAD, IQUAD, IQUAD, IQUAD, IQUAD, IQUAD, IQUAD, IQUAD,
2          A, IQUAD, IQUAD, IQUAD, IQUAD, IQUAD, IQUAD, IQUAD,
3          IQUAD, IQUAD)
WRITE(20, 10) IQUAD, IQUAD, IQUAD, IQUAD, IQUAD, IQUAD, IQUAD, IQUAD,
2.0 FORWAT('INTEGRAL DE SIMPSON :', //, ' DEF = ', IQUAD //, ' IQUAD = ',
1      ' = ', IQUAD //, ' IQUAD = ', IQUAD //, ' IQUAD = ', IQUAD //,
2      ' IQUAD = ', IQUAD //, ' IQUAD = ', IQUAD //)
WRITE(20, 10) IQUAD, IQUAD,
3.0 FORWAT(' INTEGRAL DE SIMPSON :', //, ' DEF = ', IQUAD //,
1      ' IQUAD = ', IQUAD //)
STOP
END

```

A saída ( arquivo 20) seria :

```

/FXC( 0)/ = 1.500000E+01
ITERACAO = 1
FACTORILACAO
/FXC( 1)/ = 0.100000E+01

ITERACAO = 2
MUDANCA DA L
/FXC( 2)/ = 0.111111E+01

ITERACAO = 3
MUDANCA DA U
/FXC( 3)/ = 0.090909E+01

ITERACAO = 4
MUDANCA DA I
/FXC( 4)/ = 0.111111E+01

ITERACAO = 5
MUDANCA DA O
/FXC( 5)/ = 0.111127E+01

ITERACAO = 6
MUDANCA DA C
/FXC( 6)/ = 0.090909E+01
    
```

SALIDA DE SHELSE :

```

NIT = 6
NIT-1A = 6
NUTILL = 10
NUTYIB = 10
NSTATO = 1
CODERR = 6
    
```

MELHOR X ENCONTRADO =  
-1.030 -1.310 -1.310 -1.310 -1.310 -1.310 -1.310 -1.310 -1.310 -1.310

f(x) =  
0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000

10.-Data

13 de maio de 1982-

X.X.X.X.X.X.X.