MÉTODOS QUASE-NEWTON PARA SISTEMAS NÃO LINEARES ESPARSOS COM ESTRUTURA DINÂMICA DE DADOS

Mário César Zambaldi

Este exemplar corresponde à redação final da tese de mestrado devidamente corrigida e defendida pelo Sr. Mário César Zambaldi e aprovada pela comissão julgadora

Campinas, 24 de Outubro de 1990.

Prof. Dr. José Mário Martínez Perez
orientador

Dissertação apresentada ao Instituto de Mátemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para obtenção do Título de Mestre em Matemática Aplicada na área Otimização e Pesquisa Operacional.

AGRADECIMENTOS

Ao Prof. Dr. José Mário Martínez pela judiciosa orientação e confiança.

À Profa. Márcia A. Gomes Ruggiero pela valiosa colaboração e interesse no trabalho.

Aos Profs. Francisco de Assis M. G. Neto e Ana Friedlander pelos oportunos esclarecimentos.

Aos meus amigos e professores do Depto. de Matemática Aplicada do IMECC-UNICAMP pela agradável convivência nesse importante período.

Às instituições CNPq e FAPESP pelo suporte financeiro. Especialmente à FAPESP pelo criterioso acompanhamento do trabalho.

Não vos deixeis guiar pelas palavras dos outros, nem por tradições existentes, nem por rumores. Não vos deixeis guiar pela autoridade dos textos religiosos, nem por simples lógica ou dedução, nem por aparências, nem pelo prazer de especulação sobre opiniões, nem por verossimilhanças possíveis, nem por simples impressão ou pela idéia: Ele é nosso mestre. Desde que souberdes e sentirdes, por vos mesmos que certas coisas são desfavoráveis, falsas e ruins, então renunciai a elas. E quando souberdes e sentirdes, por vós mesmos que certas coisas são favoráveis e boas, então deveis aceitá-las e segui-las. Sede vossas próprias luzes. Sede vosso próprio apoio. Conservai-vos fiéis a verdade que há dentro de vós como sendo a única Luz.

Sidarta Gautama

À Minha Família

INTRODUÇÃO

O Grupo de Otimização do Departamento de Matemática Aplicada da UNICAMP vem trabalhando no desenvolvimento de métodos eficientes para resolução de sistemas não-lineares de grande porte e esparsos. Entre diversos produtos dessas pesquisas encontra-se o pacote computacional SNLUC que, entre outras coisas, incorpora nove métodos baseados na fatoração direta dos sistemas lineares subjacentes (subproblemas).

Neste trabalho, faz-se a implementação dos métodos que fazem parte de SNLUC, utilizando para a fatoração dos sistemas lineares, a rotina MA-28 de Harwell Subroutine Library.

A tônica fundamental é estabelecer uma comparação com SNLUC. Da mesma forma, procura-se fornecer os elementos da técnica empregada em MA-28.

O capítulo 1 tem caráter introdutório e faz uma síntese dos tópicos relacionados com o tema, incluindo a definição do problema geral e os métodos de resolução com os seus principais aspectos.

No segundo capítulo faz-se uma descrição da técnica básica para resolução direta de sistemas lineares: Eliminação Gaussiana. Da-se ênfase aos aspectos numéricos relacionados com esta técnica. A extensão para o caso esparso reside no capítulo terceiro, que procura fornecer as principais características e dificuldades que envolvem este importante

assunto: Esparsidade em métodos diretos. A idéia principal é compreender a estrutura interna de MA-28.

O quarto capítulo concerne aos métodos Quase-Newton implementados. É feita uma descrição das suas características, procedência e propriedades, considerando os principais aspectos computacionais.

No capítulo 5 são apresentados os experimentos numéricos e no capítulo 6 as conclusões finais e futuras possibilidades de pesquisa.

Finalmente, o apêndice contém alguns comentários da implementação computacional e as listagens dos programas.

ÍNDICE

1. O PROBLEMA E OS MÉTODOS DE RESOLUÇÃO

A título introdutório, este capítulo apresenta o problema típico e a formulação geral dos métodos empregados, assim como suas principais implicações.

O problema que será considerado durante todo o trabalho, é a resolução de um sistema de equações não lineares de grande porte: Encontrar $x \in \mathbb{R}^n$ de forma que,

$$F(x) = 0 \tag{1.1}$$

$$F = (f_1, \ldots, f_n)^T,$$

onde $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$, $F \in C^1$, e a matriz jacobiana $J(x)$ é esparsa (veja [12,35]).

Para a resolução de problemas desse tipo, usualmente utiliza-se o método de Newton, que caracteriza-se pelo processo iterativo:

$$x^{k+1} = x^k - J(x^k)^{-1}F(x^k). \tag{1.2}$$

Portanto, cada iteração desse método consiste no cálculo das derivadas $\partial f_i / \partial x_j$, e na resolução do sistema linear $n \times n$

$$J(x^k)s = - F(x^k), \tag{1.3}$$

onde s é o passo ou incremento, e o ponto x^{k+1} é obtido por :

$$x^{k+1} = x^k + s \tag{1.4}$$

O método de Newton possui, sob certas hipóteses adequadas sobre F (veja [35]), convergência quadrática; ou seja :

$$\frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|} \leq C \|x^k - x^*\| \tag{1.5}$$

onde C é uma constante e x^* é solução de (1.1).

Quando desenvolve-se um método procura-se obter a convergência quadrática ou algo bem próximo a ela.

Há dois problemas fundamentais na implementação do método de Newton. O primeiro é que o jacobiano $J(x^k)$ pode não estar disponível analiticamente. Isto ocorre, por exemplo, quando a função F não é fornecida analiticamente. Nestes casos, as derivadas podem ser estimadas por um esquema de diferenças finitas (veja [7]). Outro problema, é que $J(x^k)$ pode ser singular ou mal condicionado, de modo que o sistema (1.3) não pode ser empregado para obter s. Neste caso, a fatoração da matriz pode detectar a singularidade e, pode-se perturbar a matriz Jacobiana suficientemente, de modo que

fique bem condicionada e a iteração possa ser completada. Entretanto, tal modificação de $J(x^k)$ não pode ser justificada adequadamente do ponto de vista teórico em termos do problema (1.1), de modo que nem sempre é recomendável. Quando o Jacobiano é mal condicionado, recomenda-se o emprego de estratégias globais para o problema (1.1) considerando o sistema linear (1.3) (veja [12]). O contexto deste trabalho, entretanto, é o emprego de estratégias locais, de modo que faz-se a opção pela perturbação de $J(x^k)$ na fatoração (veja cap. 5).

Há casos em que o Jacobiano é difícil de calcular, ou ainda, tem um alto custo computacional. Nessas situações empregam-se métodos que utilizam uma aproximação para o Jacobiano verdadeiro. Pode ser executada de maneira simples, como manter uma matriz fixa durante todas as iterações, ou mais elaboradas, onde as aproximações são obtidas com informações geradas durante o processo iterativo. Estes são os Métodos Quase-Newton. Não deve-se esperar, portanto, convergência quadrática para os métodos Quase-Newton, uma vez que utilizam uma aproximação para $J(x^k)$. Entre os métodos que serão descritos nesse trabalho, alguns possuem propriedades de convergência satisfatórias : Superlinear,

$$\lim_{k \to \infty} \frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|} = 0 \qquad (1.6)$$

outros ainda tem convergência linear:

$$\lim_{k \to \infty} \frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|} = r < 1 \qquad (1.7)$$

De qualquer forma a iteração Quase-Newton é, geralmente, mais econômica que uma iteração Newton, de modo que a conveniência do emprego de um ou outro método depende, basicamente, do problema e do tipo da aproximação do Jacobiano empregada. Uma discussão desses métodos é apresentada no capítulo 4.

A etapa mais dispendiosa dos métodos para resolução de sistemas de equações não lineares, é a resolução do sistema linear (1.3).

A dificuldade no caso esparso, é que deve-se fazer a decomposição (Eliminação Gaussiana) de $J(x^k)$, ou de sua aproximação, para resolver (1.3). Quando transformações elementares são aplicadas à matriz, um elemento que originariamente era nulo pode tornar-se não nulo nos fatores; este fenômeno é conhecido como *Fill-in* ou preenchimento. O preenchimento é um problema significativo no caso esparso, pois somente os elementos não nulos são armazenados e, portanto, é necessário memória adicional para cada novo elemento não nulo. Outro problema crítico, é o esforço computacional que aumenta, na mesma proporção do preenchimento, pois cada novo fator deve ser operado, não raro, muitas vêzes.

Como será visto no capítulos 4 e 5, a fatoração da matriz do sistema linear, tem grande influência no desempenho no métodos Quase-Newton.

Em virtude da relevância deste tema, o capítulo que se segue dedica-se exclusivamente à resolução do sistema linear e o seguinte uma extensão para o caso esparso.

2. ELIMINAÇÃO GAUSSIANA: ASPECTOS NUMÉRICOS E ALGÉBRICOS

Este capítulo faz uma descrição do método mais popular para a resolução de sistemas: A Eliminação Gaussiana ou Fatoração **LU**. São discutidas as principais características desta técnica, a necessidade de permutação de linhas e/ou colunas no processo de pivotamento, considerações numéricas sobre estabilidade do algoritmo, condicionamento de sistemas lineares e, finalmente, alguns aspectos numéricos adicionais.

2.1 O MÉTODO

Considere o sistema triangular de equações

$$
\begin{aligned}
a_{11}x_1 &= b_1 \\
a_{21}x_1 + a_{22}x_2 &= b_2 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n
\end{aligned}
$$

que pode ser facilmente resolvido recursivamente por

$$x_1 = b_1/a_{11}$$
$$x_2 = (b_2 - a_{21}x_1)/a_{22}$$
.
.
.
$$x_n = (b_n - a_{n1}x_1 - a_{n2}x_2 \ldots - a_{nn-1}x_{n-1})/a_{nn},$$

onde os elementos da diagonal a_{ii}, $i = 1,2,\ldots n$ são não nulos, considerando o sistema não singular. Este fato motiva a idéia de procurar conduzir um sistema qualquer à um outro sistema triangular equivalente.

Utilizando a notação matricial, a idéia da Eliminação de Gauss é obter a decomposição de uma matriz **A** quadrada n x n da forma **A = LU**, onde **L** é uma matriz triangular inferior e **U** triangular superior. O sistema

$$\mathbf{Ax = b} \qquad (2.1)$$

pode, então, ser resolvido através da resolução dos sistemas triangulares

$$\mathbf{Ly = b} \qquad (2.2a)$$

$$\mathbf{Ux = y} \qquad (2.2b)$$

A resolução de (2.2a) é denominada substituição progressiva (*forward substitution*) e de (2.2b) retro-substituição (*back substitution*).

Toda matriz quadrada não singular tem uma decomposição **LU**, podendo para obtê-la, ser necessário efetuar permutações nas linhas da matriz. Isto corresponde,

simplesmente, a fazer uma ordenação das linhas do sistema linear correspondente, não perdendo, portanto, a generalidade se não forem consideradas tais permutações.

Para determinar **L** e **U** de uma matriz não singular **A**, a idéia é anular todos os elementos abaixo da diagonal principal. Considerando $a_{11} \neq 0$, subtrai-se um múltiplo da primeira equação de todas as outras para obter elementos nulos abaixo de a_{11} na primeira coluna. Considerando $m_{k1} = a_{k1}/a_{11}$ e

$$\mathbf{M}_1 = \begin{bmatrix} 1 & & & & & \\ -m_{21} & 1 & & & & \\ -m_{31} & & 1 & & & \\ \vdots & & & \ddots & & \\ \vdots & & & & \ddots & \\ -m_{n1} & \cdots & \cdots & \cdots & \cdots & 1 \end{bmatrix}$$

o novo sistema resultante será

$$\mathbf{A}^{(2)}\mathbf{x} = \mathbf{b}^{(2)}$$

com $$\mathbf{A}^{(2)} = \mathbf{M}_1\mathbf{A}, \qquad \mathbf{b}^{(2)} = \mathbf{M}_1\mathbf{b},$$

a matriz $\mathbf{A}^{(2)} = [a_{ij}^{(2)}]$ tem $a_{k1}^{(2)} = 0$, $k > 1$.

Agora, considerando $a_{22}^{(2)} \neq 0$, múltiplos da segunda equação do novo sistema são subtraidos das equações 3 a n, para obter zeros abaixo de $a_2^{(2)}$ na segunda coluna. Isto é equivalente a premultiplicar $\mathbf{A}^{(2)}$ e $\mathbf{b}^{(2)}$ por

$$M_2 = \begin{bmatrix} 1 & 0 & & & & \\ 0 & 1 & & & & \\ . & -m_{32} & 1 & & & \\ . & -m_{42} & & . & & \\ & \vdots & & & . & \\ & -m_{n2} & . & . & . & 1 \end{bmatrix}$$

onde $m_{k2} = a_{k2}^{(2)}/a_{22}^{(2)}$. Isto resulta $A^{(3)} = M_2 A_2^{(2)}$ e $b^{(3)} = M_2 b^{(2)}$.

Procedendo da mesma forma obtem-se $A^{(n)} = M_{n-1} M_{n-2} \ldots M_1 A$, uma matriz triangular superior U, e como MA = U, tem-se $A = M^{-1}U$. A matriz $L = M^{-1}$ é triangular inferior e portanto foi obtida a decomposição desejada LU da matriz A. As submatrizes quadradas, obtidas a cada passo da decomposição, cujas diagonais são $a_{kk}^{(k)}, a_{k+1,k+1}^{(k)} \ldots a_{nn}^{(k)}$, k=1,2...n, são denominadas submatrizes ativas ou reduzidas.

A representação de L pode ser mais explícita, notando que M_k^{-1} é a mesma M_k, exceto que os termos fora da diagonal tem sinais opostos. Ainda, tem-se $L = M^{-1} = M_1^{-1} M_2^{-1} \ldots M_{n-1}^{-1}$, que facilmente se verifica ser,

$$L = \begin{bmatrix} 1 & 0 & & & & \\ m_{21} & 1 & & & & \\ m_{31} & m_{32} & 1 & & & \\ . & & & . & & \\ & & & & . & \\ & \vdots & & & & \\ & & & & & . \\ m_{n1} & m_{n2} & . & . & . & 1 \end{bmatrix}$$

portanto, L pode ser obtida diretamente em termos dos cálculos

necessários na eliminação.

Se o sistema original (2.1) é utilizado para apenas um vetor **b**, o vetor **y** satisfazendo **Ly = b** é frequentemente calculado simultaneamente com **L** na forma $\mathbf{y} = \mathbf{b}^{(n)} = \mathbf{Mb}$. Uma vez que a decomposição **LU** de **A** foi obtida, a solução para qualquer vetor **b** pode ser obtida somente pela resolução dos dois sistemas triangulares (2.2).

Variantes da Eliminação de Gauss como os algorítmos Doolittle e Crout, diferenciam-se daquela apenas pela sequência dos cálculos (veja [15]). São convenientes em aplicações que exigem que a eliminação seja feita por linhas, por colunas ou alternadamente.

O custo computacional associado à eliminação, mostra que o número operações é dado por:

$$\frac{2}{3}.n^3 - \frac{1}{2}.n^2 - \frac{1}{6}.n,$$

para fatoração **LU**. O termo que domina o custo para n grande é

$$\frac{2}{3}.n^3 + O(n^2);$$

quando **A** for simétrica, será $\frac{1}{3}.n^3 + O(n^2)$, e para os sistemas triangulares, simplesmente $n^2 + O(n^2)$.

Na prática, os elementos da diagonal $a_{kk}^{(k)}$ de $\mathbf{A}^{(k)}$ podem ser nulos ou muito próximos de zero. Neste caso, é importante que a k-ésima linha seja permutada com uma outra linha abaixo dela. Quaisquer que sejam as permutações de linhas necessárias durante a eliminação, a fatoração obtida será a mesma que seria obtida, caso as linhas de **A** fossem permutadas da mesma forma antes da eliminação, e esta aplicada sem trocas de linhas à matriz **A** permutada.

As permutações podem ser representadas, em notação matricial, por uma matriz **P**, denominada matriz de permutação,

obtida da matriz identidade aplicando a mesma seqüência de trocas de linhas. Assim, será obtida a decomposição de PA (i. é. PA = LU). Naturalmente estas mesmas observações são válidas quando for introduzida reordenação de colunas.

O efeito das permutações de linhas e colunas pode ser incorporada facilmente. Se P e Q são matrizes de permutação, a fatoração fica,

$$PAQ = LU \tag{2.3}$$

$$PAQQ^{T}x = Pb \tag{2.4}$$

de modo que (2.2) fica

$$\begin{aligned} y &= Pb \\ Lz &= y \\ Uw &= z \\ x &= Qw. \end{aligned} \tag{2.5}$$

O emprego das permutações está associado ao controle de estabilidade numérica do algorítmo. No processo da eliminação, a solução computacional obtida, evidentemente, não é a solução exata. Isto deve-se aos erros de aritmética de ponto flutuante que se propagam durante o processo. Existem outros tipos de erros que tem origem no próprio problema, provenientes, na maioria das vezes, do mal escalamento dos dados. Esses são os próximos assuntos a serem discutidos.

2.2 CONSIDERAÇÕES SOBRE ESTABILIDADE NUMÉRICA

Considere x a solução exata de (2.1) e $\tilde{x}$ a solução aproximada obtida pela utilização de algum algoritmo, sujeita a erros de aritmética computacional. Se a solução computada $\tilde{x}$ é a solução exata de um problema aproximado do problema original, diz-se que o algoritmo utilizado é estável.

Uma maneira de verificar a estabilidade, que geralmente é empregada, é examinar o resíduo $r = b - Ax$, avaliando se $\|r\|$ é pequena comparada com $\|b\|$ ou com $\|A\|\|x\|$, onde $\|.\|$ é qualquer uma das normas de operadores $\|.\|_\infty$, $\|.\|_1$ ou $\|.\|_2$. Embora este seja um modo simples de detectar a estabilidade, por outro lado não garante um resultado definitivamente seguro (veja [4]), sendo apenas um *bom indício* daquela.

Uma outra forma é verificar se os elementos da matriz H são " pequenos ". Obtem-se $\tilde{x}$ e não x pois, em vez da fatoração LU exata, obtem-se a fatoração $A \cong \tilde{L}\tilde{U}$ aproximada e, portanto, A deve ser substituida por $A + H$, isto é

$$\tilde{L}\tilde{U} = A + H \qquad (2.6)$$

onde $H = \tilde{L}\tilde{U} - A$, é a matriz de erro da fatoração e deve ser pequena para algoritmos estáveis.

Um resultado prático (veja [15]) para estimar o crescimento dos elementos da matriz H é,

$$|h_{ij}| \leq 5.01\ \varepsilon\ n\ \rho, \qquad (2.7)$$

onde $\rho = \max_{i\,j\,k} |a_{ij}^{(k)}|$ e ε é a precisão do computador.

É importante considerar o problema bem escalado. O

mal escalamento geralmente está associado com a utilização inconveniente das unidades, estabelecendo desproporções consideráveis entre as grandezas envolvidas. Normas e condicionamento (veja seção 2.3) são afetados pelo escalamento do problema.

Uma maneira usual de controlar o crescimento dos fatores, é exigir $|l_{ij}| \leq 1$ para todos os elementos da matriz L. Isto é obtido reordenando as linhas da matriz durante a eliminação, de modo que o pivô $a_{kk}^{(k)}$, verifique a desigualdade,

$$|a_{kk}^{(k)}| \geq |a_{ik}^{(k)}|, \quad i > k. \tag{2.8}$$

Esta é a estratégia do pivotamento parcial e consiste, basicamente, em evitar pequenos pivôs relativos, da mesma forma que se evita o pivô nulo em aritmética exata. Na prática, a Eliminação Gaussiana com pivotamento parcial é considerado um algorítmo estável, e nesse caso obtem-se,

$$\rho \leq 2^{n-1} \max_{i,j} |a_{ij}| \tag{2.9}$$

que é um limitante a priori para o crescimento de h_{ij}.

Se linhas e colunas são ordenadas em cada estágio da eliminação para assegurar a desigualdade,

$$|a_{kk}^{(k)}| \geq |a_{ij}^{(k)}|, \quad i \geq k, \; j \geq k \tag{2.10}$$

tem-se o pivotamento completo ou total. e ρ passa a ser (veja [41])

$$\rho \leq f(n) \max|a_{ij}|, \qquad (2.11)$$

com

$$f(n) = [n(2^1 3^{1/2} 4^{2/3} \ldots n^{1/n-1})]^{1/2}.$$

Observe que $f(n) \ll 2^{n-1}$ para n grande. O pivotamento total não é na prática muito empregado pois envolve $\frac{1}{3}n^3 + O(n)$ comparações para uma matriz densa.

A dificuldade com o pivotamento parcial (ou completo), é que a desigualdade (2.8) (ou (2.10)) é muito restritiva para sistemas esparsos. Uma estratégia alternativa para este caso, é o pivotamento com um parâmetro de tolerância, que permite maior liberdade na escolha dos pivôs. Consiste em escolher o pivô que satisfaz a desigualdade

$$|a_{kk}^{(k)}| \geq u \ |a_{ik}^{(k)}|, \quad i > k. \qquad (2.12)$$

onde $0 < u \leq 1$, é o parâmetro de tolerância, e deve ser escolhido convenientemente. Observe que isto não passa de uma generalização do pivotamento parcial (u=1) e que, nesse caso $|l_{ij}| \leq u^{-1}$. O limitante ρ para este caso fica

$$\rho \leq (1+u^{-1})^{n-1} \max_{i,j} |a_{ij}|. \qquad (2.13)$$

Este procedimento oferece pouco benefício quando A é densa, mas é muito útil no caso de matrizes esparsas.

Devido ao uso restrito dos limitantes para ρ, que foram relacionados até agora, e porque algumas vezes a escolha dos pivôs é predeterminada, geralmente é conveniente monitorar a estabilidade durante a fatoração ou fazê-la a posteriori. Isto envolve o refinamento iterativo, a escolha de uma nova seqüência pivotal ou mesmo um trabalho com maior precisão.

Calcular H, $\|\tilde{L}\|$, $\|\tilde{U}\|$ ou mesmo $\rho = \max_{i,j,k} |a_{ij}^{(k)}|$ envolve um alto custo. Utiliza-se, então, um limitante para ρ em [15] :

$$\rho \leq \max_{i} \|(l_{i1} \ldots l_{ii})\|_p \ \max_{j} \|(u_{1j} \ldots u_{jj})\|_q, \tag{2.14}$$

com $1/p + 1/q = 1$, e

$$\rho \leq n \max_{i,j} |u_{ij}|, \tag{2.15}$$

para o pivotamento parcial com p = 1 e q = ∞.

Uma técnica alternativa para evitar problemas de estabilidade numérica é a fatoração ortogonal. A matriz **A** é fatorada da seguinte forma,

$$\mathbf{A} = \mathbf{QU} \tag{2.16}$$

onde **Q** é uma matriz ortogonal e **U** triangular superior, de modo que **Ax = b** é facilmente obtido premultiplicando **b** por $\mathbf{Q}^T$ e da retro-substituição em **U**. Métodos que utilizam esta técnica são algumas vezes empregados por suas boas propriedades numéricas (veja [41]), mas são pelo menos duas vezes mais caros que a Eliminação Gaussiana.

2.3 CONSIDERAÇÕES SOBRE CONDICIONAMENTO DE SISTEMAS LINEARES

Na prática, muitas vezes os dados dos problemas (como 2.1) são obtidos por observações experimentais e portanto estão sujeitos a pequenas perturbações, originadas por exemplo, dos instrumentos de medidas com precisão

específica.

Assim, é conveniente cogitar se algumas perturbações nos dados do problema, causam também pequenas alterações na solução. Se isso ocorre diz-se que o problema é bem condicionado e, naturalmente, mal condicionado caso contrário. Esta propriedade está intrinsecamente relacionada ao problema e não tem relação alguma com o algoritmo utilizado para obter a solução.

Considere a solução $\tilde{x}$ do sistema

$$(A + H)\tilde{x} = b \qquad (2.17)$$

com as quantidades $\|H\|$, $\|r\|=\|b-A\tilde{x}\|$ e $\|\tilde{r}\|=\|\tilde{b}-A\tilde{x}\|$ relativamente pequenas. Embora, como foi visto na seção anterior, isto garanta a estabilidade da solução, não está assegurada ainda a viabilidade da solução, de modo que $\|x-\tilde{x}\|$ pode ser relativamente grande. É o caso em que se dá o mal condicionamento .

Para o problema $Ax = b$, se b é perturbado de δb, causando uma perturbação δx em x verifica-se o sistema

$$A(x+\delta x)=b+\delta b,$$

tomando normas em $Ax = b$ e $\delta x = A^{-1}\delta b$, obtem-se

$$\frac{\|\delta x\|}{\|x\|} \leq \|A\| \, \|A^{-1}\| \, \frac{\|\delta b\|}{\|b\|}. \qquad (2.18)$$

Analogamente, considerando o sistema perturbado

$$(A+\delta A)(x+\delta x) = b,$$

subtraindo $Ax = b$, premultiplicando por A^{-1}e tomando as normas obtem-se

$$\frac{\|\delta x\|}{\|x+\delta x\|} \leq \|A\| \, \|A^{-1}\| \, \frac{\|\delta A\|}{\|A\|}. \qquad (2.19)$$

A quantidade $\|A\|\|A^{-1}\|$ é denominada número de condição da matriz A, denotado por $k(A)$.

Observando (2.18) e (2.19), o número de condição pode ser visto sob dois aspectos significativos. Em primeiro lugar pode ser considerado como um limitante da razão entre o erro relativo da solução e o erro relativo dos dados. Representa também, juntamente com a precisão do computador, um *intervalo* de soluções admissíveis para uma família de problemas $(A+\Delta A, b+\Delta b)$.

2.4 ASPECTOS NUMÉRICOS ADICIONAIS

Embora (2.18) e (2.19) considerem o efeito das perturbações em A e b separadamente, na prática resolve-se o problema

$$(A+\delta A)(x+\delta x)=b+\delta b,$$

com o efeito das perturbações nos dados em A e b simultaneamente, que por sua vez fornece (veja [15]),

$$\frac{\|\delta x\|}{\|x\|} \leq \frac{k(A)}{1 - k(A)\frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|} \right). \qquad (2.20)$$

Observe que (2.20) consubstancia as observações

introdutórias das seções 2.2 e 2.3. Se o algoritmo é estável, a definição assegura que um problema *próximo* do original foi resolvido, isto é, $(\|\delta b\|/\|b\| + \|\delta A\|/\|A\|)$ é *pequeno*. A viabilidade da solução estará assegurada se problemas na vizinhança possuirem soluções na vizinhança da solução original, isto é, se $k(A)$ for *pequeno*. Concluindo, o efeito da instabilidade e do mal condicionamento são cumulativos.

Evidentemente, o número de condição é um valor crítico para acessar a viabilidade da solução, portanto é importante ser capaz de calculá-lo, pelo menos estimá-lo razoavelmente, ou ainda, simplesmente, detectar o mal condicionamento. A priori, poderia se pensar em calcular o maior e menor valores singulares de A, calcular A^{-1} e avaliar $\|A\|\|A^{-1}\|$ ou mesmo o determinante de A (pois singularidade é mal condicionamento). O modo mais prático, o cálculo do determinante, não é seguro pois pode-se obtê-lo pequeno com uma boa solução ou obtê-lo grande com uma solução ruim. As outras alternativas são impraticáveis no sentido que tem um alto custo computacional. Alternativas práticas são:

i) Observar pequenos pivôs.

ii) Resolver um problema com solução conhecida (construir o termo independente como a soma dos elementos da linha gerando solução $(1,\ldots,1)^T$, por exemplo) e checar o resultado.

iii) A estimativa LINPACK em [6]. Consiste, basicamente, em calcular x em $A^Tx = b$, para certo vetor b específico, e obter y por $Ay = x$. Utiliza-se então, $\|y\|_1/\|x\|_1$ como uma estimativa de $\|A^{-1}\|_1$.

iv) O emprego do refinamento iterativo.

A alternativa mais efetiva é *iii)* seguida de *iv)*, *ii)* e *i)* (veja [15]). O refinamento iterativo, entretanto, é uma técnica eficiente para melhorar a solução e também para avaliar a sensibilidade da perturbação dos dados.

Considere $\tilde{x}$ a solução obtida pela Eliminação Gaussiana, $x^{(1)} = \tilde{x}$ e o resíduo $r^{(1)} = b - Ax^{(1)}$, então

$$A^{-1}r^{(1)} = A^{-1}b - x^{(1)} = x - x^{(1)},$$

portanto, pode-se obter a correção $A\Delta x^{(1)} = r^{(1)}$, e construir uma nova solução aproximada

$$x^{(2)} = x^{(1)} + \Delta x^{(1)},$$

generalizando, tem-se

$$r^{(k)} = b - Ax^{(k)}$$

$$A\Delta x^{(k)} = r^{(k)}$$

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)}, \quad k = 1,2\ldots$$

se os resíduos $r^{(k)}$ são computados usando uma melhor precisão, o processo usualmente converge. Além de melhorar a solução, o refinamento iterativo pode ser utilizado para indicar o provável erro em $\tilde{x}$ e, realmente, este talvez seja o seu papel mais importante. Finalmente, observe que **A** foi substituída por $\tilde{L}\tilde{U}$ e que, no preenchimento, estarão números da magnitude de $\varepsilon.\max|a_{ij}^{(k)}|$, que é uma mudança significativa, mas que pode ser corrigida pelo refinamento iterativo.

3. ESPARSIDADE EM MÉTODOS DIRETOS

Neste capítulo faz-se um estudo dos principais aspectos relacionados com a fatoração esparsa, baseado no esquema utilizado em MA-28. São caracterizadas as fases de resolução, dando ênfase às estratégias de pivotamento e o trabalho computacional envolvido.

3.1 INTRODUÇÃO

Sempre quando se procura explorar a esparsidade na resolução de sistemas lineares, o objetivo principal é a manutenção da estabilidade numérica sem introduzir demasiado preenchimento, que pode comprometer a resolução do problema, tanto pela disponibilidade de memória como pelo custo adicional decorrente do aumento do número de operações.

A implementação prática de métodos diretos basicamente envolve três fases distintas: ANALISE, FATORIZE E SOLVE, como estabelecida em [15]. Embora esse não seja um padrão absoluto, a maioria das rotinas computacionais tem essas características que, resumidamente, tem a seguinte forma:

ANALISE

1. Adequar estrutura de dados.
2. Determinar a seqüência pivotal.
3. Preparar estrutura para as fases posteriores.
4. Fornecer estatísticas de execução.

FATORIZE

1. Transferir novos valores numéricos para estrutura interna padrão.
2. Fatorar a matriz com a seqüência pivotal escolhida.
3. Estimar o número de condição da matriz.
4. Monitorar a estabilidade para assegurar a viabilidade da fatoração.

SOLVE

1. Executar as permutações definidas pela seqüência pivotal.
2. Executar a substituição progressiva e a retro-substituição.
3. Executar as permutações na solução.

Quando a matriz for simétrica, ou quase simétrica, e os pivôs da diagonal fornecerem uma fatoração estável, estas três fases são bem distinguidas. É o caso em que a fase ANALISE fornece a seqüência pivotal analisando somente o modelo de esparsidade, não considerando os valores numéricos.

No caso não simétrico, o padrão de esparsidade e os valores numéricos devem ser considerados[1] e, na verdade, a fatoração é praticamente completada nesta fase. Evidentemente, o custo deve ser consideravelmente mais elevado que no caso simétrico, pois os fatores numéricos tem influência decisiva na escolha dos pivôs, realizada pela ordenação das linhas e colunas das submatrizes ativas.

Uma ilustração da necessidade de ordenação para manutenção da esparsidade na eliminação é dada a seguir, onde permutações prévias permitem não introduzir preenchimento, ao contrário da matriz original, que produz o máximo preenchimento.

FIG. 3.1 Necessidade de permutações para preservar a esparsidade

1

Embora SNLUC seja destinado para o caso não simétrico, as três etapas são distintas com a fase simbólica fixando a estrutura para a fatoração numérica (veja cap. 5).

O emprego da estratégia de Markowitz [15] para este exemplo fornece tal ordenação. Consiste em selecionar como pivô, a cada passo da Eliminação Gaussiana, o elemento da submatriz reduzida que tem o menor produto entre o número de elementos não nulos da sua linha e da sua coluna.

Existem outras técnicas alternativas como permutar a matriz de forma que os elementos nulos fiquem isolados. É o caso de estrutura banda, onde não haverá preenchimento se não houver permutação posterior, e caso triangular por blocos onde, além de não provocar preenchimento fora da estrutura, a decomposição será feita somente nos blocos diagonais.

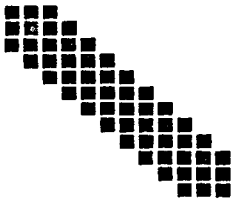


Fig. 3.2a Estrutura banda

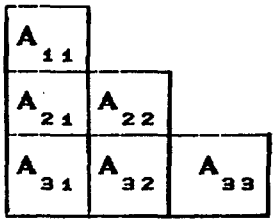


Fig. 3.2b estrutura bloco triangular (inferior)

Na fase ANALISE para o caso não simétrico, também deve ser feita a checagem da estabilidade numérica. Uma técnica usual é controlar o crescimento dos fatores pela estratégia do parâmetro de tolerância descrita na seção 2.2. Um limitante a priori é dado por Gear [18],

$$\max_i |a_{ij}^{(k)}| \leq (1 + u^{-1})^{p_j} \max_i |a_{ij}| \qquad (3.1)$$

onde p_j é o número de elementos fora da diagonal na j-ésima coluna de U. Mas como acontece com os limitantes a priori da

seção 2.2, este também é demasiado pessimista, de modo que é melhor checar a estabilidade numérica durante o processo. Se ocorrer instabilidade, uma nova ANALISE pode ser feita com maior valor para u. Um modo de acessar a estabilidade, além daquele descrito na seção 2.2, é substituir n de alguma forma na expressão

$$|h_{ij}| \leq 5.01\ \varepsilon\ \rho\ n, \tag{3.2}$$

que, para problemas grandes (n = 100000) pode influenciar negativamente $|h_{ij}|$. Gear [18] generalizou esse limite para o caso esparso, obtendo

$$|h_{ij}| \leq 1.01\ \varepsilon\ \rho(up^3 + (1+u)p^2) + O(\varepsilon^2) \tag{3.3}$$

onde

$$p = \max_j p_j$$

Apesar de n não aparecer em (3.3), aparece p^3 que pode significar, em alguns casos, uma deficiência deste resultado. Outra possibilidade, que geralmente é feita, é o cálculo do resíduo.

Na fase FATORIZE, considere que uma estrutura estática foi fornecida, contendo a seqüência de pivôs e o modelo de preenchimento. Nesse estágio, a questão está em estabelecer se a fatoração é estável para um novo conjunto de valores numéricos. No caso em que se der a instabilidade, a fase ANALISE pode ser feita novamente. Na prática, entretanto, problemas com o mesmo padrão de esparsidade tendem a ser estáveis e realmente este fato foi confirmado nos experimentos (veja cap. 5).

Evidentemente, a fase ANALISE é muito mais dispendiosa que a fase FATORIZE (veja cap. 5), o que não ocorre no caso da implementação do caso simétrico, onde o

custo é da mesma ordem. Ao contrário do que ocorre no caso denso, a fase SOLVE é relevante no caso esparso, já que depende do preenchimento gerado na fatoração.

Finalmente, no caso denso o custo computacional é considerado de $O(n^3)$ operações. No caso esparso, entretanto, tende a ser bem menor, principalmente quando a densidade da matriz for relativamente baixa e, naturalmente, dependendo da quantidade de preenchimento ocorrido na fatoração. Considere que a quantidade de elementos introduzidos não excede o número de elementos eliminados, e que o número médio de elementos nas linhas de **U** seja c. O número de operações num passo da eliminação é, então em média, menor que $2c^2$; portanto o número total de operações é menor que $2c^2n$. Embora não seja um resultado teórico, é um argumento plausível limitar o preenchimento e, na prática, isto realmente ocorre, de forma que para n grande $O(c^2n) \ll O(n^3)$.

3.2 ESTRATÉGIAS DE PIVOTAMENTO

Não há um critério único, bem determinado ou absoluto para a escolha do pivô na Eliminação Gaussiana para matrizes esparsas. A príncipio, o objetivo poderia estar em obter o mínimo de preenchimento. Entretanto, isso pode ter um elevado custo computacional no sentido de que, às vezes, seja mais conveniente provocar uma quantidade limitada de preenchimento, além de ter que considerar o problema da estabilidade. Pode-se estar interessado em resolver vários problemas com a mesma estrutura ou ainda explorar arquiteturas paralelas.

Portanto, não deve-se padronizar em estabelecer o melhor critério de ordenação para a escolha do pivô, uma vez que a escolha está subordinada ao problema de interesse. Assim, há que se analisar as disponibilidades e, a partir daí, procurar objetivos sempre relativos.

De qualquer forma, todas as estratégias devem ter algum controle de preenchimento, que pode ser, a cada passo, obter um objetivo sem considerar os passos seguintes, denominadas estratégias locais; e outras que procuram preservar ou conduzir a matriz à alguma estrutura especial (banda ou triangular por blocos por exemplo), denominadas estratégias globais. Em seguida são discutidas algumas dessas estratégias.

3.2.1 O CRITÉRIO DE MARKOWITZ

Suponha que se esteja no estagio k da Eliminação Gaussiana. Para cada linha i da submatriz ativa $(n-k+1)x(n-k+1)$ considere r_i o número de elementos da linha i e c_j na coluna j. O critério de Markowitz consiste em escolher o elemento $a_{ij}^{(k)}$ da submatriz ativa, que não seja *muito pequeno* e que minimize a expressão de Markowitz,

$$M_{ijk} = (r_i^{(k)} - 1)(c_j^{(k)} - 1). \qquad (3.4)$$

Observe que o uso de (3.4) e não $r_i^{(k)} * c_j^{(k)}$, é para escolher uma linha ou coluna que possuam somente um elemento (linha ou

preenchimento ocorrerá com esta escolha.

Markowitz interpretou esta estratégia como, uma vez escolhidos os k primeiros pivôs, modificar o mínimo de coeficientes possíveis na submatriz ativa neste estágio, ou ainda, uma aproximação que minimiza o preenchimento, pois uma vez escolhido $a_{ij}^{(k)}$ como pivô, ocorrerão no máximo $(r_i^{(k)}-1)(c_j^{(k)}-1)$ preenchimentos. Também pode ser considerada como minimizar o número de operações, pois procedendo assim, faz-se $r_i(c_j-1)$ multiplicações e $(r_i-1)(c_j-1)$ adições.

Implementar a estratégia de Markowitz exige o conhecimento do modelo de esparsidade da submatriz reduzida, que é atualizada em cada estágio da eliminação. Portanto é necessário o acesso por linhas e colunas. Também não pode ser feita simbólicamente, já que deve ser capaz de evitar pequenos pivôs. Ainda, requer que o procedimento de busca seja habilmente limitado, pois num procedimento normal estaríamos no caso do pivotamento completo.

A precaução com elementos *pequenos* (estabilidade), pode ser feita introduzindo o pivotamento com parâmetro de tolerância, seção (2.2). Evidentemente, deve-se considerar a preocupação em não ser muito rígido na escolha do pivô, de modo a minimizar a busca.

Finalmente, a estratégia de Markowitz não é aquela que introduz o mínimo de preenchimento. Este fato é ilustrado no exemplo abaixo,

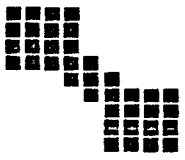


FIG. 3.9 Caso onde Markowitz não produz o mínimo preenchimento

onde a escolha de pivôs nas diagonais na ordem original não produz preenchimento, enquanto que a primeira escolha de Markowitz seria em a_{55}, que produz 2 preenchimentos.

3.2.2 OUTRAS ESTRATÉGIAS DE PIVOTAMENTO

Como foi obvservado, a estratégia de Markowitz não é aquela que introduz o mínimo de preenchimento. A estratégia local de mínimo preenchimento, que é assim denominada, consiste em escolher o pivô, a cada estágio da Eliminação Gaussiana, não *muito pequeno* que introduz o mínimo de preenchimento. Obviamente, o custo computacional desse procedimento é elevado pelos mesmos motivos ressaltados da alternativa de Markowitz, ainda acrescido de uma busca mais criteriosa.

Um caso especial de Markowitz é quando A possui uma estrutura simétrica, juntamente com a hipótese de que os elementos da diagonal forncem pivôs numericamente aceitáveis. Nesse caso o pivô, a cada estágio, será $a_{ii}^{(k)}$ selecionado com i satisfazendo:

$$r_i^{(k)} = \min_t r_t^{(k)}. \qquad (3.5)$$

Esta é a estratégia de Mínimo Grau, devido a relação com o grafo associado (veja [15]).

Uma alternativa para minimizar o custo da seleção dos pivôs é o emprego de estratégia de ordenação a priori. Consiste basicamente em estabelecer algum critério antes da eliminação, seguido de algum controle de preenchimento durante

a mesma. Exemplos desta alternativa são:

i) ORDENAÇÃO DE COLUNAS(LINHAS) A PRIORI.

ii) ORDENAÇÃO DE COLUNAS(LINHAS) A PRIORI COM MÍNIMO CUSTO DE LINHA(COLUNA).

Consistem em ordenar as colunas(linhas) da matriz original em ordem crescente de elementos não-nulos. Na primeira, em cada estágio da eliminação, é selecionado o pivô que minimiza a parte reduzida segundo o custo original das linhas(colunas). Na segunda o pivô é escolhido segundo o menor custo de linhas(colunas) da submatriz ativa neste estágio. Existem versões simétricas para *i)* e *ii)*. Evidentemente, técnicas desse tipo não se caracterizam como estratégias locais, uma vez que as permutações das colunas são escolhidas a priori.

Existem outras estratégias simplificadoras que procuram reduzir os custos na seleção de pivôs sem produzir demasiado preenchimento, até mesmo fundamentadas em predições probabilísticas. Por outro lado, há aquelas que procuram estruturas especiais, como por exemplo, reagrupar os fatores numa faixa próxima às diagonais principal ou secundária, ou ainda à estrutura bloco-triangulares (veja [42]).

Há também os esquemas ditos combinados, que nada mais são do que uma combinação dos esquemas anteriores, onde a ordenação dinâmica é precedida de uma ordenação estática.

Finalmente, a menos que se tenha um bom indício da conveniência do emprego de uma ou outra técnica, as estratégias do tipo Markowitz são mais eficientes de modo geral. Estas são as conclusões em [15], onde são apresentados experimentos com os métodos aqui discutidos.

3.3 CONSIDERAÇÕES COMPUTACIONAIS

É oportuno discutir as técnicas que envolvem a implementação computacional para se ter uma idéia da estrutura interna de MA-28, ou ainda aproveitá-la em aplicações específicas como no caso de sistemas não lineares, onde uma seqüência de sistemas lineares, geralmente com a mesma estrutura são resolvidos.

Na fase ANALISE, a estratégia de Markowitz com o parâmetro de tolerância é empregada. Consiste, portanto, em escolher o pivô que minimize a expressão de Markowitz (3.4) entre todos os elementos da matriz reduzida, que satisfazem a desigualdade

$$|a_{ij}^{(k)}| \geq u \max_{t \geq k} |a_{it}^{(k)}|, \qquad (3.6a)$$

ou a desigualdade,

$$|a_{ij}^{(k)}| \geq u \max_{t \geq k} |a_{tj}^{(k)}|. \qquad (3.6b)$$

Este procedimento prevê, a princípio, que todos os elementos devem ser acessados e isso é, evidentemente, impraticável. Exemplificando, considere uma matriz com n = 10000, com 40000 elementos não nulos e um pivô escolhido com $r_i = c_j = 3$. Neste caso, o primeiro passo da eliminação exigirá 40.000 testes para somente 10 operações. Na prática, entretanto, emprega-se esquemas de armazenamento e acesso ordenados para r_i e c_j. Curtis e Reid [8] propuseram a busca em ordem crescente de número de elementos na linha(coluna).

Quando um elemento satisfazendo (3.6a) tiver M_{ij} tão baixo quanto os elementos não acessados, a busca termina. Na verdade, pode ocorrer que o pivô não seja escolhido na linha com mínimos r_i e c_j, ou seja

$$M_{ij} = (r_i - 1)(c_j - 1) > \min_t (r_t - 1) \min_t (c_t - 1) \qquad (3.7)$$

isto ocorrerá quando não houver elemento na posição (i,j) ou o elemento não for aceitável numericamente. De qualquer forma, a expectativa é de que M_{ij} seja próximo do mínimo e, portanto, esse procedimento é empregado.

É fundamental, para este esquema, ter um modo de acesso por linhas e colunas na ordem crescente de r_i e c_j. Uma rotina sort em Knuth [26] permite ordenar n números entre 1 e n em $O(n)$ operações. Mesmo assim, o trabalho seria intolerável pois, voltando ao exemplo anterior, necessitar-se-ia cerca de 20000 operações num passo da eliminação. Uma alternativa é obter um esquema de armazenamento que permita a ordenação rápida de r_i e c_j em função da ordenação do passo anterior e, desse modo, o trabalho de ordenação seria feito somente no primeiro passo da eliminação. Um esquema deste tipo é apresentado, sistematicamente, em [13]. Basicamente consiste em armazenar as linhas e colunas com o mesmo número de elementos numa lista duplamente ligada com vetores r_i e c_j como cabeças dessas listas. Esta estrutura permite fácil acesso a uma linha ou coluna com um determinado número de elementos especificado. Da mesma forma, facilmente se obtem a estrutura atualizada com a remoção de elementos eliminados com as respectivas alterações dos ponteiros. As posições que armazenam conecções para linhas e colunas vão sendo liberadas

a cada passo da eliminação e podem ser aproveitadas para armazenar outras informações necessárias, como por exemplo, a seqüência de pivôs. A título de observação, o procedimento descrito acima é puramente simbólico, sem utilizar fatores numéricos.

Essa estrutura também permite que a escolha dos pivôs, do ponto de vista numérico, seja feita diretamente. Considere que a matriz seja armazenada por linhas (evidentemente os mesmos argumentos são válidos para colunas). Isso significa que o teste de estabilidade é melhor feito por linhas e, portanto, é melhor utilizar (3.6a). Evidentemente, qualquer linha unitária (com um só elemento) deverá ser escolhida em primeiro lugar pois, é o caso que $M_{ij} = 0$, além de satisfazer o teste de estabilidade (3.6a) para qualquer valor de u. Por outro lado, o elemento de uma coluna unitária pode não ser selecionado para pivô, uma vêz que devem ser considerados os outros elementos na linha para o teste de estabilidade. O primeiro elemento encontrado que satisfaz (3.6a) e tem o menor M_{ij} obtido até então, é considerado um pivô potencial e o problema fica restrito somente em terminar a busca.

O principal problema com a finalização do processo, é que a varredura não garante que os elementos sejam acessados em ordem crescente de M_{ij} e, portanto, a busca não pode ser terminada quando um elemento satisfazendo (3.6a) é encontrado. O exemplo da figura (3.4) ilustra este fato, onde a varredura da linha 4 fornece a_{42} ou a_{43} com $M_{42} = M_{43} = 5$, enquanto que a_{11} ou a_{21} com $M_{11} = M_{21} = 4$, ainda não foram verificados.

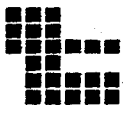


Fig. (3.4) Problema com a finalização da busca.

Entretanto, buscam-se linhas e colunas em ordem crescente de linhas/colunas e, portanto, em qualquer fase da eliminação obtem-se um limitante da quantidade dos elementos não acessados. Assim, estando num estágio buscando linhas com r_i elementos, o mínimo de M_{ij} posteriores é $(r_i - 1)^2$, pois todas as colunas com menos de r_i elementos já foram acessadas. Quando se está numa coluna c_j, o mínimo M_{ij} será $c_j(c_j - 1)$. Assim que o M_{ij} obtido do pivô potencial for menor ou igual ao limite mínimo possível, a busca termina.

Pode se pensar que a busca seja longa. O exemplo abaixo ilustra esse fato, onde os elementos da diagonal são inaceitáveis pelo teste de estabilidade se $u = 1/7$. As linhas e colunas 1 a 4 serão varridas antes que algum elemento fora da diagonal seja acessado como pivô.

```
1       7
  1     7
    1   7
      1 7
7 7 7 7 1
```

Fig. (3.4) Caso de busca longa.

Entretanto, experimentos em [16] mostram que, geralmente, a escolha do pivô é feita após a busca em poucas linhas e colunas.

Zlatev [43] restringe a estratégia de Markowitz, propondo busca por linhas em ordem crescente de quantidade de elementos, restringindo-a para determinado número de linhas (recomendando 3). Poder-se-ia esperar demasiado acréscimo de preenchimento com esse procedimento em relação a estratégia de Markowitz estrita, mas isso não tende a ocorrer. Efetivamente,

esta modificação pode mesmo gerar uma quantidade inferior de fatores do que a estratégia original, pelos motivos ressaltados no final da seção (3.2.1), e geralmente, proporciona economia de tempo computacional. Um exemplo que ilustra a efetividade desse procedimento é dado em [15], onde uma matriz com n = 2000, banda 5 e faixa 42, com elementos 4 na diagonal e -1 fora da diagonal. Markowitz requer 51.0 segundos num IBM 3081K gerando 91487 fatores, enquanto que a modificação de Zlatev requer 25.9 segundos gerando 90797 fatores. Realmente, experimentos mostram bons resultados com esta modificação e a alternativa de Zlatev foi incorporada em versões mais recentes de MA-28.

Como foi discutido anteriormente, a escolha da seqüência pivotal em ANALISE é a etapa responsável pelo desempenho resolução do sistema linear, pois definirá o preenchimento na fatoração, que por sua vez, determinará o desempenho na resolução dos sistemas triangulares.

As fases FATORIZE e SOLVE, não diferenciam-se substancialmente entre as diferentes rotinas de resolução de sistema lineares espasos. Basicamente, envolvem manipulação de vetores esparsos com operações elementares, esquemas de armazenamento e seqüência dos cálculos.

4. MÉTODOS QUASE-NEWTON

Este capítulo trata dos métodos Quase-Newton implementados. Faz-se breve introdução acerca da origem e uma classificação segundo o tipo dos métodos : Atualização Secante com ou sem correção direta da fatoração e aqueles ditos de memória limitada. Finalizando, são apresentadas as principais características de cada um deles

4.1 INTRODUÇÃO

Os métodos Quase-Newton (Veja [1-3, 9-12, 20, 25, 27, 28, 31-34, 38 e 40]) são indicados para situações onde as derivadas analíticas não estão disponíveis ou são difíceis de calcular. Basicamente são estruturados da seguinte forma:

$$B_k s = - F(x^k) \qquad (4.1)$$

$$x^{k+1} = x^k + s. \qquad (4.2)$$

Desse modo, cada iteração é caracterizada pela avaliação da função $F(x^k)$ e da resolução do sistema linear (4.1). Para a iteração seguinte, a matriz B_{k+1} é obtida de B_k utilizando uma fórmula de recorrência envolvendo x^k, x^{k+1}, $F(x^k)$, $F(x^{k+1})$. Frequentemente, B_{k+1} é escolhida como uma das

matrizes que satisfazem a "Equação Secante"(veja [12]) :

$$B_{k+1}s = y \tag{4.3}$$

com

$$y = F(x^{k+1}) - F(x^k)$$

e, nesse sentido, os métodos são denominados Métodos Secantes.

O Método mais conhecido para problemas pequenos e densos é o primeiro método de Broyden [1,3,11,12]. Este método utiliza uma matriz de correção de posto um para obter B_{k+1} de B_k:

$$B_{k+1}s = \frac{(y - B_k s)s^T}{s^T s} \tag{4.4}$$

Utilizando (4.3) e a fórmula de Sherman-Morrison em [21], B_k^{-1} pode ser obtido de B_{k-1}^{-1} usando $O(n^2)$ operações. Desse modo a fatoração Q-R de B_{k+1} pode ser obtida da fatoração Q-R de B_k usando $O(n^2)$ operações [34]. Portanto, se (4.1),(4.2) e (4.4) forem utilizados, além da economia computacional no cálculo das derivadas, haverá também considerável redução no custo do cálculo da solução do sistema linear (4.1). Por esses motivos, o primeiro método de Broyden pode ser mais eficiente do que o método de Newton, mesmo quando as derivadas estão facilmente disponíveis, apesar da sua convergência mais lenta.

No caso de problemas de grande porte a situação não é a mesma. Se B_k é esparsa, e (4.4) é utilizado, B_{k+1} tende a ser densa e pode não ter relação alguma com a estrutura do Jacobiano na iteração k. Broyden [2] e Schubert [33,38] desenvolveram uma variante do primeiro método de Broyden onde as matrizes B_k permanecem com o mesmo padrão de esparsidade de

$J(x^k)$, satisfazendo a equação secante (4.3), e utilizando o princípio da variação mínima. Entretanto, a diferença B_{k+1}- B_k, não é tão grande como no primeiro método de Broyden, e portanto, uma relação simples entre B_k e B_{k+1} parece não ser possível. Além disso, quando o método de Schubert é empregado para atualização de B_k, deve-se considerar a resolução de (4.1) integralmente, que tem, evidentemente, um alto custo computacional.

4.2 MÉTODOS COM ATUALIZAÇÃO DIRETA DA FATORAÇÃO

Embora a motivação principal desses métodos esteja em evitar o cálculo do Jacobiano, alguns deles são desenvolvidos com a perspectiva de uma sensível economia na resolução do sistema linear (4.1): fatoração L-U ($O(n^3)$ flops) e a resolução dos sistemas triangulares ($O(n^2)$ flops); que é, na verdade, a parte mais dispendiosa numa iteração típica.

As observações finais da seção anterior motivaram Dennis e Marwil [9] a desenvolver o primeiro método Quase-Newton onde a fatoração L-U de B_{k+1} é obtida diretamente da fatoração L-U de B_k, proporcionando substancial economia computacional na resolução de (4.1). Basicamente, o método de Dennis-Marwil conserva os fatores da matriz L e modifica os fatores da matriz U, entre as iterações consecutivas, preservando o padrão de esparsidade de U, através de uma fórmula do tipo Schubert. Infelizmente, este método não possui propriedades de convergência completamente satisfatórias. Realmente, a convergência local é obtida somente quando são introduzidos os recomeços, que consistem em, a cada número fixo de iterações, utilizar o Jacobiano

verdadeiro, isto é $B_k = J(x^k)$.

Martinez [30] introduziu um método onde a fatoração LDM^T de B_k é armazenada e somente os fatores da matriz D são modificados para obter a fatoração de B_{k+1}. Este método pertence a uma família de métodos introduzida posteriormente em [31]. Os métodos desta família possuem propriedades de convergência local linear, mas a convergência superlinear é obtida somente com os recomeços.

Chadee [5], generalizou o método de Johnson e Austria [25], introduzindo um método localmente superlinear onde a fatoração L-U de B_{k+1} é obtida modificando simultaneamente os fatores L-U de B_k. Infelizmente, as inversas das matrizes triangulares L devem possuir um padrão de esparsidade definido para que o método de Chadee possa ser utilizado e, portanto, sua aplicabilidade fica restrita a casos de algumas estruturas especiais da matriz Jacobiana.

Martínez em [32] introduziu uma extensa família que inclui, na sua maioria, métodos superlinearmente convergentes para resolução de sistemas de equações não lineares. O método de Dennis-Marwil não pertence a esta família, mas pode ser considerado como um limite de uma subfamília paramétrica que contém o método de Chadee.

4.3 MÉTODOS COM MEMÓRIA LIMITADA

Nos métodos aqui discutidos, a matriz B_{k+1} é obtida adicionando a B_k um fator de correção de posto um:

$$B_{k+1} = B_k + u_k v_k^T \qquad (4.5)$$

A dificuldade da implementação, está no fato de que a matriz B_{k+1} não resulta esparsa mesmo que B_k o seja. Isto significa que a estrutura esparsa de matriz Jacobiana não pode ser explorada e, evidentemente, não há como armazenar as fatorações L-U na estrutura de dados reservada.

Utilizando a fórmula de Sherman-Morrison, obtem-se uma expressão para B_{k+1}^{-1} em funcão de B_k^{-1}, conforme deduzido em [22], de modo a tornar possível a implementação computacional no caso esparso. A matriz inversa de B_{k+1} em (4.5) existe se,

$$(1 + v_k^T B_k^{-1} u_k) \neq 0 \qquad (4.6)$$

e é dada por

$$B_{k+1}^{-1} = B_k^{-1} - \frac{B_k^{-1} u_k v_k B_k^{-1}}{1 + v_k^T B_k^{-1} u_k} . \qquad (4.7)$$

Definindo,

$$w_k = \frac{B_k^{-1} u_k}{1 + v_k^T B_k^{-1} u_k} \qquad (4.8)$$

pode-se escrever,

$$B_{k+1}^{-1} = (I - w_k v_k^T) B_k^{-1} \qquad (4.9)$$

aplicanmdo as fórmulas (4.8) e (4.9), obtem-se :

$$B_{k+1}^{-1} = (I - \omega_k \upsilon_k^T) (I - \omega_{k-1} \upsilon_{k-1}^T) \ldots (I_o - \omega_o \upsilon_o^T) B_o^{-1} \qquad (4.10)$$

Os vetores adicionais, ω_k e υ_k, calculados à cada iteração, precisam ser armazenados para as iterações subsequentes. Por este motivo, o número de iterações consecutivas de cada método é restrita pelo espaço de memória disponível reservado para estes vetores. Outra limitação a considerar, é que o esforço computacional aumenta a cada iteração. Portanto, é necessário estabelecer critérios para recomeços, tanto pela disponibilidade de armazenamento como pelo esforço computacional da iteração em questão.

O vetor s, solução de $B_k s = -F(x^k)$ é obtido aplicando (4.10) para efetuar o produto $B_k^{-1}(F(x^k))$

4.4 CARACTERÍSTICAS DOS MÉTODOS

Neste trabalho é feita uma comparação entre o método de Newton, o método de Newton Modificado, o método de Schubert, o método de Dennis-Marwil, três métodos da família introduzida em [30,31] e dois métodos de memória limitada. O principal objetivo entretanto, é a comparação do emprego desses métodos segundo maneiras distintas de efetuar a fatoração (veja cap. 5).

Além de (4.1) e (4.2), os algorítmos basicamente contém a atualização de B_k, um teste para detectar (modificar) possível singularidade nos fatores das matrizes envolvidas, controle de passo e o teste de parada (cap. 5). Possuem também a primeira iteração comum, que é Newton, e são introduzidos os recomeços (seção 4.2).

Em seguida, os métodos são apresentados resumidamente, caracterizando uma iteração típica e o tipo de convergência que eles possuem.

1. NEWTON : $B_k = J(x^k)$

- Cálculo do Jacobiano
- Resolução do sistema linear
- Convergência quadrática

2. NEWTON MODIFICADO : $B_k = J(x^0)$

- Resolução dos sistemas triangulares em (2.2)
- Convergência linear, superlinear com recomeços

3. SCHUBERT : B_k pela fórmula de Schubert [38]

- Atualização de B_k
- Resolução do sistema linear
- Convergência superlinear

4. DENNIS-MARWIL : $B_K = L \cdot U_K$

- Atualização de U_k
- Resolução dos sistemas triangulares em (2.2)
- Convergência linear com recomeços[1]

5. ATUALIZAÇÃO DO FATOR DIAGONAL : $B_k = L \cdot D_k \cdot U$

6. ESCALAMENTO DE LINHAS : $B_k = D_k \cdot L \cdot U$

7. ESCALAMENTO DE COLUNAS : $B_k = L \cdot U \cdot D_k$

(para os métodos 2.5, 2.6 e 2.7)

- Atualização de D_k
- Resolução dos sistemas triangulares em (2.2)
- Convergência linear, superlinear com recomeços

---

[1] A princípio, o método de Dennis-Marwil não possui boas propriedades de convergência e, na verdade, a convergência linear só é obtida com recomeços. Entretanto ele é o limite de uma família de métodos que tem convergência superlinear e tende, portanto, a ter este tipo de comportamento na prática (veja [9]).

8. BROYDEN : B_k por (4.5)

- Atualização de B_{k+1}^{-1} com $w_k = \dfrac{B_k^{-1}F(x^{k+1})}{s_k^T\,(s_k + B_k^{-1}F(x^{k+1}))}$

$$v_k = s_k$$

- Resolução de sistemas triangulares em (2.2)
- 2n posições de memória adicionais por iteração
- Convergência superlinear.

9. CUM (Column Updating Method [23]) : B_k por (4.5)

- Atualização de B_{k+1}^{-1} com $w_k = \dfrac{B_k^{-1}F(x^{k+1})}{e_{jk}^T\,(s_k + B_k^{-1}F(x^{k+1}))}$

$$v_k = e_{jk}$$

onde $j = \text{Argmax}\ [\ |s_i|\ ,\ i = 1,\ldots,n\ ]$

- Resolução de sistemas triangulares em (2.2)
- n+1 posições de memória adicionais por iteração
- Convergência linear, superlinear com recomeços

5. EXPERIMENTOS NUMÉRICOS

Este capítulo contém os testes numéricos realizados. É feita a comparação entre os métodos descritos no capítulo 4, bem como os desempenhos de SNLDIN e SNLUC. Os experimentos envolvem cinco problemas. Os quatro primeiros são provenientes da literatura científica clássica e o último é um problema de fluxo de cargas em redes de potência.

5.1 INTRODUÇÃO

Como foi salientado anteriormente o objetivo principal é a comparação do desempenho dos métodos, segundo maneiras distintas de resolução dos subproblemas, utilizando estrutura dinâmica : SNLDIN, ou estrutura estática : SNLUC. Na implementação de SNLDIN, as rotinas Quase-Newton de SNLUC em [24] são adaptadas à estrutura de dados de MA-28.

Em todos os métodos, a primeira iteração é Newton. O critério de parada consiste em :

$$\|F(x^k)\|_\infty < E_1 \qquad \text{ou} \qquad \|s\|_\infty < E_2$$

para convergência, e

$$\|F(x^k)\|_\infty > BIG$$

para divergência,

onde E_1, E_2 e *BIG* são parâmetros fornecidos pelo usuário. A execução também é interrompida quando excede um tempo determinado de CPU ou quando excede um número ITMAX de iterações. Também é introduzido o parâmetro de perturbação para singularidade na fatoração, que consiste em, cada vez que um elemento da diagonal for menor que TOL, este elemento é substituido por este parâmetro. O controle para inibir passos grandes é através de XTOL, se $\|s\|_\infty >$ XTOL então s é substituido por

$$s \cdot \frac{\text{XTOL}}{\|s\|_\infty} \quad .$$

Cada uma das rotinas tem características próprias em relação às fases de resolução dos sistemas lineares :

SNLDIN

A fatoração LU esparsa é feita utilizando a estratégia de Markowitz com parâmetro de tolerância, como descrito no capítulo 3. Permite o emprego da fatoração com a seqüência pivotal pré-fixada na primeira iteração Newton, pois, os sistemas não lineares mantém a mesma estrutura da matriz Jacobiana para todas as iterações. Portanto ,uma iteração Newton ou Schubert, que resolve o sistema linear a cada iteração, pode ser executada de duas maneiras. Como não houve casos de instabilidade numérica com o uso das seqüências pivotais pré-fixadas, e devido a economia proporcionada, esta opção foi adotada. O desempenho sem o uso desta facilidade pode ser obtido, observando o número de iterações e o tempo de uma iteração Newton ou Schubert com escolha da seqüência pivotal, que são relacionadas juntamente com as tabelas dos

experimentos.

SNLUC [22]

Em SNLUC a fatoração é executada com as três fases distintas como foi salientado no capítulo 3. SNLUC executa, primeiramente, uma fase simbólica utilizando o algoritmo de George e Ng [19] somente com a estrutura da matriz Jacobiana. Fixa a estrutura de dados que armazenará qualquer fatoração com permutação de linhas, executada com quaisquer conjunto de valores numéricos. Portanto, SNLUC também considera o fato de que as estruturas das matrizes jacobianas ficam fixas. Assim a fase simbólica é executada no início da resolução do problema e, posteriormente, são executadas as fatorações esparsas com pivotamento parcial na estrutura fixada.

Todos os testes foram executados no VAX11/785 da Universidade Estadual de Campinas, utilizando o compilador FORTRAN77. As medidas de tempo, contidas no corpo das tabelas, são tomadas em segundos de CPU.

Para interpretação dos resultados define-se:

N : dimensão do SNL.
SNLUC : desempenho de SNLUC.
SNLDIN : desempenho de SNLDIN.
ITER : número de iterações.
T.ITER : tempo de uma iteração Quase-Newton (ou Newton para o método de Newton).
T.TOTAL : tempo total de execução.
TFATSIMB : tempo da fatoração simbólica em SNLUC.
TNEW : tempo da primeira iteração Newton em SNLDIN(i.é. com a escolha de pivôs).

TSCHU : tempo de uma iteração Schubert com escolha da seqüência pivotal.

O tempo das iterações Newton e Schubert das tabelas referem-se ao tempo de uma iteração com seqüência pivotal pré-fixada.

Os problemas testes são mostrados a seguir com as respectivas estruturas dos Jacobianos. Para todos os problemas o ponto inicial é $x^0 = (-1,\ldots,-1)^T$, foram fixados $E_1 = E_2 = 10^{-4}$, ITMAX = 100, TOL = 10^{-7} e XTOL = 10 . As exceções são: para o problema 5, x^0 tem valores 0 e 1 para as componenentes θ e V respectivamente e ITMAX = 15 e $E_1 = 10^{-2}$ para N = 2190 e XTOL = 5 para o problema 4.

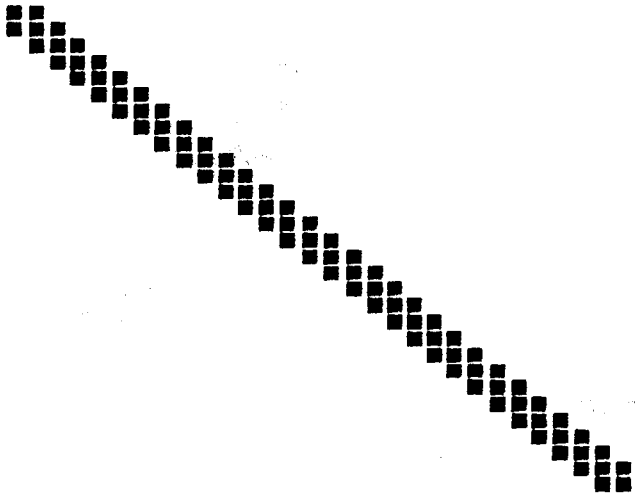
5.2 PROBLEMA 1 (Broyden Tridiagonal [1,2])

$$f_1(x) = (3 - 2x_1)x_1 - 2x_2 + 1$$

$$f_i(x) = (3 - 2x_i)x_i - x_{i-1} - 2x_{i+1} + 1 \qquad i = 2,\ldots,n-1$$

$$f_n(x) = (3 - 2x_n)x_n - x_{n-1} + 1$$

Estrutura do Jacobiano para N = 30



Resultados para N = 5000

	SNLDIN			SNLUC		
MÉTODO	ITER	T.ITER	T.TOTAL	ITER	T.ITER	T.TOTAL
NEWTON	3	4.69	19.25	3	1.23	5.82
NEW. MOD.	9	0.44	13.39	9	0.57	6.93
SCHUBERT	6	4.53	32.52	6	1.57	11.01
D. MARWIL	5	0.80	13.07	5	0.84	6.52
LINHA	5	0.43	11.59	6	0.58.	6.08
COLUNA	5	0.45	11.67	5	0.58	5.51
DIAGONAL	5	0.44	11.63	5	0.61	5.63
BROYDEN	5	0.63	12.39	5	0.98	7.09
CUM	5	0.57	12.15	5	0.87	6.64

TFATSIMB = 1.85 seg.

ITNEW = 9.87 seg.

TSCHU = 10.05 seg.

Neste problema pode-se observar nítida vantagem de SNLUC em relação a SNLDIN para todos os métodos. Observe, entretanto, que todas as iterações Quase-Newton, exceto Schubert, são ligeiramente mais econômicas em SNLDIN. A vantagem para SNLUC ocorre em razão do alto custo da primeira iteração Newton em SNLDIN. Mesmo as iterações Newton e Schubert com a seqüência pivotal pré-fixada, que neste caso requerem cerca de metade do tempo de uma iteração com escolha de pivôs, são consideravelmente mais dispendiosas em SNLDIN do que uma iteração em SNLUC. Isto faz com que a diferença se acentue ainda mais nesses métodos.

O desempenho em SNLDIN não teve alterações para diferentes valores de u, o parâmetro de tolerância. Isto deve-se ao fato de que a matriz é carregada na diagonal, além, é claro, da própria estrutura tridiagonal.

Neste problema, os métodos que tiveram melhor desempenho foram Linha, Coluna, Diagonal e New. Mod. seguidos de Broyden, CUM e Dennis-Marwil. Os métodos Schubert e Newton não obtiveram bom desempenho em relação aos outros métodos; ainda, devido ao alto custo das iterações destes métodos, os recomeços só foram convenientes em Schubert com duas iterações Newton e duas Schubert com 7.92 seg. em SNLUC e 23.46 em SNLDIN.

Finalmente, para diferentes dimensões, como por exemplo 1000 a 6000, os resultados tem, proporcionalmente, o mesmo comportamento.

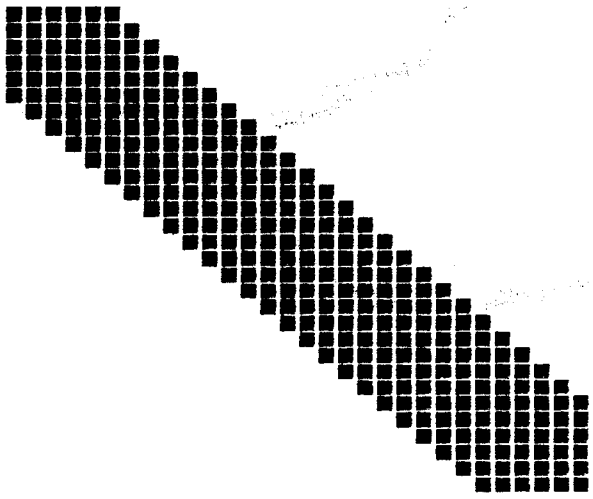
5.3 PROBLEMA 2 (Broyden Banda [2])

$$f_i(x) = (3 - 5x_i^2)x_i + 1 - \sum_{j \in I_i} (x_j + x_j^2) \qquad i = 1,\ldots,n$$

onde $I_i = [i_1,\ldots,i_2] - [i]$,

$i_1 = \max[1, i - 5]$, $i_2 = \min[n, i + 5]$

Estrutura do jacobiano para N = 30



Resultados para N = 5000

	SNLDIN			SNLUC		
MÉTODO	ITER	T.ITER	T.TOTAL	ITER	T.ITER	T.TOTAL
NEWTON	4	19.92	94.57	4	4.97	26.21
NEW. MOD.	17	1.45	58.01	17	1.17	30.01
SCHUBERT	9	16.29	165.13	9	5.60	56.11
D. MARWIL	11	2.06	55.41	11	2.01	31.41
LINHA	6	1.29	41.26	6	1.30	17.81
COLUNA	6	1.30	41.31	6	1.30	17.81
DIAGONAL	6	1.29	41.26	6	1.30	17.81
BROYDEN	9	1.58	47.45	9	2.24	22.91
CUM	9	1.51	46.89	9	2.42	24.37

TFATSIMB = 6.32 seg.
ITNEW = 34.81 seg.
TSCHU = 34.70 seg.

Praticamente todas as observações do problema 1 são pertinentes neste caso, como pode ser observado na tabela acima. Note que as iterações Quase-Newton, exceto Schubert, demandam o mesmo tempo, em média, em SNLDIN e SNLUC.

Os recomeços foram convenientes em Schubert com três iterações Newton e duas Schubert com 34.51 seg. em SNLUC e 102.57 seg. em SNLDIN. Também em Dennis-Marwil com duas iterações Newton e seis Quase-Newton com 27.71 seg. em SNLUC.

5.4 PROBLEMA 3

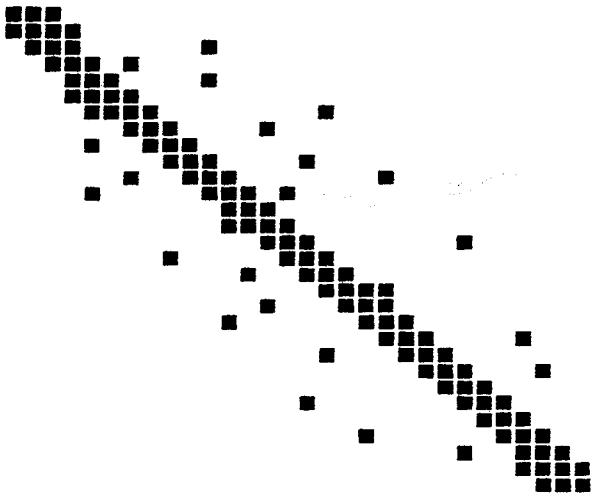
$$f_1(x) = (3 - 2x_1)x_1 - 2x_2 + 0.5x_{\alpha 1} + 1$$

$$f_i(x) = (3 - 2x_i)x_i - x_{i-1} - 2x_{i+1} + 0.5x_{\alpha j} + 1 \quad i = 2,\ldots,n-1$$

$$f_n(x) = (3 - 2x_n)x_n - x_{n-1} + 0.5x_{\alpha n} + 1 \ ,$$

para p_j, $j = 1,2,\ldots,n$, escolhido aleatóriamente nos intervalos $\alpha_{jmin} = \max[1, j - b]$ e $\alpha_{jmax} = \min[n, j + b]$ para um parâmetro b que define a faixa.

Estrutura do Jacobiano para N = 30, b = 10



Resultados para N = 1000 e b = 100.

	SNLDIN			SNLUC		
MÉTODO	ITER	T.ITER	T.TOTAL	ITER	T.ITER	T.TOTAL
NEWTON	4	10.78	95.51	4	9.70	46.69
NEW. MOD.	11	0.26	65.77	11	0.67	24.45
SCHUBERT	6	10.69	116.62	6	9.68	64.89
D. MARWIL	7	0.54	66.41	7	1.23	25.10
LINHA	6	0.24	64.37	6	0.70	21.43
COLUNA	6	0.26	64.47	6	0.78	21.47
DIAGONAL	6	0.28	62.91	6	0.78	21.47
BROYDEN	7	0.29	64.91	7	0.80	22.37
CUM	7	0.27	64.79	7	0.73	21.15

TFATSIMB = 8.94 seg.
ITNEW = 63.17 seg.
TSCHU = 62.64 seg.

Como ocorreu nos problemas anteriores, o desempenho SNLUC para o problema 3 foi substancialmente superior, ocasionado pelo alto custo da primeira iteração Newton em SNLDIN. Note, agora, que as iterações Quase-Newton (exceto Schubert) em SNLDIN tem custo bem inferior a SNLUC.

Em SNLDIN não houve variações consideráveis para diferentes valores de u. Uma iteração Newton (ou Schubert) com os pivôs fixos, é cerca de 1/4 de uma iteração selecionando os pivôs.

Os métodos que tiveram melhor desempenho foram

Linha, Coluna, Diagonal, Broyden e CUM, seguidos de New. Mod. e Dennis-Marwil. Novamente os métodos Newton e Schubert não mostraram resultados satisfatórios. Os recomeços foram convenientes em Schubert com duas iterações Newton e duas Schubert com 46.60 seg. em SNLUC e 90.46 em SNLDIN.

5.5 PROBLEMA 4 (Poisson [38])

Esse problema é o sistema de equações não lineares proveniente da discretização do problema de valor de contorno de Poisson, com L subdivisões ($N = L^2$).

$$\Delta u = \frac{u^3}{1 + s^2 + t^2} \quad , \quad 0 \le s \le 1, \quad 0 \le t \le 1$$

$$u(0,t) = 1 ,$$

$$u(1,t) = 2 - e^t , \qquad t \in [0,1]$$

$$u(s,0) = 1 ,$$

$$u(1,s) = 2 - e^s , \qquad s \in [0,1]$$

Estrutura do Jacobiano para N = 36 (L = 6)

Resultados para N = 225

	SNLDIN			SNLUC		
MÉTODO	ITER	T.ITER	T.TOTAL	ITER	T.ITER	T.TOTAL
NEWTON	3	0.76	3.81	3	0.66	2.87
NEW. MOD.	5	0.04	2.45	5	0.10	2.05
SCHUBERT	5	0.73	5.21	4	0.68	3.59
D. MARWIL	5	0.08	2.61	5	0.14	2.08
LINHA	6	0.05	2.54	6	0.10	2.01
COLUNA	8	0.04	2.57	6	0.11	2.10
DIAGONAL	6	0.04	2.54	6	0.13	2.21
BROYDEN	4	0.05	2.44	4	0.13	2.12
CUM	5	0.03	2.45	5	0.10	1.95

TFATSIMB = 0.89 seg.

ITNEW = 2.29 seg.

TSCHU = 2.19 seg.

Resultados para N = 961

	SNLDIN			SNLUC		
MÉTODO	ITER	T.ITER	T.TOTAL	ITER	T.ITER	T.TOTAL
NEWTON	4	5.57	35.42	4	10.0	47.51
NEW. MOD.	5	0.20	19.51	5	0.77	20.94
SCHUBERT	5	5.56	40.95	5	8.29	56.57
D. MARWIL	4	0.43	20.00	5	1.52	23.63
LINHA	5	0.22	19.59	5	0.84	21.30
COLUNA	6	0.27	19.76	6	0.69	21.57
DIAGONAL	5	0.25	19.71	8	0.88	23.50
BROYDEN	4	0.26	19.49	4	1.02	20.62
CUM	5	0.23	19.63	5	0.85	20.04

TFATSIMB = 8.55 seg.

ITNEW = 18.71 seg.

TSCHU = 19.30 seg.

Resultados para N = 1600

	S N L D I N			S N L U C		
MÉTODO	ITER	T.ITER	T.TOTAL	ITER	T.ITER	T.TOTAL
NEWTON	4	12.91	81.02	4	29.30	137.03
NEW. MOD.	5	0.40	43.89	5	2.25	58.49
SCHUBERT	5	12.90	93.89	5	29.35	167.22
D. MARWIL	4	0.85	44.84	5	4.15	66.05
LINHA	5	0.41	43.93	5	2.38	58.36
COLUNA	6	0.42	44.39	6	2.41	60.67
DIAGONAL	7	0.43	44.87	9	2.39	62.78
BROYDEN	4	0.43	43.58	4	2.51	56.40
CUM	5	0.41	43.93	5	2.42	55.84

TFATSIMB = 19.83 seg.

ITNEW = 42.29 seg.

TSCHU = 43.05 seg.

Neste problema verifica-se melhor desempenho de SNLDIN em relação à SNLUC. Esta vantagem acentua-se na medida do acréscimo da dimensão do problema. Isto deve-se ao padrão de esparsidade, uma vez que as faixas tem distância L da diagonal principal.

Observe que o custo de uma iteração Quase-Newton em SNLDIN é substancialmente mais econômica que em SNLUC, mesmo para N = 225, onde SNLUC tem ligeira vantagem no tempo total. Deve-se notar, entretanto, o alto custo das iterações Newton e

Schubert com escolha de pivós. Por outro lado, as iterações com os pivôs fixos são cerca de 1/2 para N = 225, e 1/3 para N = 961 e N = 1600, mais econômicas do que aquelas, e por este motivo tem custo mais baixo que SNLUC.

Em SNLDIN, os resultados não tiveram variações substanciais para diferentes valores do parâmetro de tolerância.

Neste problema os métodos Linha, Coluna, Diagonal, New.Mod., Broyden, CUM e Dennis-Marwil tiveram resultados semelhantes com bom desempenho. Como nos três primeiros problemas, Newton e Schubert não tiveram desempenho satisfatório.

Os recomeços proporcionam vantagem somente em Schubert. Para N = 961, por exemplo, requer duas iterações Newton e duas Schubert, com 35.40 seg. e 45.13 seg. para SNLUC e SNLDIN respectivamente.

5.6 PROBLEMA 5 (Fluxo de Cargas[17][1])

1

O problema (P) representa o sistema de equações não lineares típico para o problema de Fluxo de Carga. Na verdade, a formulação com maiores detalhes pode ser verificada em [17]. Omitem-se aqui somente as especificações técnicas dos termos e posterior manipulação com outras grandezas associadas.

$$(P)\quad \begin{cases} \Delta P_k(\theta,V) = P_k^{esp} - P_k(\theta,V) = 0 \\ \\ \Delta Q_k(\theta,V) = Q_k^{esp} - Q_k(\theta,V) = 0 \end{cases}$$

com,

$$\Delta P_k(\theta,V) = V_k \sum_{m \in K_k} V_m \left(G_{km}\cos\theta_{km} + B_{km}\operatorname{sen}\theta_{km}\right)$$

$$\Delta Q_k(\theta,V) = V_k \sum_{m \in K_k} V_m \left(G_{km}\operatorname{sen}\theta_{km} + B_{km}\cos\theta_{km}\right)$$

onde,

V_k e θ_k são as variáveis

P_k e Q_k são constantes

G_{km} e B_{km} são constantes específicas que dependem do problema e fornecem o padrão de esparsidade. São nulas quando não houver conecção (ou barras) k - m na rede de fluxo de cargas.

K_k define o conjunto na rede no nó k.

Estrutura do jacobiano para N = 54

Resultados para N = 54 (30 barras)

	SNLDIN			SNLUC		
MÉTODO	ITER	T.ITER	T.TOTAL	ITER	T.ITER	T.TOTAL
NEWTON	3	0.19	0.85	3	0.21	0.81
NEW. MOD.	5	0.03	0.59	5	0.04	0.58
SCHUBERT	5	0.15	1.07	5	0.19	0.93
D. MARWIL	5	0.03	0.59	6	0.10	0.70
LINHA	7	0.03	0.65	7	0.09	0.74
COLUNA	21	0.03	1.07	26	0.06	2.04
DIAGONAL	9	0.03	0.71	14	0.07	1.09
BROYDEN	5	0.03	0.59	5	0.11	0.83
CUM	4	0.03	0.56	4	0.12	0.75

TFATSIMB = 0.18 seg.

ITNEW = 0.47 seg.

TSCHU = 0.36 seg.

Resultados pa N = 182 (118 barras)

	SNLDIN			SNLUC		
MÉTODO	ITER	T.ITER	T.TOTAL	ITER	T.ITER	T.TOTAL
NEWTON	3	1.28	4.35	3	1.65	6.09
NEW. MOD.	6	0.34	3.49	6	0.70	4.46
SCHUBERT	6	0.67	5.14	6	1.35	7.89
D. MARWIL	5	0.37	3.27	7	0.74	4.88
LINHA	23	0.33	9.05	12	0.45	8.60
COLUNA	--	----	-------	--	----	-------
DIAGONAL	9	0.34	4.51	23	0.47	11.48
BROYDEN	5	0.38	3.31	5	0.81	4.36
CUM	6	0.34	3.49	6	0.73	4.80

TFATSIMB = 1.15 seg.

ITNEW = 1.79 seg.

TSCHU = 1.10 seg.

Resultados para N = 2190 (1138 barras)

	SNLDIN			SNLUC		
MÉTODO	ITER	T.ITER	T.TOTAL	ITER	T.ITER	T.TOTAL
NEWTON	4	102.82	419.40	4	587.45	2470.51
BROYDEN	12	29.93	440.17	12	120.38	1212.73
CUM	12	29.05	430.49	12	113.51	1186.71

TFATSIMB = 120.71 seg

TNEW = 110.94 seg.

A convergência não se deu nos métodos que não foram relacionados, como por exemplo na tabela para N = 2190, onde somente os métodos Newton, Broyden e CUM convergiram. As lacunas que não contém números também significam divergência, como o caso do método Coluna em SNLUC para N = 54.

Observe em geral a melhor desempenho SNLDIN para o problema de Fluxo de Cargas. Isto é justificado pelo padrão de esparsidade da matriz Jacobiana, que produz excessivo preenchimento na fatoração simbólica em SNLUC, o que pode ser comprovado pela diferença de desempenho nas iterações Quase-Newton.

Em SNLDIN houve algumas pequenas variações no desempenho para diferentes valores de u, cujo valor na tabela corresponde a 0.3. Uma iteração do método de Newton para N = 2190 com u = 1.0 demanda 138.06 seg.. Neste problema a razão entre uma iteração Newton e Schubert com os pivôs fixos e uma iteração com escolha de pivôs é cerca de 0.4 para N = 54, 0.65 para N = 182 e 0.91 para N = 2190.

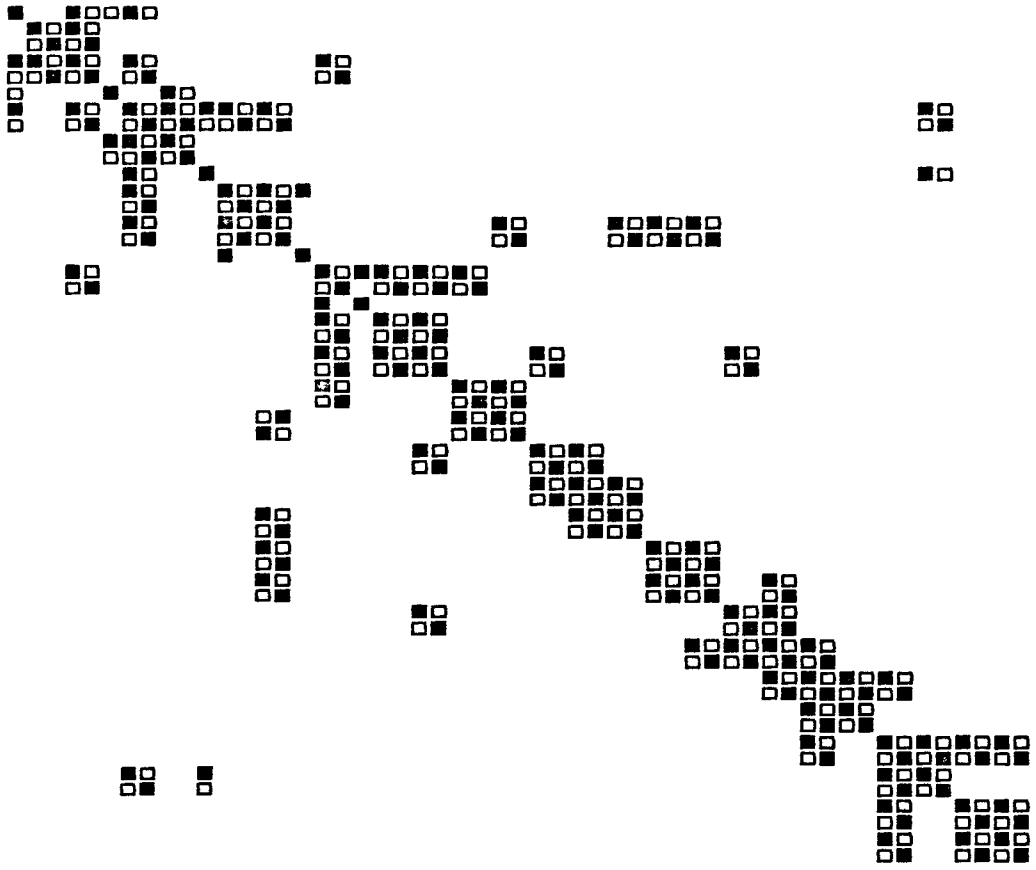
Para N = 54 e N = 182 os melhores métodos são CUM, Broyden e New. Mod. seguidos de Dennis-Marwil e posteriormente Newton e Schubert. Para N = 2190 somente CUM, Broyden e Newton convergem com superior desempenho para os dois primeiros. Em geral, os métodos Linha, Coluna e Diagonal não apresentam resultados satisfatórios para este problema, não convergindo na maioria dos casos. Os recomeços proporcionam pequeno benefício somente em Schubert, como nos exemplos anteriores.

Observando a estrutura da matriz jacobiana para o Problema de Fluxo de Cargas e o desempenho dos métodos, principalmente em SNLUC, pode-se cogitar que o grau de preenchimento seja elevado. Realmente, SNLUC na fase de fatoração simbólica para N = 182, prevê um máximo de 20.7% de

densidade para a fatoração na estrutura original de 3.18%, enquanto que MA-28 em SNLDIN, produz a matriz fatorada de 4.1%.

Pode-se cogitar se os resultados melhorariam caso alguns componentes da matriz Jacobiana fossem abandonados e a matriz resultante fosse utilizada como aproximação para o Jacobiano verdadeiro. Esta é uma opção de DURAN em [17], que elimina alguns elementos para melhorar o padrão de esparsidade da matriz, através de um critério técnico. Obviamente, neste caso, não se usa o Jacobiano verdadeiro e, por este motivo, denominar-se-á o experimento com o Jacobiano falso. Da mesma forma o método de Newton será agora denominado Newton Falso.

Estrutura do Jacobiano Falso para N = 54



Resultados para N = 54 (30 barras)

	S N L D I N			S N L U C		
MÉTODO	ITER	T.ITER	T.TOTAL	ITER	T.ITER	T.TOTAL
NEW. FALSO	17	0.14	2.49	17	0.14	2.41
NEW. MOD.	13	0.02	0.49	13	0.06	0.84
BROYDEN	10	0.03	0.52	10	0.06	0.71
CUM	9	0.02	0.41	9	0.07	0.71

TFATSIMB = 0.12 seg.

ITNEW = 0.25 seg.

Resultados para N = 182 (118 barras)

	S N L D I N			S N L U C		
MÉTODO	ITER	T.ITER	T.TOTAL	ITER	T.ITER	T.TOTAL
NEW. FALSO	9	1.17	10.70	9	1.44	12.25
NEW. MOD.	9	0.33	3.98	9	0.33	4.79
BROYDEN	8	0.34	3.72	9	0.33	4.54
CUM	8	0.33	3.65	8	0.30	4.21

TFATSIMB = 0.73 seg.

ITNEW = 1.34 seg.

Neste caso só se obteve convergência para os métodos Newton Falso, Newton Modificado Broyden e CUM. Em relação ao

jacobiano verdadeiro obtem-se ligeira vantagem em N = 54 com os métodos New. Mod, Broyden e CUM em SNLDIN. Para N = 182 ocorre desempenho semelhante para os mesmos métodos e, para N = 2190, o emprego do jacobiano falso mostrou-se ineficiente, não se obtendo convergência.

SNLDIN tem desempenho superior neste caso, como pode ser observado nas tabelas, com praticamente as mesmas justificativas para o caso do jacobiano verdadeiro.

5.7 OBSERVAÇÕES

De modo geral, os experimentos consubstanciam a tônica do capítulo 3, no que diz respeito ao trabalho de escolha dos pivôs na eliminação esparsa.

Em síntese ressalta-se:

i) Simplificar a busca dos pivôs numa estrutura previamente fixada, em detrimento de uma fatoração numérica que pode ter alto custo caso o preenchimento seja demasiado (SNLUC).

ii) Apostar numa criteriosa seleção de pivôs, simultaneamente definindo a estrutura, no sentido de obter a fatoração numérica com custo reduzido (SNLDIN).

O desempenho bem superior de SNLUC nos dois primeiros problemas, é justificado pelo grau de esparsidade da matriz fatorada, que é o mesmo para SNLUC e SNLDIN, sendo que o último executa superior esforço para a escolha dos mesmos

pivôs. Fato não muito diferente ocorre no problema 3 onde, embora a estrutura seja pior, os valores numéricos favorecem a escolha de pivôs com a estratégia de SNLUC.

Nos dois últimos problemas a situação é oposta, principalmente no problema 5. Nestes casos ressalta-se a importância de preocupar-se com a esparsidade do problema, empreendendo o trabalho mais elaborado na seleção da seqüência pivotal, buscando um elemento numericamente razoável e que não introduza muito preenchimento.

Finalmente, deve-se salientar que, quando se utiliza a mesma seqüência pivotal em SNLDIN, fica caracterizado o emprego de uma estrutura estática nas iterações Newton e Schubert. De qualquer forma, esta é uma opção de MA-28 e tem sentido lançar mão desta facilidade após excessivo trabalho na escolha de pivôs na primeira iteração.

6. CONCLUSÕES E FUTURAS POSSIBILIDADES

A utilização da estrutura dinâmica de dados na resolução de sistemas não lineares esparsos, mostra-se conveniente quando os elementos da matriz Jacobiana estão mais distribuidos, ou seja, quanto pior for o padrão de esparsidade.

Por outro lado, como pode ser verificado, esta técnica tem um elevado custo computacional para problemas onde a estrutura, embora esparsa, não tenha muita aleatoriedade; ou ainda, tenha seus elementos concentrados relativamente próximos a diagonal. Nesses casos estratégias simplificadoras de pivoteamento são aconselháveis.

Evidentemente, cada problema tem suas próprias características e, para resolvê-los muitas vezes há rotinas bem específicas. Para resolver problemas com estrura banda, por exemplo (como 1 e 2), deve-se utilizar um procedimento adequado para tal; da mesma forma como deve-se aproveitar a estrutura simétrica do problema de Fluxo de Cargas; e é muito sabido que para problemas provenientes da discretização de Equações Diferencias, como o problema Poisson, os métodos iterativos são mais adequados. A opção da alternativa mais oportuna, depende do problema considerado.

O espírito do trabalho, entretanto, é o desenvolvimento e avaliação de métodos de carater genérico e muitos problemas da vida real se enquadram nesse contexto, de modo que acredita-se aqui, ter obtido os objetivos pretendidos

Até o momento, todos os métodos desenvolvidos no

projeto da UNICAMP se baseiam na resolução direta do sistema linear por fatoração LU. Para determinadas estruturas do Jacobiano isto é impraticável por conduzir a excessivo preenchimento e consequentemente a exagerado tempo computacional e uso de memória. Nesses casos, os sistemas lineares devem ser resolvidos usando métodos iterativos, o que da origem aos métodos de Newton Inexatos. Agora, os métodos iterativos são intoleravelmente lentos, se não forem pré-condicionados adequadamente [21]. Muitos trabalhos publicados em revistas de Física Computacional e Análise Numérica, consistem essencialmente na aplicação de pré-condicionadores desenvolvidos para problemas específicos. Martínez [29] desenvolveu recentemente uma teoria completa sobre métodos de Newton Inexatos com pré-condicionadores gerados por fórmulas secantes.

O direcionamento futuro do tema deste trabalho, consiste em desenvolver praticamente as idéias de [29] e resolver problemas teóricos em aberto. Especificamente pode-se vislumbrar os seguintes assuntos de pesquisa:

i) Uso de pré-condicionadores secantes em conjunção com métodos de Gradiente Conjugados para o sistema linear: Extensão da teoria de [29] para o CUM como pré-condicionador.

ii) Resolução do sistema linear : Direções conjugadas ou um método secante. Teoria : extensão (os Secantes não reduzem norma do erro de sistema linear como é exigido em [29]).

iii) Escolha do pré-condicionador Secante. Aplicabilidade de Broyden 2 [1,11]. Desenvolvimento de um método "CUM inverso".

iv) Pré-condicionadores Secantes versus pré-condicionadores baseados na matriz.

v) Fatoração inicial para inicializar o método pré-condicionador secante : Jacobiano truncado talvez, Escalamento ad hoc ?

vi) Implementações computacionais e incorporação ao pacote SNLUC [24,23].

APÊNDICE

IMPLEMENTAÇÃO COMPUTACIONAL - PROGRAMAS

Na implementação de SNLDIN as rotinas Quase-Newton em [22] são adequadas à estrutura de dados de MA-28, além de considerarem o efeito das permutações de colunas geradas na decomposição.

SNLDIN é elaborado de forma que um programa principal deve chamá-lo como subrotina, que também deve conter definições de dimensões e parâmetros (como os descritos no cap.5), e outra chamada de uma rotina que define a estrutura do Jacobiano. Além da rotina ESTRUTURA (veja listagem), deve-se elaborar subrotinas que definem a função e o Jacobiano que são chamadas a cada iteração por SNLDIN. A listagem anexa também contém o exemplo do problema 3, onde pode-se verificar o esquema de SNLDIN.

O parâmetro NELDECOMP, define a dimensão do vetor extendido que armazenará a decomposição da matriz Jacobiana; quando for insuficiente é necessário incrementá-lo.

As chamadas de MA-28 se dão através das rotinas Newton com:

F01BRF : para iterações com escolhas de pivôs.

F01BSF : para iterações com os pivôs fixos.

Os parâmetros de MA-28 são definidos na subrotina

MESTRA e são especificados em [13]. Os parâmetros ETA, RPMIN e GROW fornecem o controle de estabilidade e estimativas de condicionamento como salientado no capítulo 2 (GROW, por exemplo, fornece uma estimativa do tipo ρ definido na seção 2.2).

```
*     PROGRAMA PARA TESTAR O SNL BROYDEN - 3
*-----------------------------------------------------------------------
*
      COMMON / INFORMACOES / N,NEL,NELDECOMP,U
      COMMON / TOTERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / FAIXA       / LF
      COMMON / SEMENTE     / ISEM
      COMMON / PARAM       / PQ,PE
      COMMON / VETICN      / ICNO( 5000 )
      COMMON / NTESTE      / NTES1
*
      REAL      * 4    X( 5000 )
      INTEGER   * 4    IVECT(5000),JVECT(5000)
      CHARACTER*20 NTES1
*
      Type 10
10    Format(////,5x,' ENTRE COM A DIMENSAO DO SNL ---> :  ',$ )
      Accept * , N
c     Type 15
15    Format( //,5x,' ENTRE COM  A  SEMENTE, ISEM ---> :  ',$ )
c     Accept *, Isem
      Type 17
17    Format( //,5x,' ENTRE   COM   A    FAIXA    ---> :  ',$)
      Accept *, LF
      type 19
19    FORMAT(//,5X,' ENTRE C/ O PARAMETRO DE TOLERANCIA ---> : ',$)
      ACCEPT *, U
*
      NTES1 = ' BROYDEN3 '
      ISEM  = 1131
      PQ    = 2.0
      PE    = 0.5
      Xtol  = 1.E-04
      Ftol  = 1.E-04
      Fmax  = 1.E+10
      Distx = 10.0
      Itmax = 50
      XZERO = -1.0
*
      NEL = 30000
      NELDECOMP = NEL + 5*N
*
      DO I = 1 , N
        X( I ) = XZERO
      END DO
*
      CALL ESTRUTURA(IVECT,JVECT)
*
      CALL SNLDIN(X,IVECT,JVECT)
*
      CALL EXIT
      END
```

```
*-----------------------------------------------------------------------
*       ESTRUTURA  P/ BROYDEN - 3
*-----------------------------------------------------------------------
*
      SUBROUTINE ESTRUTURA(IVECT,JVECT)
*
      COMMON / INFORMACOES / N,NEL
      COMMON / VETICN      / ICNO(5000)
      Common / Faixa       / Lf
      Common / Semente     / Isem
*
      INTEGER * 4 IVECT(NEL),JVECT(NEL)
*
*     Linha 1
      IVECT( 1 ) = 1
      JVECT( 1 ) = 1
      IVECT( 2 ) = 1
      JVECT( 2 ) = 2
      Linf       = 3
      Lsup       = 1 + Lf
      Ale        = Ran( Isem )
      Icol       = Ifix( Ale*( Lsup-Linf+ 1 ) ) + Linf
      IVECT( 3 ) = 1
      JVECT( 3 ) = Icol
      Ind        = 4
*
      Do I = 2 , ( N-1 )
            DO L = IND , (IND + 3)
               IVECT( L ) = I
            END DO
*           Gera a coluna extra
            Linf = Max( 1 , ( I - LF ))
            Lsup = Min( N , ( I + LF ))
            Mult = Lsup - Linf + 1
            Icol = I
            Do While( Icol .Ge. (i-1) .and. Icol .Le. (i+1))
                  Ale  = Ran( Isem )
                  Icol = Ifix( Ale*Mult ) + Linf
            End do
            If( Icol .Lt. ( i-1 )) Then
                  JVECT( ind )     = icol
                  JVECT( ind+1 )   = i - 1
                  JVECT( ind+2 )   = i
                  JVECT( ind+3 )   = i + 1
                                     Else
                  JVECT( ind )     = i - 1
                  JVECT( ind+1 )   = i
                  JVECT( ind+2 )   = i + 1
                  JVECT( ind+3 )   = Icol
            End if
            Ind  =   Ind + 4
      End do
*
```

```
*       -----
*
        DO _ = IND , (IND + 2)
           JVECT( _ ) = _
        END DO
        Linf       = Max( 1 , ( n-Lr ))
        Lsup       = N - 2
        Mult       = Lsup - Linf + 1
        Ale        = Ran( Isem )
        Icol       = Ifix( Ale*Mult ) + Linf
        JVECT( Ind ) = Icol
        JVECT( Ind+1 ) = N - 1
        JVECT( Ind+2 ) = N
        NEL            = Ind + 2
        DO I = 1 , NEL
           ICNO( I ) = JVECT(I)
        END DO
*
        Return
        End
*
*
*-----------------------------------------------------------------------
*    FUNCAO PARA SNL BROYDEN - 3
*-----------------------------------------------------------------------
*
      Subroutine FuncAO( X,F )
*
        COMMON / INFORMACOES / N
        COMMON / VETICN      / ICNO( 5000 )
        Common / Param       / PQ , PE
*
        Integer * 4 Apt
        Real    * 4 X( N ) , F( N )
*
*       Funcao 1
*
        Ind = ICNO(3)
        F( 1 ) = X(1)*(-PQ*X(1)+3.0)-2.0*X(2)+PE*X(ind)+1.0
*
*       Funcoes 2 ate ( n-1 )
*
        APT = 4
        Do I = 2 , ( n-1 )
          Ind = ICNO( Apt )
          If( Ind .Eq. ( i-1 ) ) Ind = ICNO( Apt + 3 )
*
          F( I ) = X(i)*(-PQ*X(i)+3.0)-X(i-1)-2.0*X(i+1)+PE*X(ind)+1.0
*
          APT = APT + 4
        End do
*
*       Funcao N
```

```
*
        ind = ICNO( Apt )
        F( N ) = X(n)*(-PQ*X(n)+3.0)-X(n-1)+P_*X(ind)+1.0
*
*
        Return
        End
*
*
*-----------------------------------------------------------------------
*    JACOBIANO PARA BROYDEN - 3
*-----------------------------------------------------------------------
*
        SUBROUTINE JACOBIANO( X,A )
*
        COMMON / INFORMACOES / N,NEL
        COMMON / VETICN      / ICNO(5000)
        Common / PARAM       / PQ,PE
        Common / Faixa       / Lf
        Common / Semente     / Isem
*
        Real     * 4  X( N ) , A( NEL )
        INTEGER * 4  APT
*
*       DERIVADAS PARCIAIS DA 1A. EQUACAO
*
        A( 1 ) = -2.0*PQ*X(1) + 3.0
        A( 2 )   = -2.0
        A( 3 ) = PE
        Ind        = 4
*
*       DERIVADAS PARCIAIS DAS EQUACOES 1 A ( N-1 )
*
        Do I = 2 , ( N-1 )
               ICOL = ICNO( IND )
               IF( ICOL .LT. ( I-1 ) ) THEN
                        A( ind )      = PE
                        A( ind + 1 ) =  -1.0
                        A( ind + 2 ) =  -2.0*PQ*X(i) + 3.0
                        A( ind + 3 ) =  -2.0
                                        Else
                        A( ind )      =  -1.0
                        A( ind+1 )    =  -2.0*PQ*X(i)+3.0
                        A( ind+2 )    =  -2.0
                        A( ind+3 )    =  PE
                End if
                Ind   =   Ind + 4
        End do
*
*
*       DERIVADAS PARCIAIS DA EQUACAO N
*
```

```
      A( IND   )      =   1.
      A( IND+1 )      =  -1.0
      A( IND + 2)     =  -2.0*PO*X(N)+3.0
*
      NEL = IND + 2
*
      Return
      End
```

```
C-----------------------------------------------------------------------
C       SUBROTINA       S N L D I N
C-----------------------------------------------------------------------
C
C       Diretivas de opcao de metocos e forma de execucao
C       Chamada da rotina Mestra
C       Saida dos resutados
C-----------------------------------------------------------------------
C
        SUBROUTINE SNLDIN(X,IVECT,JVECT)
C
C
C
        COMMON / INFORMACOES /  N,NEL,NELDECOMP,U
        COMMON / TOLERANCIAS /  DISTX,FMAX,FTOL,ITMAX,XTOL
        COMMON / CONV        /  INDICADOR
        COMMON / NORMAS      /  XNORMA,FNORMA
        COMMON / OPCOES      /  OPT
        COMMON / NMET        /  NMET1,NMET2
        COMMON / NTESTE      /  NTES1,NTES2
C
        INTEGER  * 4    IVECT(NEL),JVECT(NEL)
        CHARACTER * 20 NMET2,NTES1,NTES2
        CHARACTER * 30 NMET1,IOPSEQ
C
C       Diretivas para selecao dos metodos
C
        TYPE 50
50      FORMAT(//,T15,' METODOS QUASE NEWTON IMPLEMENTADOS : ')
        TYPE 55
55      FORMAT(T15,37('-') ,/ )
700     TYPE 60
60      FORMAT( T10,'( 0 ) METODO DE NEWTON ',/,
     1          T10,'( 1 ) METODO DE NEWTON MODIFICADO ',/,
     2          T10,'( 2 ) METODO DE DENNIS MARWIL ',/,
     3          T10,'( 3 ) METODO DE SCHUBERT ',/,
     5          T10,'( 4 ) METODO DE ESCALAMENTO DE COLUNAS ',/,
     6          T10,'( 5 ) METODO DE ESCALAMENTO DE LINHAS ',/,
     7       T10,'( 6 ) METODO DE ATUALIZACAO DO FATOR DIAGONAL',/,
     8          T10,'( 7 ) METODO DE BROYDEN ',/,
     9          T10,'( 8 ) METODO DE MARTINEZ ',///,
     9          T10,' NUMERO DO METODO ESCOLHIDO ?   ',$ )
        ACCEPT * , NME
        IF ( NME .GT. 8 ) GO TO 700
C
        IF (( NME .EQ. 0) .OR. ( NME .EQ. 3 )) THEN
        TYPE  717
C
C       Diretivas de forma de execucao
C
717     FORMAT(T15,' BRF DIGITE -> 0 ',/,
     1         T15,' BSF DIGITE -> 1 ',/,
     2         T15,' OPCAO ESCOLHIDA ?  ',$ )
```

```
C
      IF ( NME .EQ. 0 ) THEN
       NMET1 = '        NEWTON '
       NMET2 = ' NEWTON'
      ELSE IF( NME .EQ. 1 ) THEN
       NMET1 = '     NEWTON MODIFICADO '
       NMET2 = ' NEW MODIF '
      ELSE IF( NME .EQ. 2 ) THEN
       NMET1 = '          DENNIS MARWIL '
       NMET2 = ' DENNIS MARWIL '
      ELSE IF( NME .EQ. 3 ) THEN
       NMET1 = '              SCHUBERT '
       NMET2 = ' SCHUBERT '
      ELSE IF( NME .EQ. 4 ) THEN
       NMET1 = '      ESCALAMENTO DE COLUNAS '
       NMET2 = ' COLUNA '
      ELSE IF( NME .EQ. 5 ) THEN
       NMET1 = '      ESCALAMENTO DE LINHAS '
       NMET2 = ' LINHA '
      ELSE IF( NME .EQ. 6 ) THEN
       NMET1 = 'ATUALIZACAO DO FATOR DIAGONAL '
       NMET2 = ' DIAGONAL '
      ELSE IF( NME .EQ. 7 ) THEN
       NMET1 = '              BROYDEN '
       NMET2 = ' BROYDEN '
      ELSE IF( NME .EQ. 8 ) THEN
       NMET1 = '              MARTINEZ '
       NMET2 = ' MARTINEZ '
      ELSE
       NMET1 = ' '
       NMET2 = ' '
      END IF
C
      IF ( OPT .EQ. 1 ) THEN
       IOPSEQ = 'MESMA SEQUENCIA DE PIVOS : BSF'
      ELSE
       IOPSEQ = 'DIFERENTE SEQ. DE PIVOS : BRF'
      END IF
C
C     Chamada da rotina MESTRA : Monitora execucao
C
      CALL MESTRA(NME,X,IVECT,JVECT,TEMPTOT,TNEW,ITER)
C
      IF (( INDICADOR .EQ. 7 ) .OR. ( INDICADOR .EQ. 3 )) RETURN
C
C     Saida dos resultados
C
      TYPE 114
114   FORMAT(//,'  *******     RESULTADOS  FINAIS     ******* ')
      WRITE(*,197) NTES1
197   FORMAT(//,' PROBLEMA : ',A20 )
```

```
      WRITE(*,224) NMET2
224   FORMAT(/,' METODO :   ',A20 )
      IF (( NME .EQ. 0 ) .OR. ( NME .EQ. 3 )) THEN
       WRITE(*,225) IOPSEQ
225    FORMAT(/,' UTILIZANDO ',A30 )
      END IF
      WRITE(*,919) N
919   FORMAT(/,' DIMENSAO : ',I6)
      WRITE(*,929) U
929   FORMAT(/,' PAR. TOLERANCIA : ',E16.8)
      IF (INDICADOR .EQ. 3) THEN
         TYPE 104
104      FORMAT(//,' ***   ESTOURO NO VALOR DA FUNCAO   *** ')
         RETURN
      END IF
C
      IF (INDICADOR .EQ. 2) TYPE 105
105   FORMAT(/,'  ****   CONVERGENCIA NA NORMA DA  FUNCAO   **** ')
C
      IF (INDICADOR .EQ. 1) TYPE 106
106   FORMAT(/,'  ****   CONVERGENCIA NA NORMA DE  DELTAX   **** ')
C
      WRITE(*,201) FNORMA
201   FORMAT(/,'  NORMA   DA   FUNCAO      ---->  ',E16.8)
      WRITE(*,202) XNORMA
202   FORMAT(/,'  NORMA   DE   DELTAX      ---->  ',E16.8)
      WRITE(*,110) ITER
110   FORMAT(/,'  NUMERO  DE  ITERACOES    ---->  ',I4)
      TEMPQN = (TEMPTOT - TNEW)/(ITER - 1)
      WRITE(*,153) TNEW
153   FORMAT(/,'  TEMPO PRIM. IT. NEWTON   ---->  ',E16.8)
      WRITE(*,107) TEMPQN
107   FORMAT(/,'  TEMPO IT. QUASE-NEWTON   ---->  ',E16.8)
      WRITE(*,108) TEMPTOT
108   FORMAT(/,'  TEMPO     TOTAL          ---->  ',E16.8)
C
      RETURN
      END
C
C
C -------------------------------------------------------------
C     ROTINA      M E S T R A
C-------------------------------------------------------------
C
C     Monitoramento de execucao :
c     Diretivas Newton - Quase-Newton
C     Definicao de dimensoes
C     Definicao dos parametros de MA-28
C-------------------------------------------------------------
C
      SUBROUTINE MESTRA(NME,X,IVECT,JVECT,TEMPTOT,TNEW,ITER)
C
C
```

```
      COMMON / INFORMACOES /   N,NEL,NELBLOCOS,U
      COMMON / TOLERANCIAS /   DISTX,FMAX,FTOL,STMAX,XTOL
      COMMON / CONV        /   INDICADOR
      COMMON / NORMAS      /   XNORMA,FNORMA
      COMMON / OPCOES      /   OPT
C
C     Fixando dimensoes maximas
C
      INTEGER  IVECT(NEL),JVECT(NEL),ICN(120000),ILPTR(6000),
     1         IUPTR(6000),IKEEP(30000),IDISP(10),PMTL(6000),
     2         PMTC(6000),IRN(80000)
      LOGICAL    GROW,LBLOCK,ABORT(4),MNEWTON
      REAL   * 4  X(N),RHS(6000),A(120000),B(6000),DELTAX(6000),
     1  D(6000),FX(6000),AAUX(30000),VBROY(50000),VCOL(30000)
C
C     Definicao dos parametros de MA-28
C
      NZ    =  NEL
      LIRN  =  80000
      LICN  =  120000
      NA    =  N
      UA    =  U
C
      LBLOCK     = .FALSE.
      GROW       = .FALSE.
      ABORT(1)   = .TRUE.
      ABORT(2)   = .TRUE.
      ABORT(3)   = .FALSE.
      ABORT(4)   = .TRUE.
      IFAIL      = 110
      MTYPE      = 1
      ETA        = 1.0E-04
C
      MNEWTON = .TRUE.
      ITER =0
      INDICADOR = 0
      TEMPTOT   = 0
      TEMPTOTFAT = 0
C
      CALL FUNCAO(X,RHS)
C
      CALL NORMA(RHS)
C
      WRITE(*,11) ITER
11    FORMAT(/,'  ITERACAO ---->   ',I4)
C
      WRITE(*,98) FNORMA
98    FORMAT(/,'  NORMA DA FUNCAO   ---->   ',E16.8)
C
      IF (INDICADOR .EQ. 2)  THEN
         INDICADOR = 7
         TYPE 101
```

```
101         FORMAT(///,' ***    CHUTE INICIAL = SOLUCAO    *** ')
            RETURN
          END IF
C
          IF (INDICADOR .EQ. 3) THEN
            TYPE 102
102         FORMAT(///,'***    ESTOURO NO VALOR DA FUNCAO EM X0     *** ')
            RETURN
          END IF
C
C
777       ITER = ITER + 1
          WRITE(*,111) ITER
111       FORMAT(/,'  ITERACAO --->  ',I4)
C
          IF ( ( ITER .EQ. 1 ) .OR. ( NME .EQ. 0 ) ) THEN

           CALL  NEWTON(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,LICN,MTYPE,
     1                  IFAIL,GROW,LBLOCK,ABORT,UA,ETA,ITER,IVECT,
     2                  JVECT,A,ICN,IRN,IKEEP,IDISP,NME,
     3       ILPTR,IUPTR,PMTL,PMTC,B,DELTAX,D,FX,AAUX,VBROY,VCOL)
C
           IF ( ITER .EQ. 1 ) THEN
             TNEW = TEMPO
           END IF
C           WRITE(*,194) TEMPFAT
C 194       FORMAT(//,' TEMPO DA FATORACAO ---> ',E16.8)
C
          ELSE
C
           CALL QUASENEWTON(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,LICN,
     1               MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,ETA,ITER,
     2       IVECT,JVECT,A,ICN,IRN,IKEEP,IDISP,NME,ILPTR,IUPTR,
     3   PMTL,PMTC,B,DELTAX,D,FX,AAUX,MNEWTON,VBROY,VCOL)
          END IF
C
          WRITE(*,198) FNORMA
198       FORMAT(/,'  NORMA DA FUNCAO    --->  ',E16.8)
          WRITE(*,199) XNORMA
199       FORMAT(/,'  NORMA DE DELTAX    --->  ',E16.8)
          WRITE(*,189) TEMPO
189       FORMAT(/,'  T E M P O          --->  ',E16.8)
C
          TEMPTOT     = TEMPTOT + TEMPO
C
C
          IF ( ITER .GT. ITMAX ) THEN
            TYPE 103
103       FORMAT(///,'*** EXCEDEU O NUMERO MAXIMO DE ITERACOES *** ')
            RETURN
          END IF
C
          IF (INDICADOR .EQ. 0) GO TO 777
```

```
C
C       ******  FINAL DA ROTINA MESTRA ******
C
        RETURN
        END
C
C
C-------------------------------------------------------------
C       SUBROTINA NEWTON
C-------------------------------------------------------------
C
C       Diretivas de instrucao para primeira iteracao
C       Newton, segundo o Quase-Newton escolhido
C-------------------------------------------------------------

         SUBROUTINE NEWTON(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,LICN,
     1                     MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,ETA,
     2                  ITER,IVECT,JVECT,A,ICN,IRN,IKEEP,IDISP,NME,
     3      ILPTR,IUPTR,PMTL,PMTC,B,DELTAX,D,FX,AAUX,VBROY,VCOL)
C
C
        COMMON / INFORMACOES / N
        COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
        COMMON / CONV        / INDICADOR
        COMMON / NORMAS      / XNORMA,FNORMA
C
C
        LOGICAL    GROW,LBLOCK,ABORT(4)
        REAL   * 4 X(1),RHS(1),W(6000),A(1),B(1),
     1  DELTAX(1),D(1),FX(1),AAUX(1),VBROY(1),VCOL(1)
        INTEGER * 4 IRN(1),ICN(1),IKEEP(1),IW(48000),IDISP(10),
     1    IVECT(1),JVECT(1),ILPTR(1),IUPTR(1),PMTL(1),PMTC(1)
C
C       Chama rotinas Newton
C
        GO TO  (100,200,300,400,500,600,700,800) NME
C
C       NME = 0  ===> METODO DE NEWTON
C
        CALL  NEWTONPURO(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
     1            LICN,MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,
     2       ETA,ITER,IVECT,JVECT,A,ICN,IRN,IKEEP,IDISP,
     3       ILPTR,IUPTR,PMTL,PMTC)
C
        RETURN
C
C       NME = 1  ===> NEWTON P/ NEWTON MODIFICADO
C
100     CALL  NEWTONPURO(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
     1            LICN,MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,
     2     ETA,ITER,IVECT,JVECT,A,ICN,IRN,IKEEP,IDISP,
     3     ILPTR,IUPTR,PMTL,PMTC)
C
```

```
      RETURN
C
C       NME = 2  ===> NEWTON P/ DENNIS MARWIL
C
200   CALL  NEWDM(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,LICN,
     1          IFAIL,GROW,LBLOCK,ABORT,ITER,IVECT,JVECT,
     2  UA,A,ICN,IRN,IKEEP,ILPTR,IUPTR,PMTL,PMTC,B,DELTAX)
C
      RETURN
C
C       NME = 3  ===> NEWTON P/ SCHUBERT
C
300   CALL NEWSCHU(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,LICN,
     1  MTYPE,IFAIL,GROW,LBLOCK,ABORT,ITER,IVECT,JVECT,UA,
     2  A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,PMTL,PMTC,FX,
     3  DELTAX,AAUX)
C
        RETURN
C
C       NME = 4  ===> NEWTON P/ COLUNA
C
400   CALL NEWCOL(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,LICN,
     1   MTYPE,IFAIL,GROW,LBLOCK,ABORT,ITER,IVECT,JVECT,
     2    UA,A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,PMTL,PMTC,
     3    DELTAX,D)
C
        RETURN
C
C       NME = 5  ===> NEWTON P/ LINHA
C
500     CALL NEWLIN(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,LICN,
     1   MTYPE,IFAIL,GROW,LBLOCK,ABORT,ITER,IVECT,JVECT,UA,
     2   A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,PMTL,PMTC,D,FX)
C
          RETURN
C
C       NME = 6  ===> NEWTON P/ DIAGONAL
C
600   CALL NEWDIAG(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
     1       LICN, MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,
     2       ETA,ITER,IVECT,JVECT,A,ICN,IRN,IKEEP,
     3       IDISP,ILPTR,IUPTR,PMTL,PMTC,B )
C
      RETURN
C
C       NME = 7  ===> NEWTON P/ BROYDEN
C
700   CALL NEWBROY(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
     1       LICN, MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,
     2       ETA,ITER,IVECT,JVECT,A,ICN,IRN,IKEEP,
     3       IDISP,ILPTR,IUPTR,PMTL,PMTC,VBROY)
C
      RETURN
```

```
C
C       NME = 8  ===> NEWTON P/ ATZCOL
C
800     CALL NEWMARTINEZ(X,RHS,TEMPO,TEMPFAT,NA,NZ,
     1  LIRN,LICN, MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,
     2       ETA,ITER,IVECT,JVECT,A,ICN,IRN,IKEEP,
     3       IDISP,ILPTR,IUPTR,PMTL,PMTC,VCOL)
C
C
C        ******* FINAL DA ROTINA  NEWTON ******
C
         END
C
C
C-------------------------------------------------------------
C        N E W T O N  -  P U R O
C-------------------------------------------------------------
C
C
      SUBROUTINE NEWTONPURO(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
     1   LICN,MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,ETA,ITER,IVECT,
     2   JVECT,A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,PMTL,PMTC)
C
C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
      COMMON / OPCOES      / OPT
C
C
      LOGICAL   GROW,LBLOCK,ABORT(4)
      REAL      * 4 X(1),RHS(1),W(6000),A(1)
      INTEGER   * 4 IRN(1),ICN(1),IKEEP(1),
     1     IW(48000),IDISP(10),IVECT(1),JVECT(1),PMTC(1),
     2     PMTL(1),IUPTR(1),ILPTR(1)
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      CALL LIB$INIT_TIMER(INICIO)
C
      CALL JACOBIANO(X,A)
C
C
      IF (( ITER .EQ. 1 ) .OR. ( OPT .EQ. 0 )) THEN
C
        DO I = 1 , NZ
           IRN( I ) = IVECT( I )
           ICN( I ) = JVECT( I )
```

```
            END DO
C
             CALL F01BRF(NA,NZ,A,LICN,IRN,LIRN,ICN,UA,IKEEP,IW,W,
     1           LBLOCK,GROW,ABORT,IDISP,IFAIL)
C
           ELSE
           IFAIL = 110
           ETA = 1.0E-01
C
            CALL F01BSF(NA,NZ,A,LICN,IVECT,JVECT,ICN,IKEEP,IW,W,
     1                  GROW,ETA,RPMIN,ABORT(4),IDISP,IFAIL)
C
         END IF
C
         CALL PRESOLVE(IKEEP,ILPTR,IUPTR,PMTL,PMTC)
C
         CALL SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,RHS)
C
         CALL CALCX(RHS,X,PMTC)
C
         CALL FUNCAO(X,RHS)
C
         CALL NORMA(RHS)
C
         CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
         TEMPO = FLOAT(IFIM)
C
C         ******** FINAL NEWTON PURO ******
C
         RETURN
         END
C
C-------------------------------------------------------------
C       N E W T O N   -    D E N N I S   M A R W I L
C-------------------------------------------------------------
C
        SUBROUTINE NEWDM(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
     1     LICN,IFAIL,GROW,LBLOCK,ABORT,ITER,IVECT,JVECT,
     2     UA,A,ICN,IRN,IKEEP,ILPTR,IUPTR,PMTL,PMTC,
     3     B,DELTAX)
C
C
         COMMON / INFORMACOES / N
         COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
         COMMON / CONV        / INDICADOR
         COMMON / NORMAS      / XNORMA,FNORMA
C
C
         LOGICAL   GROW,LBLOCK,ABORT(4)
         REAL * 4 X(1),RHS(1),W(1),A(1),B(1),DELTAX(1)
         INTEGER  * 4 IRN(1),ICN(1),IKEEP(1),
     1   IW(48000),IDISP(10),IUPTR(1),ILPTR(1),PMTL(1),PMTC(1),
```

```
     2   IVECT(1),JVECT(1)
C
C
          INICIO = 0
          IFIM   = 0
          TEMPO  = 0
C
          CALL LIB$INIT_TIMER(INICIO)
C
          CALL JACOBIANO(X,A)
C
            DO I = 1 , NZ
               IRN( I ) = IVECT( I )
               ICN( I ) = JVECT( I )
            END DO
C
             CALL F01BRF(NA,NZ,A,LICN,IRN,LIRN,ICN,UA,IKEEP,IW,W,
     1           LBLOCK,GROW,ABORT,IDISP,IFAIL)
C
          CALL LIB$STAT_TIMER(2,IFFAT,INIFAT)
C
          TEMPFAT = FLOAT(IFFAT)
C
          CALL PRESOLVE(IKEEP,ILPTR,IUPTR,PMTL,PMTC)
C
          CALL  SOLVL(A,ICN,ILPTR,IUPTR,PMTL,RHS)
C
         DO  I = 1 , N
              B( I ) = RHS( I )
         END DO
C
          CALL SOLVU(A,ICN,ILPTR,IUPTR,PMTC,RHS)
C
          CALL CALCX(X,RHS,PMTC)
C
          DO I = 1 , N
             DELTAX( I ) = -RHS( I )
          END DO
C
          CALL FUNCAO(X,RHS)
C
          CALL NORMA(RHS)
C
          CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
          TEMPO = FLOAT(IFIM)
C
C       ******* FINAL NEWTON - DENNIS MARWIL ******
C
          RETURN
          END
C
C
```

```
C-----------------------------------------------------------------------
C       N E W T O N   -   S C H U B E R T
C-----------------------------------------------------------------------
C
        SUBROUTINE NEWSCHU(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
      1 LICN,MTYPE,IFAIL,GROW,LBLOCK,ABORT,ITER,IVECT,JVECT,
      2   UA,A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,PMTL,PMTC,
      3    FX,DELTAX,AAUX)
C
C
        COMMON / INFORMACOES / N
        COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
        COMMON / CONV        / INDICADOR
        COMMON / NORMAS      / XNORMA,FNORMA
C
C
        LOGICAL   GROW,LBLOCK,ABORT(4)
        REAL * 4 X(1),RHS(1),W(1),A(1),FX(1),
      1          DELTAX(1),AAUX(1)
        INTEGER  * 4 IRN(1),ICN(1),IKEEP(1),
      1 IW(48000),IDISP(10),IUPTR(1),ILPTR(1),PMTL(1),PMTC(1),
      2 IVECT(1),JVECT(1)
C
C
        INICIO = 0
        INIFAT = 0
        IFIM   = 0
C
        CALL LIB$INIT_TIMER(INICIO)
C
        CALL JACOBIANO(X,A)
C
        NZA = NZ
C
        DO 10 I = 1 , NZ
              AAUX( I ) = A( I )
10      CONTINUE
C
        DO 13 I = 1 , N
           FX( I ) = RHS( I )
13      CONTINUE
C
          DO I = 1 , NZ
             IRN( I ) = IVECT( I )
             ICN( I ) = JVECT( I )
          END DO
C
           CALL F01BRF(NA,NZ,A,LICN,IRN,LIRN,ICN,UA,IKEEP,IW,W,
      1         LBLOCK,GROW,ABORT,IDISP,IFAIL)
C
        NZ = NZA
C
C
```

```
      CALL PRESOLVE(IKEEP,ILPTR,IUPTR,PMTL,PMTC)
C
      CALL  SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,RHS)
C
      CALL CALCX(RHS,X,PMTC)
C
      DO  I = 1 , N
          DELTAX( I ) = -RHS( I )
      END DO
C
      CALL FUNCAO( X,RHS )
C
      CALL NORMA(RHS)
C
      CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
      TEMPO = FLOAT(IFIM)
C
C     ******* FINAL NEWTON - SCHUBERT ******
C
      RETURN
      END
C
C-------------------------------------------------------
C     N E W T O N   -   C O L U N A
C-------------------------------------------------------
C
      SUBROUTINE NEWCOL(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
     1 LICN,MTYPE,IFAIL,GROW,LBLOCK,ABORT,ITER,IVECT,JVECT,
     2    UA,A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,PMTL,PMTC,
     3    DELTAX,D)
C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
C
C
      LOGICAL   GROW,LBLOCK,ABORT(4)
      REAL * 4 X(1),RHS(1),W(1),A(1),D(1),DELTAX(1)
      INTEGER  * 4 IRN(1),ICN(1),IKEEP(1),
     1  IW(48000),IDISP(10),IUPTR(1),ILPTR(1),PMTL(1),PMTC(1),
     2  IVECT(1),JVECT(1)
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      CALL LIB$INIT_TIMER(INICIO)
C
      CALL JACOBIANO(X,A)
```

```
C
            DO I = 1 , NZ
               IRN( I ) = IVECT( I )
               ICN( I ) = JVECT( I )
            END DO
C
C
             CALL F01BRF(NA,NZ,A,LICN,IRN,LIRN,ICN,UA,IKEEP,IW,W,
     1           LBLOCK,GROW,ABORT,IDISP,IFAIL)
C
C
          CALL PRESOLVE(IKEEP,ILPTR,IUPTR,PMTL,PMTC)
C
          CALL  SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,RHS)
C
          CALL CALCX(RHS,X,PMTC)
C
          DO I = 1 , N
             DELTAX( I ) = -RHS( I )
          END DO
C
          CALL FUNCAO(X,RHS)
C
          CALL NORMA(RHS)
C
          DO I = 1 , N
             D( I ) = 1.0
          END DO
C
          CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
          TEMPO = FLOAT(IFIM)
C
C       ******* FINAL NEWTON - COLUNA ******
C
          RETURN
          END
C
C
C-------------------------------------------------------------
C       N E W T O N   -   L I N H A
C-------------------------------------------------------------
C
        SUBROUTINE NEWLIN(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
     1  LICN,MTYPE,IFAIL,GROW,LBLOCK,ABORT,ITER,IVECT,JVECT,
     2  UA,A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,PMTL,PMTC,D,FX)
C
C
          COMMON / INFORMACOES / N
          COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
          COMMON / CONV        / INDICADOR
          COMMON / NORMAS      / XNORMA,FNORMA
C
```

```
C
      LOGICAL   GROW,LBLOCK,ABORT(4)
      REAL * 4 X(1),RHS(1),W(1),A(1),FX(1),D(1)
      INTEGER  * 4 IRN(1),ICN(1),IKEEP(1),
     1  IW(48000),IDISP(10),IUPTR(1),ILPTR(1),PMTL(1),PMTC(1),
     2  IVECT(1),JVECT(1)
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      CALL JACOBIANO(X,A)
C
        DO I = 1 , NZ
           IRN( I ) = IVECT( I )
           ICN( I ) = JVECT( I )
        END DO
C
         CALL F01BRF(NA,NZ,A,LICN,IRN,LIRN,ICN,UA,IKEEP,IW,W,
     1          LBLOCK,GROW,ABORT,IDISP,IFAIL)
C
C
      CALL PRESOLVE(IKEEP,ILPTR,IUPTR,PMTL,PMTC)
C
      DO I = 1 , N
         FX( I ) = RHS( I )
      END DO
C
      CALL PRESOLVE(IKEEP,ILPTR,IUPTR,PMTL,PMTC)
C
      CALL  SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,RHS)
C
      CALL CALCX(RHS,X,PMTC)
C
      CALL FUNCAO(X,RHS)
C
      CALL NORMA(RHS)
C
      CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
      TEMPO = FLOAT(IFIM)
C
      DO 20 I = 1 , N
         D( I ) = 1.0
20    CONTINUE
C
C     ******* FINAL NEWTON - LINHA ******
C
      RETURN
      END
C
C
```

```
C-----------------------------------------------------------------------
C          N E W T O N  -  D I A G O N A L
C-----------------------------------------------------------------------
C

      SUBROUTINE NEWDIAG(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
     1     LICN, MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,ETA,ITER,
     2     IVECT,JVECT,A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,
     3     PMTL,PMTC,B )

C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
C
C
      LOGICAL    GROW,LBLOCK,ABORT(4)
      REAL       * 4 X(N),RHS(1),W(1),A(1),B(1)
      INTEGER    * 4 IRN(1),ICN(1),IKEEP(1),
     1     IW(48000),IDISP(10),IVECT(1),JVECT(1)
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      CALL LIB$INIT_TIMER(INICIO)
C
      CALL JACOBIANO(X,A)
C
      DO I = 1 , NZ
           IRN( I ) = IVECT( I )
           ICN( I ) = JVECT( I )
      END DO
C
         CALL F01BRF(NA,NZ,A,LICN,IRN,LIRN,ICN,UA,IKEEP,IW,W,
     1        LBLOCK,GROW,ABORT,IDISP,IFAIL)
C
C
      CALL PRESOLVE(IKEEP,ILPTR,IUPTR,PMTL,PMTC)
C
      CALL SOLVL(A,ICN,ILPTR,IUPTR,PMTL,RHS)
C
      DO 10 I = 1 , N
         B( I ) = RHS( I )
10    CONTINUE
C
      CALL SOLVU(A,ICN,ILPTR,IUPTR,PMTC,RHS)
C
      CALL CALCX(RHS,X,PMTC)
C
```

```
      CALL FUNCAO(X,RHS)
C
      CALL NORMA(RHS)
C
      CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
      TEMPO = FLOAT(IFIM)
C
C      ******* FINAL NEWTON DIAGONAL  ******
C
      RETURN
      END
C
C
C-----------------------------------------------------------
C       N E W T O N  -  B R O Y D E N
C-----------------------------------------------------------
C
C
       SUBROUTINE NEWBROY(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
     1        LICN, MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,ETA,ITER,
     2        IVECT,JVECT,A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,
     3        PMTL,PMTC,VBROY)

C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
      COMMON / CONTBROY    / KBROY,ITBROY,KBROYNS
C
C
      LOGICAL   GROW,LBLOCK,ABORT(4)
      REAL * 4 X(1),RHS(1),W(1),A(1),VBROY(1)
      INTEGER   * 4 IRN(1),ICN(1),IKEEP(1),
     1     IW(48000),IDISP(10),IVECT(1),JVECT(1)
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      CALL LIB$INIT_TIMER(INICIO)
C
      CALL JACOBIANO(X,A)
C
      DO I = 1 , NZ
           IRN( I ) = IVECT( I )
           ICN( I ) = JVECT( I )
      END DO
C
         CALL F01BRF(NA,NZ,A,LICN,IRN,LIRN,ICN,UA,IKEEP,IW,W,
```

```
     1          LBLOCK,GROW,ABORT,IDISP,IFAIL)
C
      CALL PRESOLVE(IKEEP,ILPTR,IUPTR,PMTL,PMTC)
C
      CALL SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,RHS)
C
      CALL CALCX(RHS,X,PMTC)
C
      DO I = 1 , N
         VBROY( I ) = - RHS( I )
      END DO
C
      KBROY    = 0
      KBROYNS  = 0
C
      CALL FUNCAO(X,RHS)
C
      CALL NORMA(RHS)
C
      CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
      TEMPO = FLOAT(IFIM)
C
C      ******* FINAL NEWTON BROYDEN  ******
C
      RETURN
      END
C
C
C-----------------------------------------------------------------
C       N E W T O N  -  M A R T I N E Z
C-----------------------------------------------------------------
C
C
       SUBROUTINE NEWMARTINEZ(X,RHS,TEMPO,TEMPFAT,NA,NZ,
     1   LIRN,LICN, MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,ETA,ITER,
     2       IVECT,JVECT,A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,
     3       PMTL,PMTC,VCOL)

C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
      COMMON / CONTCOL     / KATCOL,ITCOL,KATCOLNS
      COMMON / INDATZCOL   / INDCOL
C
C
      LOGICAL   GROW,LBLOCK,ABORT(4)
      REAL * 4 X(1),RHS(1),W(1),A(1),VCOL(1)
      INTEGER   * 4 IRN(1),ICN(1),IKEEP(1),
     1     IW(48000),IDISP(10),IVECT(1),JVECT(1)
```

```
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      CALL LIB$INIT_TIMER(INICIO)
C
      CALL JACOBIANO(X,A)
C
      DO I = 1 , NZ
           IRN( I ) = IVECT( I )
           ICN( I ) = JVECT( I )
      END DO
C
      CALL F01BRF(NA,NZ,A,LICN,IRN,LIRN,ICN,UA,IKEEP,IW,W,
     1          LBLOCK,GROW,ABORT,IDISP,IFAIL)
C
      CALL PRESOLVE(IKEEP,ILPTR,IUPTR,PMTL,PMTC)
C
      CALL SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,RHS)
C
      CALL CALCX(RHS,X,PMTC)
C
      VCOL(1) = FLOAT(INDCOL)
C
      DO I = 1 , N
         VCOL( I + 1 ) = - RHS( I )
      END DO
C
      KATCOL   = 0
      KATCOLNS = 0
C
      CALL FUNCAO(X,RHS)
C
      CALL NORMA(RHS)
C
      CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
      TEMPO = FLOAT(IFIM)
C
C      ******** FINAL NEWTON MARTINEZ  ******
C
      RETURN
      END
C
C
C-----------------------------------------------------------------
C           Q U A S E  --  N E W T O N
C-----------------------------------------------------------------
C
C
      SUBROUTINE QUASENEWTON(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
```

```
     1                  LICN,MTYPE,IFAIL,GROW,LBLOCK,ABORT,UA,ETA,
     2    ITER,IVECT,JVECT,A,ICN,IRN,IKEEP,IDISP,NME,ILPTR,IUPTR,
     3    PMTL,PMTC,B,DELTAX,D,FX,AAUX,MNEWTON,VBROY,VCOL)
C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
C
C
      LOGICAL    GROW,LBLOCK,ABORT(4),MNEWTON
      REAL       * 4 X(1),RHS(1),W(60000),A(1),B(1),DELTAX(1),
     1             D(1),FX(1),AAUX(1),VBROY(1),VCOL(1)
      INTEGER    * 4 IRN(1),ICN(1),IKEEP(1),
     1     IW(48000),IDISP(10),IVECT(1),JVECT(1),
     2     ILPTR(1),IUPTR(1),PMTL(1),PMTC(1)
C
C     CHAMA ROTINAS QUASE-NEWTON
C
      GO TO ( 100,200,300,400,500,600,700,800 ) NME
C
C
C     NME = 1  ===>  NEWTON MODIFICADO
C
C
100   CALL NEWMOD(X,RHS,TEMPO,A,ICN,ILPTR,IUPTR,
     1     PMTL,PMTC,NA,LICN,MTYPE,IKEEP,IDISP)
C
      RETURN
C
C     NME = 2  ===>  DENNIS MARWIL
C
200   CALL DENNISMARWIL(X,RHS,TEMPO,A,ICN,ILPTR,
     1              IUPTR,PMTL,PMTC,B,DELTAX)
C
      RETURN
C
C     NME = 3  ===>  SCHUBERT
C
C
300   CALL SCHUBERT(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,LICN,
     1 MTYPE,IFAIL,GROW,LBLOCK,ABORT,ITER,IVECT,JVECT,UA,
     2 ETA,A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,PMTL,PMTC,
     3 FX,DELTAX,AAUX)
C
      RETURN
C
C     NME = 4  ===>  COLUNA
C
400   CALL COLUNA(X,RHS,TEMPO,A,ICN,ILPTR,IUPTR,PMTL,PMTC,
     1    DELTAX,D,NA,LICN,MTYPE,IKEEP,IDISP)
C
```

```
      RETURN
C
C     NME = 5   ===>  LINHA
C
500   CALL LINHA(X,RHS,TEMPO,A,ICN,ILPTR,IUPTR,PMTL,PMTC,
     1    FX,D,NA,LICN,MTYPE,IKEEP,IDISP)
C
      RETURN
C
C     NME = 6   ===>  DIAGONAL
C
600   CALL DIAGONAL(X,RHS,TEMPO,NA,A,ICN,IKEEP,ILPTR,IUPTR,
     1                PMTL,PMTC,B,MNEWTON)
C
      RETURN
C
C     NME = 7   ===>  BROYDEN
C
700   CALL BROYDEN(X,RHS,TEMPO,NA,A,ICN,ILPTR,IUPTR,
     1                PMTL,PMTC,VBROY)
C
      RETURN
C
C     NME = 8   ===>  MARTINEZ
C
800   CALL MARTINEZ(X,RHS,TEMPO,NA,A,ICN,ILPTR,IUPTR,
     1                PMTL,PMTC,VCOL)
C
      RETURN
C
C
C      ******* FINAL DA ROTINA  QUASE-NEWTON ******
C
      END
C
C
C-----------------------------------------------------------
C      N E W T O N  -  M O D I F I C A D O
C-----------------------------------------------------------
C
      SUBROUTINE NEWMOD(X,RHS,TEMPO,A,ICN,ILPTR,IUPTR,
     1          PMTL,PMTC,NA,LICN,MTYPE,IKEEP,IDISP)
C
C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
C
C
```

```
      REAL       * 4 X(1),RHS(1),W(6000),A(1)
      INTEGER    * 4 ICN(1),IKEEP(1),IDISP(10),PMTC(1)
      INTEGER    * 4 ILPTR(1),IUPTR(1),PMTL(1)
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      CALL LIB$INIT_TIMER(INICIO)
C
C
      CALL SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,RHS)
C
      CALL CALCX(RHS,X,PMTC)
C
      CALL FUNCAO(X,RHS)
C
      CALL NORMA(RHS)
C
      CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
      TEMPO = FLOAT(IFIM)
C
C      ******* FINAL NEWTON MODIFICADO ******
C
      RETURN
      END
C
C
C-----------------------------------------------------------
C           D E N N I S   M A R W I L
C-----------------------------------------------------------
C
C
      SUBROUTINE DENNISMARWIL(X,RHS,TEMPO,A,ICN,ILPTR,
     1                IUPTR,PMTL,PMTC,B,DELTAX)
C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
      COMMON / STEPCONTROL / FAT
C
C
      REAL * 4 X(1),RHS(1),A(1),B(1),DELTAX(1),W(6000)
      INTEGER  * 4 ICN(1),IUPTR(1),ILPTR(1),
     1                PMTL(1),PMTC(1)
C
C
      INICIO = 0
      IFIM   = 0
```

```
      TEMPO  = 0
C
      CALL LIB$INIT_TIMER(INICIO)
C
      BETA = FAT - 1.0
C
      CALL  SOLVL(A,ICN,ILPTR,IUPTR,PMTL,RHS)
C
      PREC = 1.0E-04 * XNORMA
C
C
      DO  10 I = 1 , N
          VALBL = BETA*B( I )
          VALB  = RHS( I ) + VALBL
          IBEG  = IUPTR(I)
          IEND  = ILPTR(I+1) - 1
          IF ( I .EQ. N ) IEND = IUPTR(N)
          SOMA  = 0.0
          DO 20 K = IBEG , IEND
                IF( A(K) .NE. 0.0 ) THEN
                    IND = ICN(K)
                    SOMA = SOMA + DELTAX(IND)*DELTAX(IND)
                END IF
20        CONTINUE
          Z = SQRT(SOMA)
          IF ( Z .GT. PREC ) THEN
            K   = IBEG
            IND = ICN(K)
            A(K) = A(K) + ( DELTAX(IND)*VALB )/SOMA
           IF ( ABS(A(K)) .LT. 1.E-07 ) CALL FATSING(A(K))
           DO 30 K = ( IBEG + 1 ) , IEND
              IF ( A(K) .NE. 0.0 ) THEN
                 IND = ICN(K)
                 A(K) = A(K) + ( DELTAX(IND)*VALB )/SOMA
              END IF
30          CONTINUE
          END IF
C
       B( I ) = RHS( I )
10    CONTINUE
C
      CALL SOLVU(A,ICN,ILPTR,IUPTR,PMTC,RHS)
C
      CALL CALCX(RHS,X,PMTC)
C
      DO I = 1 , N
         DELTAX( I ) = -RHS( I )
      END DO
C
      CALL FUNCAO(X,RHS)
C
      CALL NORMA(RHS)
C
```

```
      CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
      TEMPO = FLOAT(IFIM)
C
C     ******* FINAL   DENNIS MARWIL ******
C
      RETURN
      END
C
C
C-----------------------------------------------------------------------
C              S  C  H  U  B  E  R  T
C-----------------------------------------------------------------------
C
C
      SUBROUTINE SCHUBERT(X,RHS,TEMPO,TEMPFAT,NA,NZ,LIRN,
     1 LICN,MTYPE,IFAIL,GROW,LBLOCK,ABORT,ITER,IVECT,JVECT,
     2 UA,ETA,A,ICN,IRN,IKEEP,IDISP,ILPTR,IUPTR,PMTL,PMTC,
     3 FX,DELTAX,AAUX)
C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
      COMMON / STEPCONTROL / FAT
      COMMON / OPCOES      / OPT
C
      LOGICAL   GROW,LBLOCK,ABORT(4)
      REAL * 4 X(1),RHS(1),W(6000),A(1),FX(1),
     1         DELTAX(1),AAUX(1)
      INTEGER  * 4 IRN(1),ICN(1),IKEEP(1),
     1 IW(48000),IDISP(10),IUPTR(1),ILPTR(1),PMTL(1),PMTC(1),
     2 IVECT(1),JVECT(1)
      INTEGER * 4   FIM
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
      FIM    = 0
C
      CALL LIB$INIT_TIMER(INICIO)
C
      DO I = 1 , N
      W(PMTC(I)) = DELTAX(I)
      END DO
      DO I = 1 , N
      DELTAX(I) = W(I)
      END DO
C
      NZA  = NZ
      ZERO = 0.0
```

```
      BETA = FAT - 1.0
C
      PREC = 1.0E-04 * XNORMA
C
      DO  10 I = 1 , N
C
          VALFX = BETA * FX( I )
C
          VALF = RHS( I ) + VALFX
C
          FIM = FIM + 1
          INIC = FIM
          DO WHILE ( IVECT(FIM) .EQ. I )
             FIM = FIM + 1
          END DO
          FIM = FIM - 1
C
C          INIC = ILPTR( I )
C          FIM  = ILPTR( I + 1 ) - 1
C          IF ( I .EQ. N )  FIM = IUPTR(N)
           Z    = 0.0
C
          IF ( ABS( RHS( I ) ) .LT. FTOL ) THEN
C
             DO  20 K = INIC , FIM
                 A( K ) = AAUX( K )
20           CONTINUE
C
          ELSE
              DO 30 K = INIC , FIM
                    IF ( AAUX( K ) .NE. ZERO ) THEN
                         IND = JVECT( K )
                          Z = Z + DELTAX( IND ) * DELTAX( IND )
C
                    END IF
C
30            CONTINUE
C
              SZ = SQRT( Z )
C
             IF( SZ .GT. PREC ) THEN
C
                DO 40 K = INIC , FIM
C
                 IF ( AAUX( K ) .NE. 0.0 ) THEN
C
                  IND = JVECT( K )
                 A( K ) = AAUX( K ) + ( DELTAX(IND)*VALF ) / Z
                  AAUX( K ) = A( K )
C
                 ELSE
                       A( K )  = 0.0
C
```

```
                    END IF
C
40                CONTINUE
C
                ELSE
                    DO 50 K = INIC , FIM
                        A( K ) = AAUX( K )
50                  CONTINUE
                END IF
C
            END IF
C
            FX( I ) = RHS( I )
C
10        CONTINUE
          IF ( OPT .EQ. 0 ) THEN
           DO I = 1 , NZ
               IRN( I ) = IVECT( I )
               ICN( I ) = JVECT( I )
           END DO
C
           CALL F01BRF(NA,NZ,A,LICN,IRN,LIRN,ICN,UA,IKEEP,IW,W,
     1                 LBLOCK,GROW,ABORT,IDISP,IFAIL)
C
           NZ = NZA
           ELSE
C
          ETA   = 1.0E-01
          IFAIL = 110
          CALL F01BSF(NA,NZ,A,LICN,IVECT,JVECT,ICN,IKEEP,IW,W,
     1                  GROW,ETA,RPMIN,ABORT(4),IDISP,IFAIL)
          ENDIF
C
          CALL PRESOLVE(IKEEP,ILPTR,IUPTR,PMTL,PMTC)
C
          CALL SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,RHS)
C
          CALL CALCX(RHS,X,PMTC)
C
          DO I = 1 , N
             DELTAX( I ) = - RHS( I )
          END DO
C
          CALL FUNCAO(X,RHS)
C
          CALL NORMA(RHS)
C
          CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
          TEMPO = FLOAT(IFIM)
C
C       ******* FINAL   SCHUBERT ******
C
```

```
      RETURN
      END
C
C
C-----------------------------------------------------------------
C                    C O L U N A
C-----------------------------------------------------------------
C
C
      SUBROUTINE COLUNA(X,RHS,TEMPO,A,ICN,ILPTR,IUPTR,
     1   PMTL,PMTC,DELTAX,D,NA,LICN,MTYPE,IKEEP,IDISP)
C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
      COMMON / STEPCONTROL / FAT
C
C
      REAL * 4 X(N),RHS(N),A(1),D(1),DELTAX(1),W(6000)
      INTEGER  * 4 ICN(1),IUPTR(1),ILPTR(1),
     1       IKEEP(1),PMTL(1),PMTC(1),IDISP(2)
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      CALL LIB$INIT_TIMER(INICIO)
C
      CALL  SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,RHS)
C
      ALFA = 1.0E-04
      EPS = ALFA * XNORMA
C
      DO  10 I = 1 , N
C
      IF ( ABS( DELTAX(I) ) .GT. EPS .AND. FAT .GT. ALFA ) THEN
C
            D( I ) = RHS( I ) / DELTAX( I ) + D( I ) / FAT
C
      END IF
C
      IF ( ABS( D(I) ) .LT. 1.E-07 ) CALL FATSING( D(I) )

C10    CONTINUE
C
      DELTAX( I ) = RHS( I ) / D(I)
C
10    CONTINUE
C
      CALL CALCX(DELTAX,X,PMTC)
```

```
C
      DO I = 1 , N
         DELTAX( I ) = -DELTAX( I )
      END DO
C
      CALL FUNCAO(X,RHS)
C
      CALL NORMA(RHS)
C
      CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
      TEMPO = FLOAT(IFIM)
C
C
C     ******* FINAL    COLUNA ******
C
      RETURN
      END
C
C-----------------------------------------------------------------
C                  L  I  N  H  A
C-----------------------------------------------------------------
C
C
      SUBROUTINE LINHA(XK1,FXK1,TEMPO,A,ICN,ILPTR,
     1  IUPTR,PMTL,PMTC,FX,D,NA,LICN,MTYPE,IKEEP,IDISP)
C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
      COMMON / STEPCONTROL / FAT
C
C
      REAL * 4      A(1),D(1),FX(1),XK1(6000),
     1              FXK1(6000),W(6000)
      INTEGER  * 4 ICN(1),IUPTR(1),ILPTR(1),IDISP(10),
     1              PMTL(1),PMTC(1),IKEEP(1)
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      CALL LIB$INIT_TIMER(INICIO)
C
      ALFA = 1.0E-04
      EPS  = ALFA * FNORMA
C
      DO  10 I = 1 , N
C
      BETA = FAT * FX( I )
```

```
C
      IF ( ABS( BETA ) .GT. EPS ) THEN
C
      D( I ) = (( FX(I) - FXK1(I) ) / BETA ) * D(I)
C
      END IF
C
      IF ( ABS( D(I) ) .LT. 1.E-07 ) CALL FATSING( D(I) )
C
10    CONTINUE
C
      DO I = 1 , N
         FX( I ) = FXK1( I )
      END DO
      DO I = 1,N
          FXK1( I ) = FXK1( I ) / D(I)
      END DO
C
      CALL SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,FXK1)
C
C
      CALL CALCX( FXK1,XK1,PMTC )
C
      CALL FUNCAO( XK1,FXK1 )
C
      CALL NORMA(FXK1)
C
      CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
      TEMPO = FLOAT(IFIM)
C
C     ******* FINAL   LINHA ******
C
      RETURN
      END
C
C
C-----------------------------------------------------------------
C                D I A G O N A L
C-----------------------------------------------------------------
C
      SUBROUTINE DIAGONAL(X,RHS,TEMPO,NA,A,ICN,IKEEP,
     1                    ILPTR,IUPTR,PMTL,PMTC,B,MNEWTON)
C
C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
      COMMON / STEPCONTROL / FAT
C
```

```
C
      REAL       * 4 X(1),RHS(1),W(6000),A(1),B(1)
      INTEGER    * 4 ICN(1),IKEEP(1),IDISP(10),PMTC(1)
      INTEGER    * 4 ILPTR(1),IUPTR(1),PMTL(1)
      INTEGER    * 4 POS,FIM
      LOGICAL    * 1 MNEWTON
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      CALL LIB$INIT_TIMER(INICIO)
C
C
      IF( MNEWTON ) THEN

         DO 10 K = 1 , N
            INIC = IUPTR(K)
            FIM  = ILPTR( K + 1 ) - 1
            D    = A( INIC )
            DO 20 I = ( INIC + 1 ) , FIM
               A( I ) = A( I ) / D
20          CONTINUE
10        CONTINUE
       END IF
C
      MNEWTON  = .FALSE.
C
      CALL SOLVL(A,ICN,ILPTR,IUPTR,PMTL,RHS)
C
      EPS = 1.0E-04 * XNORMA
C
      DO 30 I = 1 , N
C
         POS = IUPTR( I )
         VETBL = B( I )
         UU = -FAT*( VETBL / A( POS ) )
         IF ( ABS( UU ) .GE. EPS ) THEN
C
             A( POS ) = ( RHS( I ) - VETBL ) / UU
C
      IF (ABS( A(POS) ) .LT. 1.0E-07) CALL FATSING(A(POS))
C
         END IF
C
         B( I ) = RHS( I )
C
30    CONTINUE
C
      CALL SOLVDU(A,ICN,ILPTR,IUPTR,PMTC,RHS)
C
      CALL CALCX(RHS,X,PMTC)
```

```
      CALL FUNCAO(X,RHS)
C
      CALL NORMA(RHS)
C
      CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
      TEMPO = FLOAT(IFIM)
C
C      ******* FINAL DIAGONAL ******
C
      RETURN
      END
C
C
C
C-----------------------------------------------------------
C                  B R O Y D E N
C-----------------------------------------------------------
C
C
      SUBROUTINE BROYDEN(X,RHS,TEMPO,NA,A,ICN,ILPTR,IUPTR,
     1   PMTL,PMTC,VBROY)
C
C
      COMMON / INFORMACOES / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV        / INDICADOR
      COMMON / NORMAS      / XNORMA,FNORMA
      COMMON / STEPCONTROL / FAT
      COMMON / CONTBROY    / KBROY,ITBROY,KBROYNS
C
C
      REAL * 4 X(1),RHS(1),A(1),VBROY(1)
      INTEGER  * 4 ICN(1),IUPTR(1),ILPTR(1),
     1        PMTL(1),PMTC(1),IDISP(2)
      LOGICAL * 1  FMELHORA
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      KBROY   = KBROY + 1
      KBROYNS = KBROYNS + 1
C
      CALL LIB$INIT_TIMER(INICIO)
C
      CALL  SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,RHS)
C
C     FORMULA RECURSIVA
C
      IS = 0
```

```
      IU = N
C
      DO I = 0 , ( KBROYNS - 2 )
C
         ALFAP = 0.0
C
C        PRODUTO ST * V
C
         DO J = 1 , N
            ALFAP = ALFAP + VBROY( IS + J ) * RHS( J )
         END DO
C
         DO J = 1 , N
            RHS( J ) = RHS( J ) - VBROY( IU + J ) * ALFAP
         END DO
C
         IS = IU + N
         IU = IS + N
C
      END DO
C
C     CALCULO DO VETOR U( KBROY - 1 )
C
      IS = ( 2*( KBROYNS - 1 ) * N )
C
      BETA = 0.0
      GAMA = 0.0
      WNORMA = 0.0
C
      DO I = 1 , N
C
         ELEM = VBROY( IS + I )
         W    = ELEM / FAT + RHS( I )
         BETA = BETA + ELEM * RHS( I )
         GAMA = GAMA + ELEM * W
         IF ( ABS( W ) .GT. WNORMA ) WNORMA = ABS( W )
C
      END DO
C
C     TESTE PARA DETECTAR SINGULARIDADE DE Bk
C
      PREC = 1.0E-04 * ( XNORMA * WNORMA ) + 1.E-25
C
      IF ( ABS( GAMA ) .LE. PREC ) THEN
C
C         SINGULARIDADE
C
          KBROYNS = KBROYNS - 1
C
      ELSE
C
          IU = IS + N
C
```

```
            TET = 1.0/FAT - 1.0
            DO I = 1 , N
              VBROY( IU + I ) = ( RHS(I) + TET*VBROY(IS+I) )/GAMA

            END DO
C
C         CALCULA RHS
C
          DO I = 1 , N
             RHS( I ) = RHS( I ) - VBROY( IU + I ) * BETA
          END DO
C
C         ATUALIZA APONTADOR P/ NOVO VETOR S
C
          IS = IU + N
C
          END IF
C                              FECHA TESTE DE SINGULARIDADE
C
          CALL CALCX(RHS,X,PMTC)
C
          DO I = 1 , N
             VBROY( I + IS ) = - RHS( I )
          END DO
C
          CALL FUNCAO(X,RHS)
C
          CALL NORMA(RHS)
C
          IF ( KBROYNS .EQ. ITBROY )  FMELHORA = .FALSE.
C
          CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
          TEMPO = FLOAT(IFIM)
C
C       ******* FINAL   BROYDEN ******
C
          RETURN
          END
C
C
C
C-----------------------------------------------------------
C                 M A R T I N E Z
C-----------------------------------------------------------
C
C
          SUBROUTINE MARTINEZ(X,RHS,TEMPO,NA,A,ICN,ILPTR,IUPTR,
     1     PMTL,PMTC,VCOL)
C
C
          COMMON / INFORMACOES / N
          COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
```

```
      COMMON / CONV          / INDICADOR
      COMMON / NORMAS        / XNORMA,FNORMA
      COMMON / STEPCONTROL / FAT
      COMMON / INDATZCOL     / INDCOL
      COMMON / CONTCOL       / KATCOL,LTCOL,KATCOLNS
C
C
      REAL * 4 X(1),RHS(1),A(1),VCOL(1)
      INTEGER  * 4 ICN(1),IUPTR(1),ILPTR(1),
     1        IKEEP(1),PMTL(1),PMTC(1),IDISP(2)
      LOGICAL * 1  FMELHORA
C
C
      INICIO = 0
      IFIM   = 0
      TEMPO  = 0
C
      KATCOL   = KATCOL + 1
      KATCOLNS = KATCOLNS + 1
C
      CALL LIB$INIT_TIMER(INICIO)
C
      CALL  SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,RHS)
C
C     FORMULA RECURSIVA
C
      IU = 1
C
      DO I = 1 , ( KATCOL - 1 )
C
           INDJ = INT(VCOL( IU ))
           ELEM = RHS( INDJ )
           DO J = 1 , N
                RHS( J ) = RHS( J ) - VCOL( IU + J ) * ELEM
           END DO
           IU = IU + N + 1
C
      END DO
C
C     CALCULO DO VETOR U
C
      SJ = VCOL( INDCOL + IU )
      VJ = RHS( INDCOL )
C
      SVJ = SJ / FAT + VJ
C
      TET = 1.0 / FAT  - 1.0
      DO I = 1 , N
           VCOL( I + IU ) = ( RHS(I) + TET*VCOL( IU + I ))/SVJ
      END DO
C
C     CALCULO DE  Bk*S = Fk
C
```

```
      DO I = 1 , N
         RHS( I ) = RHS( I ) - VCOL( IU + I ) * VJ
      END DO
C
C     ATUALIZACAO DO APONTADOR
C
      IU = IU + N + 1
C
      CALL CALCX(RHS,X,PMTC)
C
      VCOL( IU ) = FLOAT( INDCOL )
C
      DO I = 1 , N
         VCOL( I + IU ) = - RHS( I )
      END DO
C
      CALL FUNCAO(X,RHS)
C
      CALL NORMA(RHS)
C
      IF ( KATCOL .EQ. ITCOL )  FMELHORA = .FALSE.
C
      CALL LIB$STAT_TIMER(2,IFIM,INICIO)
C
      TEMPO = FLOAT(IFIM)
C
C     ******* FINAL   MARTINEZ ******
C
      RETURN
      END
C
C      END
C
C
C-----------------------------------------------------------------
C     SUBOUTINA  NORMA
C-----------------------------------------------------------------
C
      SUBROUTINE NORMA(F)
C
      COMMON / INFORMACOES   / N
      COMMON / CONV          / INDICADOR
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / NORMAS        / XNORMA,FNORMA
C
      REAL * 4    F(N)
C
      FNORMA = ABS( F(1) )
      DO 10 I = 2 , N
         IF ( ABS( F(I) ) .GT. FNORMA )  FNORMA = ABS( F(I) )
10    CONTINUE
C
      IF ( FNORMA .LT. FTOL )  INDICADOR = 2
```

```
      IF ( FNORMA .GT. FMAX )  INDICADOR = 3
C
      RETURN
      END
C
C
C-----------------------------------------------------------------------
C     SUBROTINA      CALCX
C-----------------------------------------------------------------------
C
      SUBROUTINE CALCX(DELTAX,X,PMTC)
C
      COMMON / INFORMACOES   / N
      COMMON / TOLERANCIAS / DISTX,FMAX,FTOL,ITMAX,XTOL
      COMMON / CONV          / INDICADOR
      COMMON / NORMAS        / XNORMA,FNORMA
      COMMON / STEPCONTROL / FAT
      COMMON / INDATZCOL   / INDCOL
C
C
      REAL * 4   DELTAX(N),X(N)
      INTEGER * 4 PMTC(N)
C
      XKNORMA = ABS( X(1) )
      XNORMA  = ABS( DELTAX(1) )
      FAT = 1.0
      INDCOL = 1
C
      DO 10 I = 2 , N
         IF ( ABS( X(I) ) .GT. XKNORMA )  XKNORMA = ABS( X(I) )
         IF ( ABS(DELTAX(I)) .GT. XNORMA) THEN
                                          XNORMA = ABS(DELTAX(I))
                                          INDCOL = I
         END IF
10    CONTINUE
C
      IF ( XNORMA .GT. DISTX )  THEN
         FAT = DISTX / XNORMA
         DO 20 I = 1 , N
            DELTAX(I) = DELTAX(I) * FAT
20       CONTINUE
         XNORMA = DISTX
       END IF
C
       DO 30 I = 1 , N
          X( PMTC(I) ) = X( PMTC(I) ) - DELTAX(I)
30     CONTINUE
C
       PREC = XTOL * ( 1 + XKNORMA)
       IF ( XNORMA .LT. PREC ) INDICADOR = 1
C
       RETURN
       END
```

```
C
C-----------------------------------------------------------------
C           P R E S O L V E
C-----------------------------------------------------------------
C
      SUBROUTINE PRESOLVE(IKEEP,ILPTR,IUPTR,PMTL,PMTC)
C
      COMMON / INFORMACOES / N
C
      INTEGER * 4 IKEEP(1),ILPTR(N),IUPTR(N),PMTC(N),PMTL(N)
C
      PMTL(1) = IKEEP(N + 1)
      PMTC(1) = IKEEP(2*N +1)
      ILPTR(1) = 1
      IUPTR(1) = 1
      DO 11 K = 2 , N
            ILPTR(K) = ILPTR(K-1) + IKEEP(K-1)
            IUPTR(K) = ILPTR(K) + IKEEP(3*N +K)
            PMTL(K) = IKEEP(N + K)
            PMTC(K) = IKEEP(2*N + K)
11    CONTINUE
C      ILPTR(N+1) = IUPTR(N) + 1
      RETURN
      END
C
C-----------------------------------------------------------------
C                       S O L V E
C-----------------------------------------------------------------
C
      SUBROUTINE SOLVE(A,ICN,ILPTR,IUPTR,PMTL,PMTC,B)
C
      COMMON / INFORMACOES / N
C
      REAL     * 4 A(1),B(N),BB(5000)
      INTEGER * 4 ILPTR(N),IUPTR(N),PMTC(N),PMTL(N),ICN(1)
C
      DO  38 I = 1 , N
          BB( I ) = B( PMTL(I) )
38    CONTINUE
C
      DO 91 K = 2 , N
            PROD = 0.0
            DO 92 KK =  ILPTR(K) , IUPTR(K) - 1
               PROD = PROD + A(KK)*BB(ICN(KK))
92          CONTINUE
            BB(K) = BB(K) + PROD
91    CONTINUE
C
C     RETRO SUBSTITUICAO
C
      BB(N) = BB(N)/A(IUPTR(N))
      DO 87 K = (N-1),1,-1
         PROD = 0
```

```
            DO 88 KK = ( IUPTR(K) + 1 ) , ( ILPTR(K+1) - 1 )
               PROD = PROD + A(KK)*BB(ICN(KK))
88          CONTINUE
         BB(K) = ( BB(K) - PROD )/A(IUPTR(K))
87       CONTINUE
C
          DO 47 I = 1 , N
          B( I ) =  BB(I)
47        CONTINUE
C
          RETURN
          END
C
C
C-----------------------------------------------------------------------
C                  S O L V L
C-----------------------------------------------------------------------
C
          SUBROUTINE SOLVL(A,ICN,ILPTR,IUPTR,PMTL,RHS)
C
          COMMON / INFORMACOES / N
C
          INTEGER * 4 ILPTR(1),IUPTR(1),PMTL(1),ICN(1)
          REAL    * 4 A(1),RHS(1),B(6000)
C
          DO  38 I = 1 , N
              B( I ) = RHS( PMTL(I) )
38        CONTINUE
C
          DO 91 K = 2 , N
                PROD = 0.0
                DO 92 KK =  ILPTR(K) , IUPTR(K) - 1
                   PROD = PROD + A(KK)*B(ICN(KK))
92              CONTINUE
                B(K) = B(K) + PROD
91        CONTINUE
          DO 39 I = 1 , N
             RHS( I ) = B( I )
39        CONTINUE
C
         RETURN
         END
C
C-----------------------------------------------------------------------
C                  S O L V U
C-----------------------------------------------------------------------
C
         SUBROUTINE SOLVU(A,ICN,ILPTR,IUPTR,PMTC,RHS)
C
         COMMON / INFORMACOES / N
C
         INTEGER * 4  ILPTR(1),IUPTR(1),PMTC(1),ICN(1)
         REAL    * 4  A(1),RHS(1),B(6000)
```

```
C
      B(N) = RHS(N)/A(IUPTR(N))
      DO 87 K = (N-1),1,-1
         PROD = 0
         DO 88 KK = ( IUPTR(K) + 1 ) , ( ILPTR(K+1) - 1 )
            PROD = PROD + A(KK)*B(ICN(KK))
88       CONTINUE
      B(K) = ( RHS(K) - PROD )/A(IUPTR(K))
87    CONTINUE
C
C       DO 26 I = 1 , N
C          B(I) = RHS(I)
C 26       CONTINUE
C
       DO 47 I = 1 , N
       RHS( I ) =  B(I)
47     CONTINUE
C
     +  RETURN
       END
C
C-----------------------------------------------------------
C                S  O  L  V  D  U
C-----------------------------------------------------------
C
      SUBROUTINE SOLVDU(A,ICN,ILPTR,IUPTR,PMTC,RHS)
C
      COMMON / INFORMACOES / N
C
      INTEGER * 4  ILPTR(1),IUPTR(1),PMTC(1),ICN(1)
      INTEGER * 4  POS,ETAPA,FIM
      REAL    * 4  A(1),RHS(1),B(6000)
C
C
      DO 10  I = 1 , N
         POS      = IUPTR( I )
         B( I ) = RHS( I ) / A( POS )
10    CONTINUE
C
      DO 87 K = (N-1),1,-1
C
         PROD = 0
C
         DO 88 KK = ( IUPTR(K) + 1 ) , ( ILPTR(K+1) - 1 )
            PROD = PROD + A(KK)*B(ICN(KK))
88       CONTINUE
      B(K) =  B(K) - PROD
87    CONTINUE
C
C       DO 26 I = 1 , N
C          B(I) = RHS(I)
C 26       CONTINUE
```

```
C
      DO 47 I = 1 , N
      RHS( I ) =  B(I)
47    CONTINUE
C
      RETURN
      END
C
C-----------------------------------------------------------------
C                F A T S I N G
C-----------------------------------------------------------------
C
      SUBROUTINE FATSING( W )
C
      REAL * 4 W

C
      IF ( W .LT. 0.0 ) THEN
         W = - 1.0E-07
        ELSE
         W =   1.0E-07
      END IF
C
      RETURN
      END
```

BIBLIOGRAFIA

[1] C. G. BROYDEN, "A class of methods for solving nonlinear simultaneous equations", Math. Comp. V. 19, 1965, pp. 577-593.

[2] C. G. BROYDEN, "The convergence of an algorithm for solving sparse nonlinear systems", Math Comp. V. 25, 1974, pp. 285-294

[3] C. G. BROYDEN, J. E. DENNIS and J. J. MORÉ, "On the local and superlinear convergence of quasi-Newton methods", J. Inst. Math. Appl. V. 12, 1973, pp. 223-245.

[4] P. A. BUSINGER, "Monitoring the numerical stability of Gaussian elimination", Numerische Math.16, 1971 pp. 360-361.

[5] F. F. CHADEE, "Sparse quasi-Newton methods and the continuation problem", T. R. SOL 85-8, Dept. of Operations Research, Stanford University, Stanford, Ca, 1985.

[6] A. K. CLINE, C. B. MOLER, G. W. STEWART and J. H WILKINSON, "An estimate for the contition number of a matrix", SIAM J. Num. Anal. 16, 1979, pp. 368-375.

[7] T. F. COLEMAN, B. S. GARBOW and J. J. MORÉ, "Software for estimation of sparse Jacobian matrices", ACM Trans. Math. Software, V. 10, 1984, pp. 329-345.

[8] A. R. CURTIS and J. K. REID, "The solution of large sparse unsymmetric systems of linear equations", J. Inst. Maths. Applics. 8, 1971, pp. 344-353.

[9] J. E. DENNIS and E. S. MARWIL, "Direct secant updates of matrix factorizations", Math. Comput., V. 38, 1982, pp. 459-476.

[10] J. E. DENNIS and J. J. MORÉ, "A characterization of superlinear convergence and its application to quasi-Newton methods", Math. Comp., V. 28, 1974, pp. 549-560.

[11] J. E. DENNIS Jr. and J. J. MORÉ, "Quasi-Newton methods, motivation and theory", SIAM R., V. 1977, pp. 46-89.

[12] J. E. DENNIS and R. SCHNABEL, "Numerical methods for unconstrained optimization and nonlinear equations", Prentice-Hall Series in Comput. Math., Prentice-Hall, NJ, 1983.

[13] I. S. DUFF, "MA28 - A set of FORTRAN subroutines for sparse unsimetric linear equations", Tech. Report AERE R-8730, Harwel, England, 1977.

[14] I. S. DUFF, "Sparse matrices and their uses." Academic Press, New York and London, 1981.

[15] I. S. DUFF, A. M. ERISMAN and J. K. REID, "Direct Methods for Sparse Matrices", Clarendon Press, Oxford,1986.

[16] I. S. DUFF and G. W. STEWART, "Sparse matrix proceedings", 1978, SIAM, Philadelphia.

[17] A. C. DURAN, "Resolução de sistemas não lineares esparsos. Sua aplicação na resolução do problema de fluxo de carga em redes de de energia elétrica", Campinas, IMECC-UNICAMP, 1989, (Tese de Mestrado).

[18] C. W. GEAR, "Numerical erros in sparse linear equations" Report UIUCDC-F-75-885, Department of computer Science, University of Illinois at Urbana-Champaign, Illinois, 1975.

[19] A. GEORGE and Ng, "Symbolic factorization for sparse Gaussian elimination with partial pivoting", SIAM Journal on Scientific and Statistical Computing, V. 8, 1987,pp. 877-898.

[20] P. E. GILL, W. MURRAY and M. H. WRIGHT, "Practical Optimization", Academic Press, New York and London, 1981.

[21] G. H. GOLUB and C. H. VAN LOAN, "Matrix Computation", The Johns Hopkins University Press, Baltimore, 1983.

[22] M. A. GOMES-RUGGIERO, "Resolução numérica de sistemas não lineares de grande porte", Campinas, FEE-UNICAMP, 1990 (Tese de Doutorado).

[23] M. A. GOMES-RUGGIERO and J. M. MARTINEZ, "The Column Updating Method for solving nonlinear equations in Hilbert space", RAIRO M2AN, 1989 (por aparecer).

[24] M. A. GOMES-RUGGIERO, J. M. MARTINEZ and A. C. MORETTI, "Implementing Algorithms for Solving Sparse Nonlinear System of Equations", submetido a SIAM J. Scientific and Statistical Computing, atualmente sendo revisado.

[25] G. W. JOHNSON and N. H. AUSTRIA, "A quasi-Newton method employing direct secant updates of matrix factorizations", SIAM J. Mumer. Anal. V. 20, 1983, pp.315-325.

[26] D. E. KNUTH, "Sorting and searching, The art of computer programming III", Addison-Wesley, Massachusetts, Paolo Alto and London, 1973.

[27] D. G. LUENBERGER, "Linear and nonlinear programming", 2nd ed.,Addison-Wesley, Reading, Mass., 1984.

[28] J. M. MARTINEZ, "A Quasi-Newton Method with Modification of one Column per Iteration", Computing, V. 33, 1984, pp. 353-362.

[29] J.M. MARTINEZ, "Local convergence theory of Inexact Newton Methods based on structured least change updates", Math. Comput., V. 5, 1990, N. 191.

[30] J. M. MARTINEZ, "A quasi-Newton method with a new updating for LDU factorization of the approximate Jacobian", Mat., Aplic. e Comput., V. 2, 1983, pp. 131-142.

[31] J. M. MARTINEZ, "A quasi-Newton method with factorization scaling for solving sparse nonlinear system of equations", Computing, V. 38, 1987, pp. 131-141.

[32] J. M. MARTINEZ, "A family of quasi-Newton methods with direct secant updates of matrix factorizations", SIAM J. on Numerical Analysis, to appear, 1988.

[33] E. S. MARWIL, "Convergence results for Schubert's method for solving sparse nonlinear equations", SIAM J. Numer. Anal. V. 16, 1979, pp 588-604.

[34] J. J. MORÉ and J. A. TRANGENSTEIN, "On the global convergence of Broyden's method", Math. Comp., V. 30, 1976, pp. 523-540.

[35] J. M. ORTEGA and W. C. RHEINBOLDT, "Iterative solution of nonlinear equations in several variables", New York, Academic Press 1970.

[36] O. OSTERBY and Z. ZLATEV, "Direct methods for sparse matrices", Lecture notes in comp. sci. 157, Spring-Vrelag, Berlin, Heidelberg, New York and Tokyo, 1983.

[37] J. K. REID, "A note on stability of Gaussian elimination", J. Inst. Maths. Applics. 8, 1971, pp. 375-375.

[38] L. K. SCHUBERT, "Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian", Math. Comput., V. 24, 1970, pp. 27-30.

[39] H SCHUWANDT, "A interval arithmetic approach for construction of a almost grobally convergent method for the solution of the nonlinear Poisson equation on the unit square", SIAM J. Sci. Stat. Comput. V. 5, 1984, pp. 427-452.

[40] Ph. L. TOINT, "Numerical solution of large sets of algebraic nonlinear equations", Math. Comp., V. 16, 1986, pp. 175-189.

[41] J. H. WILKINSON, "The algebric eigenvalue problem", Oxford University Press, London, 1965.

[42] M. C. WOLFF, "Técnicas de resolução de sistemas esparsos e atualização de matrizes para uso em algorítmos de otimização", Campinas IMECC-UNICAMP, 1988 (Tese de mestrado).

[43] Z. ZLATEV, "On some pivotal strategies in Gaussian elimination by sparse technique" SIAM J. Numer. Anal. 17, 1980, pp. 18-30.