


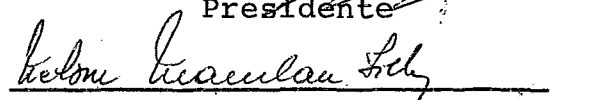
IMPLEMENTAÇÃO DE UM SISTEMA DE
ALGORITMOS DE PROGRAMAÇÃO NÃO LINEAR

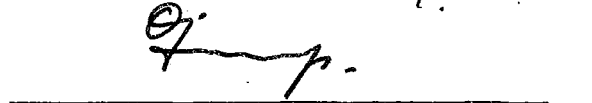
Clóvis Augusto Ribeiro

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE
PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JA-
NEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO
GRAU DE MESTRE EM CIÊNCIA (M.Sc.)

Aprovada por:



Presidente




RIO DE JANEIRO
ESTADO DA GUANABARA - BRASIL
DEZEMBRO DE 1973

À Marcia,
Eduardo e Georgiana.

AGRADECIMENTOS

Ao amigo e professor Dr. Clóvis Caezar Gonzaga pela orientação segura e incansável dedicação a este trabalho.

Ao Magnífico Reitor da Universidade Federal de Ouro Preto, Engenheiro Geraldo Parreiras e à Diretoria da Faculdade Federal de Minas e Metalurgia nas pessoas dos professores Dr. Antônio Pinheiro Filho, Dr. Antônio Moreira Calaes, Dr. Washington Moraes de Andrade, Dr. Walter Arcanjo Dornelas e Dr. Wagner Colombarolli pelo apoio que me dispensaram.

Aos amigos e colegas da COPPE por valiosas sugestões apresentadas.

Aos diretores e funcionários do NCE-UFRJ pela colaboração indispensável à realização deste trabalho.

À UFOP e à COPPE pelo imprescindível apoio financeiro.

RESUMO

Este trabalho resulta da implementação, em computador, de uma série de algoritmos destinados a resolver problemas vinculados e desvinculados da área de Programação Linear.

Os métodos são apresentados de forma resumida e, em seguida, são expostos os algoritmos correspondentes.

O trabalho é de cunho essencialmente prático e não houve, por este motivo, intenção de estabelecer desenvolvimentos teóricos inéditos, a menos de uma pequena ênfase sobre manipulação de precisões utilizadas pelos algoritmos.

O sistema foi organizado de modo a oferecer facilidades de operação aos usuários que dele fizerem uso.

ABSTRACT

This work is the result of the computer implementation of a set of nonlinear programming algorithms destined to solve constrained or unconstrained problems.

The methods are concisely discussed after which the corresponding algorithms are presented.

As the algorithms are approached from an essentially practical point of view, no original theoretical developments are pursued with the possible exception of some emphasis on the precision manipulations made by the algorithms.

The system has been organized in order to provide to the user a good easiness in operation.

INDICE

CAPÍTULO	I - INTRODUÇÃO	1
CAPÍTULO	II - O PROBLEMA E SUAS SOLUÇÕES	4
Seção	1 - Formulação do problema	5
Seção	2 - Finalidades do trabalho	7
Seção	3 - A busca unidirecional	8
Seção	4 - Minimização desvinculada	10
Seção	5 - Minimização vinculada	13
Seção	6 - Programação Linear	14
CAPÍTULO	III - MÉTODOS DE BUSCA UNIDIRECIONAL	16
Seção	1 - Método de Goldstein	17
Seção	2 - Método de Armijo	20
Seção	3 - Método de Secção Áurea (Fibonacci)	21
Seção	4 - Método de Davies-Swann-Campey-Powell	24
CAPÍTULO	IV - MÉTODOS DE MINIMIZAÇÃO DESVINCULADA COM DE RIVADAS	27
Seção	1 - Método de Cauchy	29
Seção	2 - Método de Fletcher-Reeves	31
Seção	3 - Método de Davidon-Fletcher-Powell	34
Seção	4 - Método de Broyden	40
CAPÍTULO	V - MÉTODOS DE MINIMIZAÇÃO DESVINCULADA SEM DE RIVADAS	43
Seção	1 - Método de Cauchy	44
Seção	2 - Método de Powell	46
Seção	3 - Método de Nelder-Mead	50
CAPÍTULO	VI - MÉTODOS DE MINIMIZAÇÃO VINCULADA COM DERI VADAS	57
Seção	1 - Método de Penalidades	59
Seção	2 - Método de Direções Viáveis	68

CAPÍTULO VII - MÉTODOS DE MINIMIZAÇÃO VINCULADA SEM DERIVADAS	76
Seção 1 - Método de Tolerância Flexível	77
CAPÍTULO VIII - MANIPULAÇÃO DE PRECISÕES	85
Seção 1 - Modelo implementável	87
Seção 2 - Método de Penalidades	93
Seção 3 - Método de Direções Viáveis	95
CAPÍTULO IX - ESTRUTURA DO SISTEMA	97
CAPÍTULO X - UTILIZAÇÃO DO SISTEMA	108
CAPÍTULO XI - CONCLUSÕES E SUGESTÕES	131
CAPÍTULO XII - LISTAGENS DOS PROGRAMAS	138
BIBLIOGRAFIA -	203

CAPÍTULO IINTRODUÇÃO

O presente trabalho é fruto de um idéia há algum tempo existente no Programa de Engenharia de Sistemas da COPPE-UFRJ.

A intenção era reunir métodos de programação não linear de bom desempenho, em um único bloco, para permitir aos interessados na área resolver, com relativa facilidade, os problemas com que se deparassem. Para tanto foram implementados, em computador, quinze algoritmos considerados eficientes e que são descritos sucintamente no capítulo II e em maiores detalhes nos seguintes.

O sistema foi organizado de sorte a trazer facilidades ao usuário na preparação de dados e seleção dos algoritmos que pretenda usar na solução de seus problemas. Esta parte é apresentada, em detalhes no capítulo X onde se encontram alguns exemplos de utilização do sistema.

Durante a elaboração do sistema grande também foi a preocupação em permitir, sem muitas alterações, a posterior inclusão de outros métodos de bom desempenho que não tenham sido incorporados ao conjunto. Considerações básicas a respeito são feitas no capítulo IX onde também se apresenta a estrutura do sistema ressaltando a ligação e o funcionamento relativos entre os diversos algoritmos implementados.

No capítulo VIII aborda-se um importante tema relacionado com precisões, regras de paradas e convergência dos algoritmos expostos nos capítulos de III a VII. Tais algoritmos estão divididos em métodos desvinculados e vinculados, ambos com e sem derivadas, além das buscas unidirecionais que são especificamente tratados no capítulo III. Na descrição de cada método procura-se, resumidamente, destacar sua origem, seu funcionamento e convergência, apresentando-se então o algoritmo usado basicamente na implementação. Em face do cunho profundamente prático de que se reveste o presente trabalho, não houve maiores preocupações com relação a desenvolvimentos teóricos, limitando-nos à descrição de processos existentes.

Os capítulos XI e XII trazem, respectivamente alguns resultados, sugestões e as listagens dos programas.

Referências a obras e/ou autores são numéricas e apresentadas no texto entre barras verticais, podendo ser encontradas na bibliografia após o último capítulo.

NOTAÇÃO - Alguns esclarecimentos são necessários quanto à notação empregada no presente trabalho:

- a) O termo "pertence" inerente à teoria dos conjuntos é representado pela letra grega ϵ .
- b) As expressões $\langle X, Y \rangle$ e $X'Y$ representam, indiferentemente o produto escalar entre os vetores X e Y , enquanto que XY' indica o produto matricial entre eles.
- c) Letras maiúsculas são usadas para representar conjuntos, matrizes ou vetores. Letras minúsculas para escalares. No capítulo X, entretanto, são utilizadas minúsculas na representação de vetores, na parte correspondente a programação linear.
- d) Vetores são representados por colunas (entre colchetes) ou linha (entre parênteses).
- e) O espaço euclidiano n-dimensional é representado por R^n e o conjunto dos números inteiros por N .

CAPÍTULO II

O PROBLEMA E SUAS SOLUÇÕES

Conforme já tivemos a oportunidade de ressaltar no capítulo anterior, faremos aqui uma síntese de todo o trabalho, na intenção de oferecer ao leitor uma visão de conjunto, embora o material descrito nas seções seguintes seja desenvolvido, em maiores detalhes, nos capítulos subsequentes.

Obedecendo a uma divisão natural, começaremos pela proposição do problema a ser estudado, sob suas diversas formas, passando em seguida à descrição da finalidade do trabalho, partindo finalmente para a exposição sucinta dos métodos de solução estudados.

Na seção 6 abordaremos o problema de programação linear (PPL) como um tema isolado e como um subproblema do método de direções viáveis.

O presente trabalho se constitui de quatro algoritmos de minimização desvinculada com derivadas e três sem derivadas, dois algoritmos de minimização vinculada com derivadas e um sem derivadas, um algoritmo de programação linear e quatro buscas unidirecionais. São pois, no total, quinze algoritmos implementados em computador e destinados a resolver problemas de programação não linear. A estrutura do sistema é apresentada no capítulo IX e as listagens no capítulo XII.

SEÇÃO 1 - FORMULAÇÃO DO PROBLEMA1 - O PROBLEMA VINCULADO

Dadas as funções continuamente diferenciáveis $f: \mathbb{R}^n \rightarrow \mathbb{R}$, $g: \mathbb{R}^n \rightarrow \mathbb{R}^m$ e $h: \mathbb{R}^n \rightarrow \mathbb{R}^\ell$, encontrar, se existir, um ponto \hat{X} no conjunto $V = \{X/g(X) \leq 0, h(X) = 0\}$ tal que para todo $X \in V$, $f(\hat{X}) \leq f(X)$.

O problema tal como formulado em (1) pode ser reescrito em forma mais compacta:

2 Minimizar $f(X)$

sujeito a

$$g(X) \leq 0$$

$$h(X) = 0,$$

ou ainda:

3 Minimizar $f(X)$

sujeito a

$$g_i(X) \leq 0, \quad i=1,2,\dots,m$$

$$h_j(X) = 0, \quad j=1,2,\dots,\ell$$

em (1), (2) e (3)

f é a função critério ou função objetivo,

g representa os vínculos de desigualdade,

h representa os vínculos de igualdade,

V é o conjunto viável,

$g_i: \mathbb{R}^n \rightarrow \mathbb{R}$ é a i -ésima componente de g ,

$h_i: \mathbb{R}^n \rightarrow \mathbb{R}$ é a i -ésima componente de h ,
 m é o número de restrições de desigualdades e
 l o número de restrições de igualdade.

4

O PROBLEMA DESVINCULADO

Dada a função $f: \mathbb{R}^n \rightarrow \mathbb{R}$, continuamente diferenciável, encontrar, se existir, um ponto $\hat{X} \in \mathbb{R}^n$ tal que para todo $X \in \mathbb{R}^n$, $f(\hat{X}) \leq f(X)$.

O problema (4) pode ser reescrito sob a forma

5

Minimizar $f(X)$.

Como se pode observar, o problema (5) é um caso particular de (2).

SEÇÃO 2 - FINALIDADES DO TRABALHO

O presente trabalho visa a facilitar ao usuário o tratamento de problemas de programação não linear. Evidentemente as restrições existem e são apontadas no capítulo X.

Conforme exposto anteriormente, o trabalho se compõe de uma série de algoritmos implementados em computador e a idéia básica é fornecer, de maneira eficiente e na medida do possível, as facilidades de que necessita o usuário para solucionar problemas de otimização não linear. Veremos no capítulo X que os elementos de entrada para a execução do programa são, relativamente, em pequeno número e que é até mesmo possível deixar ao sistema a tarefa de selecionar os algoritmos de acordo com o tipo de problema.

Todas as informações concernentes à utilização do sistema são encontradas no capítulo X e sua estrutura é esquematizada no capítulo IX.

Procuramos tomar como base, na seleção dos algoritmos implementados, os resultados apresentados por HIMMELBLAU, [1] e POLAK, [2], por concordarmos com os dizeres de TABAK, [3], segundo o qual as duas obras apontadas constituem atualmente a melhor escolha, o primeiro prática e o segundo teoricamente.

SEÇÃO 3 - A BUSCA UNIDIRECIONAL

Um dos fatores mais importantes na eficiência de quase todos os algoritmos de programação não linear está na acertada escolha da busca unidirecional utilizada.

Em linhas gerais, um estágio de um algoritmo de minimização escolhe uma direção S sobre a qual deve-se efetuar uma busca a partir de um ponto $X \in R^n$. Como resultado, obtém-se um ponto $\hat{X} = X + \lambda'S$, tal que $f(\hat{X}) \leq f(X)$. O ponto \hat{X} deve reduzir bastante o valor de f , de modo a garantir a convergência dos algoritmos em que as buscas são utilizadas, como comentaremos ao apresentar cada um dos métodos de busca unidirecional, no capítulo III.

A figura 1 entremostra o funcionamento da busca unidirecional

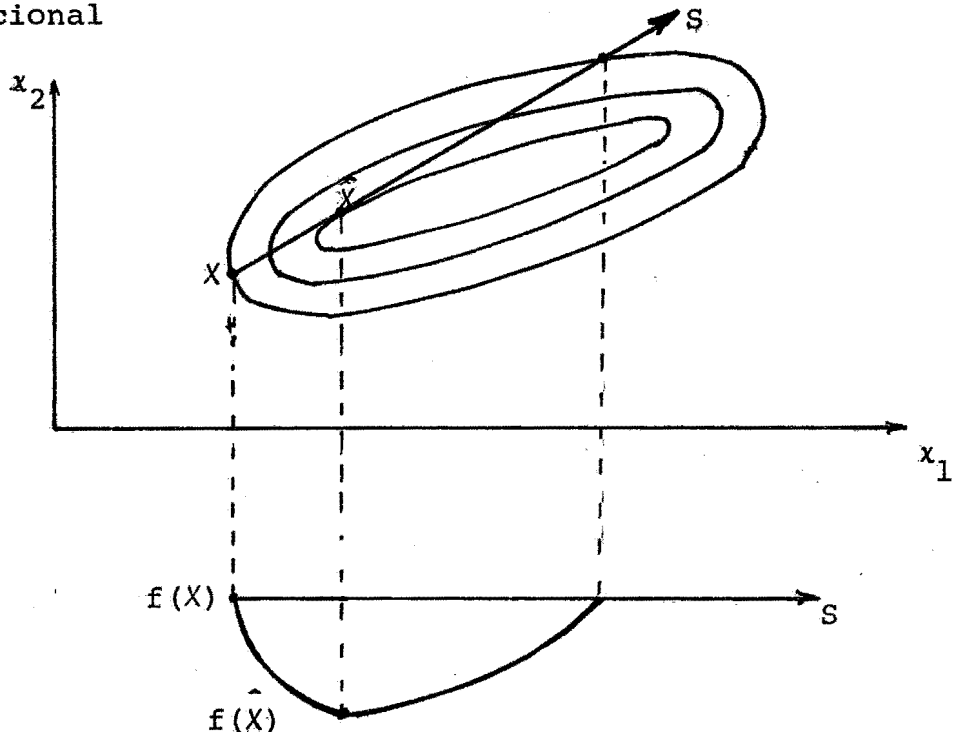


Fig. 1

Existem diversas técnicas de busca, |1|, |2|, |4|, e para o presente trabalho foram selecionadas quatro buscas respectivamente descritas nas seções de 1 a 4 do capítulo III.

A manipulação de precisões, abordada no capítulo VIII é um fator preponderante na eficiência das buscas.

SEÇÃO 4 - MINIMIZAÇÃO DESVINCULADA

6

MODELO CONCEITUAL

À excessão do processo de NELDER-MEAD (ver V.3) que utiliza uma técnica especial, todos os demais métodos de minimização desvinculada que fazem parte deste trabalho têm por base o seguinte modelo:

P1 . Escolha um ponto inicial $X_0 \in R^n$

P2 . Faça $i=0$

P3 . Calcule a partir de X_i uma direção apropriada $S_i \in R^n$

P4 . Se $\|S_i\| = 0$ PARE. Caso contrário vá para P5

P5 . Use uma busca unidirecional para calcular um escalar

$$\lambda' \geq 0 \text{ tal que}$$

$$f(X_i + \lambda'S_i) \leq f(X_i)$$

P6 . Faça $X_{i+1} = X_i + \lambda'S_i$, $i=i+1$ e vá para P3.

Existe uma grande variedade de métodos destinados a resolver o problema de minimização desvinculada (5) alguns utilizando derivadas, outros não. A diferença fundamental entre eles, está no passo P3 do modelo acima, isto é, a determinação da direção S_i . No modelo, assume-se que se um ponto X_i resolve o problema de minimização, então qualquer dos métodos fornece $S_i=0$. A menos de processos que usam princípios especiais como é o caso de NELDER-MEAD, podemos classificar os métodos de minimização desvinculada em quatro categorias principais:

- 1 - de GRADIENTES (steepest descent),
- 2 - de NEWTON,
- 3 - de DIREÇÕES CONJUGADAS,
- 4 - de MÉTRICA VARIÁVEL.

Em linhas gerais os métodos acima funcionam assim:

1. GRADIENTES - Utilizam o gradiente da função objetivo e determinam a direção de busca fazendo $S = - \nabla f(X)$.

2. NEWTON - Usam as derivadas segundas da função objetivo e determinam a direção S fazendo $S = -H^{-1}(X)\nabla f(X)$, onde $H(X)$ é a matriz Hessiana de $f(\cdot)$ no ponto X .

3. DIREÇÕES CONJUGADAS - Geram, para uma função quadrática com Hessiana definida positiva, um conjunto de direções S_i H -conjugadas, o que garante a minimização de f em, no máximo, n passos. Para funções não quadráticas perde-se essa propriedade mas a eficiência destes métodos, em tais casos, é comprovada (ver [1]). Intuitivamente, aproveita-se o fato de que, próximo de um ponto de mínimo, uma função convexa, "bem comportada", pode ser aproximada por uma quadrática.

4. MÉTRICA VARIÁVEL - Os métodos pertencentes a este grupo assemelham-se aos de NEWTON e por isto são também denominados QUASI-NEWTON. A diferença é que não utilizam as derivadas segundas. É feita uma aproximação da inversa da

Hessiana e no processo como cada aproximação é efetuada está a distinção entre os diversos métodos existentes. De um modo geral, a direção S é dada por

$$S = -E(X) \nabla f(X)$$

onde $E(X)$, também chamada matriz direcional, $|E|$, é obtida iterativamente a partir da anterior, sendo a inicial normalmente igual à matriz identidade. Maiores detalhes podem ser encontrados em (IV. 3 e 4).

Uma importante propriedade desta classe de métodos é que, para uma função quadrática, em n passos a matriz direcional se torna igual à inversa da Hessiana:

$$E_n(X) = H^{-1}(X).$$

Os métodos de métrica variável, em relação aos métodos de NEWTON, levam a vantagem de utilizar apenas informações da derivada primeira e contar com quase a mesma eficiência. Entretanto, comparados aos métodos de FLETCHER-REEVES ou CAUCHY, possuem a desvantagem de exigir substancialmente mais memória quando implementados em computador.

SEÇÃO 5 - MINIMIZAÇÃO VINCULADA

Dentre os métodos que foram desenvolvidos com a finalidade de resolver o problema (2), há os que empregam gradientes em sua própria teoria, outros que usam gradientes como uma ferramenta auxiliar e, finalmente, aqueles que dispensam tais informações. Exemplos de tais casos são respectivamente apresentados em (VI. 1 e 2) e (VII. 1).

Os métodos de minimização vinculada mais utilizados podem ser classificados, basicamente, em três categorias, |1|:

1. Extensão da metodologia linear a problemas de programação não linear através de repetidas aproximações lineares.

2. Transformação do problema de programação não linear em uma série de problemas desvinculados pelo uso de funções penalidades.

3. Uso de tolerâncias flexíveis para acomodar pontos viáveis e não viáveis.

Em nosso caso, foram selecionados três algoritmos, um de cada categoria respectivamente:

1. Método de direções viáveis,
2. Método de penalidades,
3. Método de tolerância flexível

Serão descritos os dois primeiros no capítulo VI e o último no capítulo VII. Sua utilização, em computador, é apresentada no capítulo X .

SEÇÃO 6 - PROGRAMAÇÃO LINEAR

Já tivemos a oportunidade de frisar que a programação linear foi incluída, no presente trabalho, com a finalidade precípua de resolver um sub-problema do método de direções viáveis, a ser abordado no capítulo VI. Conforme se verá, a programação linear é empregada para resolver um problema cuja solução ótima fornece a direção $S \in R^n$ de busca para o método de direções viáveis. Entretanto, o programa apresentado no capítulo XII poderá também ser usado para solucionar problemas isolados de programação linear conforme explicado na seção 3 do capítulo X.

O problema geral de programação linear é definido da seguinte maneira:

7. Encontrar, se existir, $\hat{X} \in V$, tal que

$$C' \hat{X} = \min\{C'X / X \in V\} \quad \text{onde}$$

$$V = \{X \in R^n / A X = b, \quad X \geq 0\}$$

O problema (7) pode ser reescrito:

8. Minimizar $C'X$, $X \in R^n$,

$$\text{sujeito a } A X = b,$$

$$X \geq 0$$

Em (7) e (8):

V é o conjunto viável,

$C \in R^n$ é o vetor custo,

$b \in R^m$ é o vetor básico ou restrição de recursos,

A é uma matriz $(m \times n)$.

O método simplex revisado é uma técnica eficiente destinada a resolver o problema (8) e um programa foi escrito (ver XII) com esta finalidade.

Em face de ser a programação linear um tema bastante difundido na literatura de otimização, deixamos de descrever, no presente trabalho, o método simplex revisado que é utilizado na solução do subproblema de direções viáveis e de problemas de programação linear. Na seção 3 do capítulo X apresentamos a técnica de utilização do programa na solução de PPL, isto é, a disposição que deve ser obedecida para a entrada dos dados em computador. Quando o simplex é empregado na solução do subproblema de direções viáveis, a transmissão de dados é feita interna e automaticamente por rotinas descritas no capítulo IX.

CAPÍTULO IIIMÉTODOS DE BUSCA UNIDIRECIONAL

Já nos referimos, em (II.3), à importância das buscas unidirecionais dentro dos processos de resolução de problemas de programação não linear.

No presente capítulo estudaremos cada um dos métodos de busca utilizados pelos algoritmos apresentados em IV.

Vários são os processos de minimização unidirecional existentes. Em nosso caso fizemos a seleção de quatro métodos levando em consideração a eficiência que apresentam:

1. método de GOLDSTEIN,
2. método de ARMIJO,
3. método de SECÇÃO-AUREA (FIBONACCI),
4. método de DAVIES-SWANN-CAMPEY-POWELL.

Os dois primeiros, organizados e formalizados por POLAK em |2|, além de contarem com boa eficiência e não exigirem que a função seja convexa, são de relativa simplicidade. A única restrição que apresentam é usarem informações do gradiente de $f(\cdot)$.

O terceiro (ver |2|) e o quarto (ver |1|) são os que mostram melhor comportamento quando comparados a outros métodos existentes e não descritos aqui |1|.

Para os métodos apresentados, supõe-se conhecido um ponto $X \in R^n$ e uma direção de busca $S \in R^n$.

SEÇÃO 1 - MÉTODO DE GOLDSTEIN

Esta é uma técnica de busca unidirecional bastante eficiente que requer a disponibilidade do gradiente da função no ponto X , a partir do qual se quer efetuar a busca (ver II. 4. 6). Isto vale dizer que a função em questão deve ser necessariamente diferenciável. Não é exigida convexidade [2].

Consideremos as funções definidas por:

$$\begin{aligned} 1 \quad \theta(\lambda, X) &= f(X + \lambda S) - f(X), \\ 2 \quad \underline{\theta}(\lambda, X) &= \theta(\lambda, X) - \lambda(1-\alpha) \langle \nabla f(X), S \rangle, \\ 3 \quad \bar{\theta}(\lambda, X) &= \theta(\lambda, X) - \lambda\alpha \langle \nabla f(X), S \rangle, \end{aligned}$$

onde

$\lambda, \alpha \in \mathbb{R}$, $\alpha \in (0, 0.5)$, e $S \in \mathbb{R}^n$ é a direção de busca.

O algoritmo em (4) utiliza as funções (2) e (3) na determinação do valor de λ' (ver II. 3).

Uma boa escolha para α é fazer $\alpha=0.4$ (ver. [2]).

Esse método, bem como o algoritmo da seção 2, não se baseia na aproximação de um ponto de mínimo unidirecional: procura-se um ponto $X + \lambda'S$ capaz de fornecer um valor de f suficientemente baixo para que sejam satisfeitas condições de convergência de algoritmos, expostas em [2]. A figura III. 1 ilustra a maneira de definir um intervalo em que f aprofunda-se suficientemente.

Um esboço do que ocorre, geometricamente, com uma função $f: \mathbb{R}^2 \rightarrow \mathbb{R}$ é mostrado na figura 1, onde

$$\phi(\lambda) = \lambda \langle \nabla f(X), S \rangle,$$

$$\phi_1(\lambda) = \alpha \phi(\lambda),$$

$$\phi_2(\lambda) = (1-\alpha)\phi(\lambda).$$

As funções $\bar{\theta}(\cdot, X) = \theta(\cdot, X) - \phi_1$ e $\underline{\theta}(\cdot, X) = \theta(\cdot, X) - \phi_2$ determinam, sobre a direção S , um intervalo que contém o valor λ' procurado (ver II. 3).

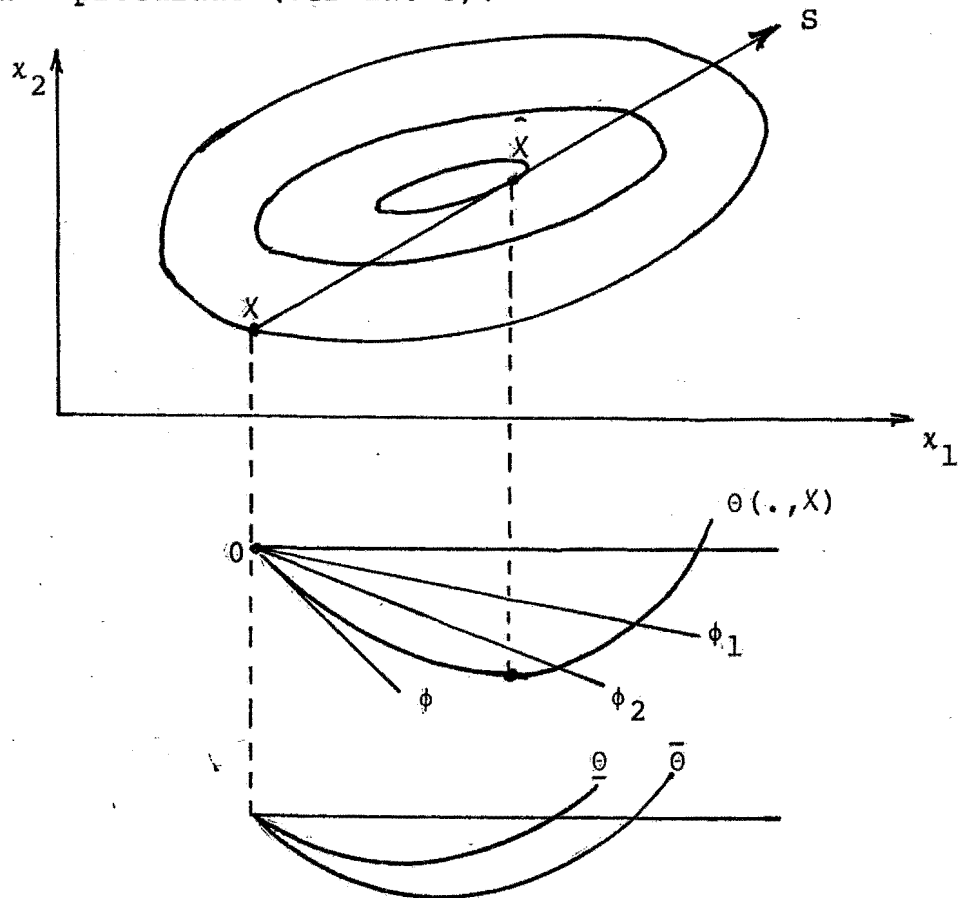


Fig. 1

O algoritmo que apresentamos a seguir se resume em obter-se λ' tal que $\underline{\theta}(\lambda', X) \geq 0$ e $\bar{\theta}(\lambda', X) \leq 0$.

4

ALGORITMO |5|

- P1 . Dados $X_i \in R^n$ e $S_i \in R^n$
- P2 . Escolha $\alpha \in (0, 0.5)$ e calcule $\rho > 0$ (ver VIII)
- P3 . Faça $\mu = \rho$.
- P4 . Calcule $\underline{\theta}(\mu, X_i)$ por (2)
- P5 . Se $\underline{\theta}(\mu, X_i) = 0$, faça $\lambda' = \mu$ e PARE. Se $\underline{\theta}(\mu, X_i) < 0$, faça $\mu = 2\mu$ e vá para P4. Se $\underline{\theta}(\mu, X_i) > 0$ vá para P6.
- P6 . Calcule $\bar{\theta}(\mu, X_i)$ por (3)
- P7 . Se $\bar{\theta}(\mu, X_i) \leq 0$, faça $\lambda' = \mu$ e PARE. Caso contrário ,
faça $a_0 = \mu/2$, $b_0 = \mu$ e vá para P8.
- P8 . Se $a_0 = \rho/2$ faça $a_0 = 0$
- Comentário Agora $\lambda' \in [a_0, b_0]$
- P9 . Faça $j = 0$
- P10. Faça $v_j = (a_j + b_j)/2$
- P11. Calcule $\underline{\theta}(v_j, X_i)$ e $\bar{\theta}(v_j, X_i)$
- P12. Se $\underline{\theta}(v_j, X_i) \geq 0$ e $\bar{\theta}(v_j, X_i) \leq 0$ faça $\lambda' = v_j$ e PARE.
- Caso contrário vá para P13.
- P13. Se $\underline{\theta}(v_j, X_i) > 0$ faça $a_{j+1} = a_j$, $b_{j+1} = v_j$, $j = j+1$ e vá
para P10. Caso contrário, faça $a_{j+1} = v_j$, $b_{j+1} = b_j$,
 $j = j+1$ e vá para P10.

SEÇÃO 2 - MÉTODO DE ARMIJO

Esta é uma busca unidirecional de menor eficiência em relação as outras três mas de grande simplicidade.

O método de Armijo apresenta uma certa semelhança com o processo de Goldstein pelo fato de utilizar a equação (3) na determinação de λ' . Um bom valor para λ' depende altamente do valor atribuído a β (ver 5) e do valor calculado para ρ (ver VIII). Uma boa escolha para α é fazer $\alpha=0,5$ (ver |2|)

5

ALGORITMO |6|

P1 . Dados $X_i \in R^n$ e $S_i \in R^n$

P2 . Escolha $\alpha \in (0,1)$, $\beta \in (0,1)$ e calcule $\rho > 0$ (ver VIII)

P3 . Faça $\mu = \rho$

P4 . Calcule $\bar{\theta}(\mu, X_i)$

P5 . Se $\bar{\theta}(\mu, X_i) \leq 0$, faça $\lambda' = \mu$ e PARE. Caso contrário, faça $\mu = \beta\mu$ e vá para P4.

SEÇÃO 3 - MÉTODO DE SECCÃO-ÂUREA

Este é um processo de busca unidirecional que tem se mostrado de grande eficiência quando comparado aos existentes (ver [1]).

Para sua aplicação não são necessárias informações da derivada primeira de $f(X)$ e, portanto, não há exigência de diferenciabilidade da função objetivo. Por outro lado $f(X)$ deve ser convexa ou nada se poderá garantir com relação ao novo ponto determinado pela busca.

Seja $\epsilon \geq 0$ uma precisão dada e X um ponto a partir do qual se quer efetuar a busca em uma direção conhecida S (ver II. 4.6). O que se pretende é determinar um $\lambda' \geq 0$ tal que $|\lambda' - \lambda^*| \leq \epsilon$, onde $\lambda^* \geq 0$ é algum valor de λ tal que

$$f(X + \lambda^*S) = \min\{f(X + \lambda S) / \lambda \geq 0\}$$

O método determina um intervalo inicial $[a, b]$ t.q. $\lambda^* \in [a, b]$ e, então, através de sucessivas divisões áureas vai diminuindo o tamanho de $[a, b]$ até atingir a condição $b - a \leq \epsilon$. O valor de λ' é calculado por

$$\lambda' = (b - a) / 2$$

No algoritmo apresentado a seguir os primeiros seis passos determinam um intervalo $[a_0, b_0]$ contendo o valor de λ' . Os demais estreitam o intervalo até atingir a precisão estabelecida por ϵ .

ALGORITMO | 2 |

- P0 . Dados $X \in \mathbb{R}^n$, $S \in \mathbb{R}^n$, $\varepsilon_1 \in \mathbb{R}$, $\varepsilon_2 \in \mathbb{R}$, $\beta \in [0.1, 0.5]$,
 $F_1 = (3 - \sqrt{5})/2 \approx 0.38$ e $F_2 = (\sqrt{5} - 1)/2 \approx 0.62$. Calcule $\rho > 0$
(ver VIII).
- P1 . Calcule $\theta(\rho) = f(X + \rho S) - f(X)$ e $\|S\|$.
- P2 . Faça $\varepsilon = \varepsilon_1 / \|S\|$, $\varepsilon' = \varepsilon_1 \|S\|$, $\varepsilon'' = \beta \varepsilon'$, $i=0$ e $\mu_0 = \rho$.
Comentário - ε é a precisão para a busca, enquanto
que ε' e ε'' são precisões para a função (ver VIII).
- P3 . Se $\theta(\rho) \geq 0$ faça $a_0 = 0$, $b_0 = \rho$, $j=0$ e vá para P7.
Caso contrário, vá para P4.
- P4 . Faça $\mu_{i+1} = 2\mu_i$.
- P5 . Calcule $\theta(\mu_{i+1})$.
- P6 . Se $\theta(\mu_{i+1}) \geq \theta(\mu_i)$, faça $a_0 = \mu_i/2$, $b_0 = \mu_{i+1}$, $j=0$ e
vá para P7. Caso contrário faça $i=i+1$ e vá para P4.
Comentário - Agora $\lambda^* \in [a_0, b_0]$.
- P7 . Se $a_0 = \rho/2$ faça $a_0 = 0$.
- P8 . Se $\ell_j = b_j - a_j \leq \varepsilon$ vá para P11. Caso contrário vá para P9.
- P9 . Faça $v_j = a_j + F_1 \ell_j$ e $w_j = a_j + F_2 \ell_j$.
- P10 . Se $\theta(v_j) < \theta(w_j)$ faça $a_{j+1} = a_j$, $b_{j+1} = w_j$, $j=j+1$ e vá
para P8.
Caso contrário faça $a_{j+1} = v_j$, $b_{j+1} = b_j$, $j=j+1$ e vá
para P8.
- P11 . Faça $\lambda' = (a_j + b_j)/2$ e calcule $\theta(\lambda')$.

P12. Se $\theta(\lambda') < -\varepsilon'$, PARE.

Caso contrário, vá para P13.

P13. Se $\varepsilon_1 < \varepsilon_2$ vá para P15.

Caso contrário vá para P14.

P14. Se $\theta(\lambda') \leq -\varepsilon''$ faça $\varepsilon = \beta\varepsilon$, $\varepsilon_1 = \beta\varepsilon_1$, $\varepsilon' = \varepsilon''$, $\varepsilon'' = \beta\varepsilon''$,
 $j = j+1$ e vá para P8.

Caso contrário faça $\varepsilon = \varepsilon/2$, $\varepsilon_1 = \varepsilon_1/2$, $\varepsilon' = \varepsilon'/2$, $\varepsilon'' = \varepsilon''/2$,
 $j = j+1$ e vá para P8.

P15. Se $\theta(\lambda') < 0$, PARE.

Caso contrário, a busca falhou, PARE

SEÇÃO 4 - MÉTODO DE DAVIES-SWANN-CAMPEY-POWELL

Esta é uma técnica que pertence a uma categoria de métodos que determinam, dentro de uma precisão pré-estabelecida, uma aproximação λ' para um ponto de mínimo unidirecional λ^* (ver seção 3), usando extrapolação e interpolação (ver |1|).

O processo dispensa informações sobre o gradiente mas a hipótese de convexidade da função é exigida para que se possa garantir que o novo ponto obtido não seja pior que o seu antecessor.

As estimativas quadráticas utilizadas usam apenas as informações de determinados pontos e valores da função nesses pontos.

O algoritmo apresentado em (9) é uma fusão de dois outros. O primeiro devido a DAVIES-SWANN-CAMPEY-|7| e o segundo a POWELL-|8|. Daquele utilizam-se os passos para a determinação do intervalo inicial que contém λ^* e deste os necessários à obtenção de um valor λ' através de progressivo estreitamento do intervalo inicial.

É utilizada a equação (1) no algoritmo.

Os seis primeiros passos estabelecem um intervalo inicial que contém λ^* e os seguintes estreitam o intervalo até atingir a precisão desejada (ver seção 3).

ALGORITMO | 1 |

Comentário

Durante o algoritmo: $\theta_i \equiv \theta(\lambda_i) = f(X) - f(X + \lambda_i S)$.

P0 . Dados $X \in R^n, S \in R^n, \epsilon_1 \in R, \epsilon_2 \in R, \beta \in [0.1, 0.5], \|S\|$. Calcule $\rho > 0$ (ver VIII).

P1 . Faça $\lambda_1 = 0, \theta_1 = 0, \lambda_2 = 0, \theta_2 = 0, \lambda_3 = 2\rho, \epsilon = \epsilon_1 / \|S\|, \epsilon' = \epsilon_1 \|S\|, \epsilon'' = \beta \epsilon'$, calcule θ_3 e faça $k=0$.

Comentário - ϵ é a precisão para a busca, enquanto que ϵ' e ϵ'' são precisões para a função (ver VIII).

P2 . Se $\theta_3 \leq \theta_1$ faça $\lambda_2 = \lambda_1 + \rho$, calcule θ_2 e vá para P6. Caso contrário vá para P3.

P3 . Faça $\lambda_1 = \lambda_2, \lambda_2 = \lambda_3, \lambda_3 = \lambda_3 + 2\rho, \theta_1 = \theta_2, \theta_2 = \theta_3$, calcule θ_3 .

P4 . Se $\theta_3 < \theta_2$ faça $\lambda_0 = \lambda_2 + \rho$, calcule θ_0 e vá para P5. Caso contrário faça $\rho = 2\rho$ e vá para P3.

P5 . Se $\theta_0 > \theta_2$ faça $\lambda_1 = \lambda_2, \theta_1 = \theta_2, \lambda_2 = \lambda_0, \theta_2 = \theta_0$ e vá para P10. Caso contrário faça $\lambda_3 = \lambda_0, \theta_3 = \theta_0$ e vá para P10.

P6 . Se $k=0$ calcule $\lambda_0 = \lambda_2 + \rho (\theta_1 - \theta_2) / 2(\theta_1 - 2\theta_2 + \theta_3)$, calcule θ_0 e vá para P8.

Caso contrário calcule $y = (\lambda_2 - \lambda_3)\theta_1 + (\lambda_3 - \lambda_1)\theta_2 + (\lambda_1 - \lambda_2)\theta_3$, $\lambda_0 = \lambda_0 + \rho$ e vá para P7.

P7 . Se $y=0$ vá para P17.

Caso contrário calcule:

$x = [(\lambda_2^2 - \lambda_3^2)\theta_1 + (\lambda_3^2 - \lambda_1^2)\theta_2 + (\lambda_1^2 - \lambda_2^2)\theta_3] / 2, \lambda_0 = x/y, \theta_0$ e vá para P8.

P8 . Calcule $\delta = (\lambda_0 - \lambda_1)x(\lambda_3 - \lambda_0)$. Se $\delta \leq 0$ vá para P9. Caso

contrário faça $k=1$ e vá para P11.

P9 . Faça $\lambda_0 = (\lambda_1 + \lambda_2)/2$, calcule θ_0 e vá para P11.

P10. Se $\lambda_3 - \lambda_1 < \epsilon$ vá para P14.

Caso contrário vá para P6.

P11. Se $\lambda_0 < \lambda_2$ vá para P13.

Caso contrário vá para P12.

P12. Se $\theta_0 > \theta_2$ faça $\lambda_1 = \lambda_2, \lambda_2 = \lambda_0, \theta_1 = \theta_2, \theta_2 = \theta_0$ e vá para P10.

Caso contrário faça $\lambda_3 = \lambda_0, \theta_3 = \theta_0$ e vá para P10.

P13. Se $\theta_0 > \theta_2$ faça $\lambda_3 = \lambda_2, \lambda_2 = \lambda_0, \theta_3 = \theta_2, \theta_2 = \theta_0$ e vá para P10.

Caso contrário faça $\lambda_1 = \lambda_0, \theta_1 = \theta_0$ e vá para P10.

P14. Se $\theta_2 > \epsilon'$ faça $\lambda' = \lambda_2$ e PARE.

Caso contrário vá para P15.

P15. Se $\epsilon_1 < \epsilon_2$ vá para P17.

Caso contrário vá para P16.

P16. Se $\theta_2 \leq \epsilon''$ faça $\epsilon = \beta \epsilon_2, \epsilon_1 = \beta \epsilon_1, \epsilon' = \epsilon'', \epsilon'' = \beta \epsilon''$ e vá para P10.

Caso contrário faça $\epsilon = \epsilon/2, \epsilon_1 = \epsilon_1/2, \epsilon' = \epsilon'/2$ e vá para P10.

P17. Se $\theta_2 > 0$ faça $\lambda' = \lambda_2$ e PARE.

Caso contrário a busca falhou, PARE.

CAPÍTULO IVMÉTODOS DE MINIMIZAÇÃO DESVINCULADA COM DERIVADAS

Os métodos estudados neste capítulo foram selecionados segundo a sua eficiência e levando-se em conta também a facilidade de operação oferecida ao usuário. Assim, o método de NEWTON (modificado de modo a assegurar convergência), de eficiência mais que reconhecida, não se encontra entre os algoritmos aqui apresentados uma vez que requer, em cada ponto, a determinação da matriz Hessiana da função: isso exigiria ao usuário um penoso trabalho de preparação dos dados mormente em problemas com elevado número de variáveis independentes. Em vez disto foram implementados os métodos de DAVIDON-FLETCHER-POWELL e de BROYDEN que requerem a disponibilidade apenas da derivada primeira da função, já que aproximam a inversa da matriz Hessiana por processos próprios.

O método de CAUCHY (steepest descent) foi incluído em face de seu efeito didático, pois é dos métodos mais antigos e de simples entendimento, e pelo bom comportamento que apresenta na resolução de um grande número de problemas.

O processo de FLETCHER-REEVES, que também faz parte deste trabalho, é um método tradicional e de grande eficiência, além de requerer pequena utilização de memória quando implementado em computador.

Nas seções seguintes são apresentados os métodos a que

acima nos referimos. Após a descrição de cada um comenta-se sua convergência.

De acordo com o modelo (II. 4.6), o algoritmo gera uma sequência de pontos (X_i) . Como, no caso geral, é impossível detetar a otimalidade de um ponto (ver [2]), um algoritmo será considerado convergente se:

- 1
 - a) a sequência (X_i) é finita e para seu último ponto \hat{X} é satisfeita a condição $\nabla f(\hat{X})=0$, ou
 - b) a sequência (X_i) é infinita e para qualquer um de seus pontos de acumulação \hat{X} é satisfeita a condição $\nabla f(\hat{X})=0$.

Algumas condições extras fornecem informações adicionais:

- 2
 - a) se a função objetivo é convexa e $\nabla f(\hat{X})=0$ então \hat{X} é um ponto de mínimo global.
 - b) se X_0 é o ponto inicial e o conjunto $C = \{X \in R^n / f(X) \leq f(X_0)\}$ for limitado então sempre haverá pontos de acumulação pois as buscas unidirecionais somente fornecem pontos em C e portanto as sequências geradas são compactas.

Na apresentação dos métodos a seguir, não se faz menção à manipulação de precisões. Supõem-se conhecidas as precisões iniciais utilizadas por cada algoritmo, segundo o tratamento específico desse assunto no capítulo VIII.

SEÇÃO 1 - MÉTODO DE CAUCHY

Este é dos mais antigos e dos mais simples processos de otimização, havendo sido introduzido pelo matemático francês A.L.CAUCHY em 1847, [9].

O método desenvolvido por CAUCHY utiliza informações da derivada primeira da função objetivo $f(\cdot)$ e se baseia no fato de que o gradiente calculado em qualquer ponto do domínio de $f(\cdot)$ aponta para a direção de máximo crescimento inicial da função. Caminhando-se, pois, na direção contrária à do gradiente estaremos na direção de máximo decréscimo inicial da função. Esta é a razão por que este método é mais conhecido por "steepest descent".

Vimos na seção 4 do capítulo II que, em relação ao modelo geral (II. 4.6), os algoritmos diferem entre si na determinação da direção $S \in R^n$ de busca. No método de CAUCHY a direção $S_i \in R^n$, no i -ésimo estágio do algoritmo é dada por

$$3 \quad S_i = -\nabla f(X_i)$$

e o novo ponto é obtido através da relação

$$4 \quad X_{i+1} = X_i + \lambda' S_i = X_i - \lambda' \nabla f(X_i)$$

onde λ' é o valor calculado pela busca unidirecional (ver III).

A relação (4) é a base do método de CAUCHY.

Demonstra-se em [2] que o método de CAUCHY é convergente, isto é, satisfaz às condições (1) para todas as buscas unidirecionais estudadas. A manipulação de precisões, nas buscas, não afeta a convergência do algoritmo, como se comentará no capítulo VIII.

5

ALGORITMO [2]

- P1 . Escolha $x_0 \in R^n$ como ponto inicial.
- P2 . Faça $i=0$.
- P3 . Calcule $\nabla f(x_i)$.
- P4 . Se $\nabla f(x_i)=0$, PARE. Caso contrário vá para P5.
- P5 . Faça $S_i = -\nabla f(x_i)$.
- P6 . Calcule λ'_i por meio de qualquer busca unidirecional (ver III).
- P7 . Faça $x_{i+1} = x_i + \lambda'_i S_i$, $i=i+1$ e vá para P3.

SEÇÃO 2 - MÉTODO DE FLETCHER-REEVES

Este método pertence à classe de direções conjugadas, sendo também conhecido por método de gradientes conjugados.

Embora o grau de convergência deste algoritmo seja inferior ao do método de NEWTON modificado, o fato de não requerer o cálculo de derivadas segundas e a inversão da matriz Hessiana, normalmente de considerável dimensão, faz com que a sua eficiência seja, na maioria das vezes, comparável à daquele método [2].

Os métodos de gradientes conjugados foram introduzidos inicialmente por HESTENES, STIEFEL e BECKMAN, [10], como processos de solução de sistemas de equações lineares. Eles possuem a interessante propriedade de minimizar uma função quadrática em, no máximo, n passos [1].

A idéia básica do método é ilustrada a seguir.

É gerada uma sequência de direções S_i que são combinações lineares entre $-\nabla f(X_i)$ e as direções anteriores de modo que, se a função objetivo for quadrática, então as direções geradas pelo algoritmo são conjugadas, [11].

Referindo-nos ao modelo em (II. 4.6), sejam $X_0 \in R^n$ o ponto inicial e $S_0 \in R^n$ a primeira direção de busca dada por

$$S_0 = -\nabla f(X_0)$$

Definamos as direções S_i , $i=1,2,\dots$ recursivamente por meio de

$$7 \quad S_{i+1} = -\nabla f(X_{i+1}) + w_i S_i$$

Demonstra-se [1] que se f for quadrática com Hessiana H definida positiva, então os valores $w_i \in \mathbb{R}$ podem se escolhidos de sorte a tornar S_0, S_1, \dots, S_{i+1} H -conjugadas. É possível demonstrar, [2], que os valores dos coeficientes w_i são dados por

$$8 \quad w_i = \frac{\langle \nabla f(X_{i+1}), \nabla f(X_{i+1}) \rangle}{\langle \nabla f(X_i), \nabla f(X_i) \rangle}$$

As relações (7) e (8) são a base do método de FLETCHER-REEVES.

Convergência para este método pode ser demonstrado para funções estritamente convexas e bidiferenciáveis, [2], desde que as buscas unidirecionais utilizadas realizem perfeita minimização em cada estágio do algoritmo.

Em nosso caso, para contornar o inevitável problema de minimizações imperfeitas efetuadas pelas buscas, o algoritmo é recomposto ("resetado") após cada conjunto de $2n$ iterações, ou após cada busca unidirecional com insucesso. Assim, se $i=2n$ então $S_{i+1} = -\nabla f(X_{i+1})$. Desta forma a convergência do algoritmo não é afetada pelo acúmulo de erros causado pelas minimizações imperfeitas das buscas, uma vez que o método de CAUCHY tem convergência demonstrada. O efeito dos erros sobre a rapidez de convergência é largamente compensado pelo aumento da rapidez das buscas unidirecionais, como se comentará em VIII.

ALGORITMO | 2 |

- P1 . Escolha $X_0 \in \mathbb{R}^n$. Se $\nabla f(X_0) = 0$, PARE:
Caso contrário vá para P2.
- P2 . Faça $i=0, k=2n$.
- P3 . Faça $g_i = S_i = -\nabla f(X_i)$.
- P4 . Calcule λ'_i através de qualquer busca unidirecional
(ver III).
- P5 . Faça $X_{i+1} = X_i + \lambda'_i S_i$.
- P6 . Calcule $\nabla f(X_{i+1})$.
- P7 . Se $\nabla f(X_{i+1}) = 0$, PARE.
Caso contrário vá para P8.
- P8 . Se $[(i+1)/k] = 1$ módulo k , faça $i=i+1$ e vá para P3.
Caso contrário vá para P9.
- P9 . Faça
- $$g_{i+1} = -\nabla f(X_{i+1})$$
- $$w_i = \frac{\langle g_{i+1}, g_{i+1} \rangle}{\langle g_i, g_i \rangle},$$
- $$S_{i+1} = g_{i+1} + w_i S_i,$$
- $i=i+1$ e vá para P4.

SEÇÃO 3 - MÉTODO DE DAVIDON-FLETCHER-POWELL

Este processo foi apresentado originalmente por DAVIDON, |12|, em 1959 e posteriormente modificado por FLETCHER e POWELL, |13|. Pertence à categoria de métrica variável, isto é, faz parte da classe de métodos que aproximam a inversa da matriz Hessiana da função, evitando, destarte, um considerável volume de cálculos que seriam aplicados na obtenção da Hessiana e em sua posterior inversão.

O método de DAVIDON-FLETCHER-POWELL apresenta muitas vantagens em relação a seus concorrentes como sejam, alta eficiência e boas propriedades de estabilidade computacional. Sua única desvantagem é a quantidade de memória necessária em computador para armazenar a aproximação da inversa da Hessiana que, de um modo geral, é de ordem elevada nos problemas reais, |2|.

O presente método, como de resto toda a família de métrica variável, possui a interessante propriedade de, em n passos, a matriz direcional tornar-se igual à inversa da Hessiana para funções quadráticas com Hessiana definida positiva.

A matriz direcional inicial é geralmente escolhida igual à matriz identidade, $E_0 = I$, embora possa ser qualquer matriz definida positiva. A cada passo vai se processando uma transformação gradual de direções de gradiente para direções de NEWTON extraíndo-se desse fato as boas

fases de comportamento daqueles dois métodos uma vez que o de CAUCHY tem boa atuação longe do ótimo enquanto que o método de NEWTON, modificado, apresenta boa performance em suas vizinhanças.

O método de DAVIDON-FLETCHER-POWELL pode ser também enquadrado na categoria dos que usam direções conjugadas. Para uma função objetivo qualquer é esse fato, mais que o de aproximar a inversa da matriz Hessiana, a razão maior de sua grande eficiência, |1|.

Com referência ao modelo em (II. 4.6), o presente método determina, em seu i -ésimo estágio, a direção de busca S_i por

$$10 \quad S_i = -E_i \nabla f(X_i)$$

onde E_i é a matriz direcional que substitui a inversa da Hessiana, $H^{-1}(X_i)$.

O novo ponto é então obtido por

$$11 \quad X_{i+1} = X_i + \lambda' S_i$$

onde λ' é calculado pela busca unidirecional (ver III).

A característica fundamental dos métodos de métrica variável é usar relações próprias para aproximar a inversa da Hessiana. A maneira como é feita esta aproximação determina essencialmente a diferença entre os diversos métodos |1|, |14| e |15|.

Para funções quadráticas entre dois estágios consecutivos, i e $i+1$, do algoritmo em estudo, é possível obter

a seguinte relação entre os respectivos pontos, (ver [1]):

$$12 \quad X_{i+1} - X_i = H^{-1}(X_i) [\nabla f(X_{i+1}) - \nabla f(X_i)],$$

onde H é uma matriz constante. A equação (12) pode ser encarada como um sistema de n equações lineares contendo um conjunto de parâmetros desconhecidos que devem ser estimados afim de se obter a aproximação da inversa de $H(X_i)$. Várias técnicas podem ser usadas para resolver o sistema acima e cada uma conduz a um diferente método de métrica variável.

Num grande grupo de métodos, $H^{-1}(X_{i+1})$ é aproximada usando informações do i -ésimo estágio:

$$13 \quad H^{-1}(X_{i+1}) \cong wE_{i+1} = w(E_i + \Delta E_i)$$

onde E_i é a matriz que aproxima $H^{-1}(X)$ e ΔE_i é u'a matriz a ser determinada e $w \in R$ é um fator de escala, uma constante, geralmente igual à unidade. Conforme já dissemos, a escolha de ΔE_i determina o tipo de método. Para garantir convergência, wE_{i+1} deve ser definida positiva e satisfazer à equação (12), quando substituí $H^{-1}(X_i)$.

No estágio $(i+1)$ temos os valores de $X_{i+1}, X_i, \nabla f(X_{i+1}), \nabla f(X_i)$ e E_i , e queremos calcular E_{i+1} tal que a relação abaixo, proveniente de (12), seja satisfeita.

$$14 \quad E_{i+1} \Delta g_i = \frac{1}{w} \Delta X_i$$

onde $\Delta g_i = \nabla f(X_{i+1}) - \nabla f(X_i)$

$$\Delta X_i = X_{i+1} - X_i$$

Seja $\Delta E_i = E_{i+1} - E_i$. A equação

15
$$\Delta E_i \Delta g_i = \frac{1}{w} \Delta X_i - E_i \Delta g_i,$$
 deve ser resolvida em relação a ΔE_i . Pode-se mostrar, por substituição direta do resultado, que a equação (15) tem a solução

16
$$\Delta E_i = \frac{1}{w} \cdot \frac{\Delta X_i Y'}{\langle Y, \Delta g_i \rangle} - \frac{E_i \Delta g_i Z'}{\langle Z, \Delta g_i \rangle}$$

onde $Y, Z \in \mathbb{R}^n$ são vetores arbitrários.

Os tipos de métodos variam conforme a escolha de Y e Z .

Se, para $w=1$, fizermos $Y=\Delta X_i$ e $Z=E_i \Delta g_i$, teremos o método de DAVIDON-FLETCHER-POWELL e a equação (16) se torna:

17
$$\Delta E_i = \frac{\Delta X_i \Delta X_i'}{\langle \Delta X_i, \Delta g_i \rangle} - \frac{(E_i \Delta g_i) (E_i \Delta g_i)'}{\langle E_i \Delta g_i, \Delta g_i \rangle},$$

e a atualização da matriz direcional é dada por

18
$$E_{i+1} = E_i + \Delta E_i$$

onde ΔE_i é a relação (17).

Convergência para o algoritmo em estudo é garantida para funções objetivo quadráticas com a matriz Hessiana definida positiva, [1].

Mais recentemente, POWELL, [16], obteve prova de convergência deste método para funções não necessariamente quadráticas porem estritamente convexas, [2].

Em face de minimizações imperfeitas efetuadas

pelas buscas, pode ocorrer que em um determinado estágio a busca não consiga achar um valor adequado (ver III) para λ' . Neste caso o algoritmo é recomposto fazendo-se $E(X_i) = I$. Este procedimento evita que a convergência do método seja afetada por acúmulo de erros devido a minimizações imprecisas das buscas.

P1 . Escolha $X_0 \in \mathbb{R}^n$. Se $\nabla f(X_0) = 0$, PARE.

Caso contrário vá para P2.

P2 . Faça $i=0$, $E_i = I$ (matriz identidade) e $g_0 = \nabla f(X_0)$.

P3 . Faça $S_i = -E_i g_i$.

P4 . Calcule λ'_i por qualquer busca unidirecional (ver III).

P5 . Calcule $\nabla f(X_i + \lambda'_i S_i)$.

P6 . Se $\nabla f(X_i + \lambda'_i S_i) = 0$, PARE.

Caso contrário faça

$$X_{i+1} = X_i + \lambda'_i S_i$$

$$g_{i+1} = \nabla f(X_{i+1})$$

$$\Delta g_i = g_{i+1} - g_i$$

$$\Delta X_i = X_{i+1} - X_i$$

$$E_{i+1} = E_i + \frac{\Delta X_i \Delta X_i'}{\langle \Delta X_i, \Delta g_i \rangle} - \frac{(E_i \Delta g_i) (E_i \Delta g_i)'}{\langle E_i \Delta g_i, \Delta g_i \rangle}$$

e vá para P7.

P7 . Faça $i=i+1$ e vá para P3.

SEÇÃO 4 - MÉTODO DE BROYDEN

O método de BROYDEN, [17], publicado em 1967 pertence também à classe de métrica variável como o processo de DAVIDON-FLETCHER-POWELL, discutido na seção anterior. A diferença entre eles reside no processo de geração da matriz direcional (ver IV. 3.10 a 18).

Em um estágio i do algoritmo em pauta, a parcela ΔE_i (ver 17) de atualização da matriz direcional E_i é dada por

$$20 \quad \Delta E_i = \frac{(\Delta X_{i-E_i} \Delta g_i) (\Delta X_{i-E_i} \Delta g_i)'}{\langle \Delta X_{i-E_i} \Delta g_i, \Delta g_i \rangle}$$

onde $\Delta X_i = X_{i+1} - X_i$ e $\Delta g_i = \nabla f(X_{i+1}) - \nabla f(X_i)$

A nova matriz direcional E_{i+1} e, então calculada por

$$21 \quad E_{i+1} = E_i + \Delta E_i$$

Como na seção 3, a convergência para o método de BROYDEN é demonstrada apenas para funções quadráticas com Hessiana definida positiva.

Se a função objetivo não é quadrática, pode ocorrer que

1 - a matriz direcional pode deixar de ser definida positiva.

2 - a parcela de correção ΔE_i pode tornar-se limitada (às vezes até mesmo para funções qua

dráticas) devido a erros de aproximação.

- 3 - Se $\Delta X_i = -\lambda_i^{-1} E_i \nabla f(X_i)$ tiver, por coincidência, a mesma direção do estágio anterior, E_{i+1} torna-se singular ou indeterminada.

Assim, no algoritmo de BROYDEN, se ocorrer pelo menos um dos dois casos

$$1 - E_i \Delta g_i = \Delta X_i,$$

$$2 - \langle (E_i \Delta g_i - \Delta X_i), \Delta g_i \rangle = 0,$$

deve-se fazer $E_{i+1} = E_i$, isto é, $\Delta E_i = 0$.

Com estas precauções a convergência não é destruída pela manipulação de precisões nas buscas unidirecionais, embora a rapidez de convergência possa ser afetada (ver VIII).

ALGORITMO |1|

P1 . Escolha $X_0 \in R^n$. Se $\nabla f(X_0)=0$, PARE.

Caso contrário vá para P2.

P2 . Faça $i=0$, $E_i=I$ (matriz identidade) e $g_0=\nabla f(X_0)$.

P3 . Faça $S_i=-E_i g_i$.

P4 . Calcule λ'_i por qualquer busca unidirecional (ver III).

P5 . Calcule $\nabla f(X_i+\lambda'_i S_i)$.

P6 . Se $\nabla f(X_i+\lambda'_i S_i)=0$, PARE.

Caso contrário, faça

$$X_{i+1} = X_i + \lambda'_i S_i$$

$$g_{i+1} = \nabla f(X_{i+1})$$

$$\Delta g_i = g_{i+1} - g_i$$

$$\Delta X_i = X_{i+1} - X_i$$

$$E_{i+1} = E_i + \frac{(\Delta X_i - E_i \Delta g_i) (\Delta X_i - E_i \Delta g_i)'}{\langle \Delta X_i - E_i \Delta g_i, \Delta g_i \rangle}$$

e vá para P7.

P7 . Faça $i=i+1$ e vá para P3.

CAPÍTULO VMÉTODOS DE MINIMIZAÇÃO DESVINCULADA SEM DERIVADAS

Os métodos de minimização que não requerem derivadas são, muitas vezes, preferidos em relação àqueles que as usam. Evidentemente existem casos em que se justifica tal preferência e, como principais, podemos citar os seguintes:

1 - A expressão analítica da função objetivo não é conhecida explicitamente. Em muitos casos o valor da função objetivo pode somente ser calculado ponto a ponto.

2 - A expressão analítica de f é conhecida mas o cálculo do gradiente é altamente trabalhoso em face da complexidade de f .

3 - Facilidade de preparação das informações iniciais de corrente da não utilização de gradientes.

Embora, de um modo geral, a eficiência desses métodos seja inferior à dos que usam gradientes, o desempenho de alguns algoritmos que não usam derivadas pode ser considerado excelente e mesmo superior ao de vários daqueles que delas fazem uso (ver |1|).

No presente capítulo apresentaremos três métodos selecionados segundo seu desempenho: os processos de POWELL, de NELDER-MEAD e de CAUCHY com redução do cálculo de derivadas.

A utilização de tais métodos será exposta no capítulo X e as listagens se encontram em XII.

SEÇÃO 1 - MÉTODO DE CAUCHY

Este é um processo de minimização desvinculada sem derivadas desenvolvido por POLAK, [2]. O algoritmo efetua um cálculo aproximado do gradiente da função usando vetores canônicos $e_i \in R^n$, $i=1,2,\dots,n$. A direção de busca é a aproximação do gradiente, com sinal trocado.

A categoria dos processos que resolvem o problema

$$1 \quad \text{Min } f(X), X \in R^n$$

sem usar derivadas é constituída basicamente de dois tipos: os que derivam de métodos que utilizam gradientes, aproximando-os através de diferenças finitas e aqueles cujo desenvolvimento conceitual independe do cálculo de derivadas. O presente algoritmo pertence ao primeiro grupo.

Em (1), $f: R^n \rightarrow R$ deve ser, pelo menos, continuamente diferenciável.

A convergência do algoritmo que apresentaremos a seguir é tratada em [2].

2

ALGORITMO | 2 |

P1 . Escolha $X_0 \in R^n$, $\beta \in [5, 10]$, $\epsilon_1 > 0$ (ver VIII).

P2 . Faça $i=0$

P3 . Faça $\epsilon = \epsilon_1 / \beta$

P4 . Calcule o vetor $S(\epsilon, X_i) \in R^n$ cuja j -ésima componente , $S_j(\epsilon, X_i)$ é definida por

$$S_j(\epsilon, X_i) = - \frac{1}{\epsilon} \left[f(X_i + \epsilon e_j) - f(X_i) \right], \quad j=1, 2, \dots, n$$

onde e_j é a j -ésima coluna da matriz identidade $(n \times n)$, isto é, $e_1 = (1, 0, \dots, 0)$, $e_2 = (0, 1, 0, \dots, 0)$, etc.

P5 . Calcule $f(X_i + \beta \epsilon S(\epsilon, X_i)) / \|S\| - f(X_i) \triangleq \Delta(\epsilon, X_i)$.

P6 . Se $\Delta(\epsilon, X_i) \geq 0$ faça $\epsilon = \epsilon / 2$ e vá para P4.

Caso contrário calcule λ_i' por qualquer busca unidirecional*.

P7 . Calcule $\theta(\lambda_i', X_i, S(\epsilon, X_i)) = f(X_i + \lambda_i' S(\epsilon, X_i)) - f(X_i)$.

P8 . Se $\theta(\lambda_i', X_i, S(\epsilon, X_i)) \leq -\|S\|\epsilon$ faça $X_{i+1} = X_i + \lambda_i' S(\epsilon, X_i)$, $i=i+1$ e vá para P4.

Caso contrário faça $\epsilon = \epsilon / 2$ e vá para P4.

*Nas buscas de GOLDSTEIN (III. 1) e ARMIJO (III. 2) o produto escalar $\langle \nabla f(X), S \rangle$ é aproximado, fazendo-se
 $\langle \nabla f(X), S \rangle \cong \Delta(\epsilon, X_i) / \epsilon$.

SEÇÃO 2 - MÉTODO DE POWELL

Este é um método sem derivadas que atinge o mínimo de uma função quadrática com Hessiana definida positiva, em no máximo n passos, através de sucessivas buscas unidirecionais ao longo de uma série de direções conjugadas partindo de um ponto inicial X_0 .

Sabemos que duas direções S_i e S_j são conjugadas se:

$$\langle S_i, QS_j \rangle = 0, \quad i \neq j \quad \text{e}$$

$$\langle S_i, QS_j \rangle \geq 0, \quad i = j,$$

onde Q é uma matriz quadrada definida positiva.

O método de POWELL se baseia no seguinte fato:

Para uma função quadrática com Hessiana definida positiva, se partirmos de um ponto X_0 e determinarmos X_1 após minimizações consecutivas em $p < n$ direções conjugadas e fizermos o mesmo, a partir de X_1 para determinar X_j , então a direção $X_j - X_1$ é conjugada em relação às p direções usadas para obter tanto X_1 como X_j , [18].

O modelo abaixo (ver [18]) dá uma boa idéia do funcionamento do método.

MODELO

- P1 . Escolha um ponto inicial X_0 .
- P2 . Faça $S_i = e_i$, onde e_i é o i -ésimo vetor canônico.
- P3 . Encontre $\lambda_i \in \mathbb{R}$ tal que $f(X_{i-1} + \lambda_i S_i)$ é mínimo.
Defina $X_i = X_{i-1} + \lambda_i S_i$, $i=1, 2, \dots, n$.
- P4 . Gere uma nova direção $S = X_n - X_0$ e faça
 $S_1 = S_2, S_2 = S_3, \dots, S_{n-1} = S_n, S_n = S$.
- P5 . Minimize $f(X_n + \lambda S_n)$ para determinar $X = X_n + \lambda S_n$.
Faça $X_0 = X$ e vá para P3 (X_0 é o novo ponto de partida).

O algoritmo (4) difere, no entanto, do modelo acima quanto à direção a ser substituída, em face do problema de convergência. Para maiores detalhes veja-se [1] e [20].

Convergência para o presente método é demonstrada em [20] para funções quadráticas onde também é apresentada uma modificação do algoritmo e provada sua convergência para funções estritamente convexas e continuamente diferenciáveis.

ALGORITMO | 1 |

Sejam $D=(S_1, S_2, \dots, S_n)$ u'a matriz $n \times n$, $S_i \in R^n$, e $\epsilon_1 \in R$ (ver VIII).

P1 . Escolha um ponto inicial $X_0 \in R^n$.

P2 . Faça $S_i = e_i$, onde $e_i = (0, \dots, 0, 1, 0, \dots, 0)$, $i=1, 2, \dots, n$, $e_i \in R^n$.

P3 . Faça $i=1$.

P4 . Determine λ_i' minimizando $f(X_{i-1} + \lambda_i S_i)$.

P5 . Faça $X_i = X_{i-1} + \lambda_i' S_i$

P6 . Se $i < n$ faça $i=i+1$ e vá para P4. Caso contrário vá para P7.

P7 . Calcule $X_{n+1} = 2X_n - X_0$.

P8 . Calcule $\delta = \max\{f(X_{i-1}) - f(X_i)\}$, $i=1, 2, \dots, n$ e chame de S_m a direção correspondente a δ .

P9 . Se, $f(X_{n+1}) < f(X_0)$ ou

$$\left[f(X_0) - 2f(X_n) + f(X_{n+1}) \right] \left[f(X_0) - f(X_n) - \delta \right]^2 < \frac{\delta}{2} \left[f(X_0) - f(X_{n+1}) \right]^2$$

vá para P12. Caso contrário vá para P10.

P10. Mantenha os mesmos valores de S_i , $i=1, 2, \dots, n$ para o próximo estágio.

P11. Se $f(X_n) \geq f(X_{n+1})$ faça $X_0 = X_{n+1}$ e vá para P14.

Caso contrário faça $X_0 = X_n$ e vá para P14.

- P12. Minimize $f(X_0 + \lambda S)$, onde $S = X_n - X_0$, e faça $X_0 = X_0 + \lambda' S$, onde λ' é o valor ótimo de λ na direção S .
- P13. Substitua S_m por S na matriz D mas fazendo de S sempre a última coluna: $D = (S_1, S_2, \dots, S_{m-1}, S_{m+1}, \dots, S_n, S)$.
- P14. Se $\|X_n - X_0\| \leq \epsilon_1$, PARE. Caso contrário vá para P3 para iniciar um novo estágio.

SEÇÃO 3 - MÉTODO DE NELDER-MEAD

O método que estudaremos a seguir segue uma teoria totalmente independente do que vimos até aqui e foi desenvolvido por NELDER-MEAD, [21], baseados em trabalho anterior de SPENDLEY, HEXT e HIMSWORTH datado de 1962 (ver [1]).

É um algoritmo que requer apenas a expressão analítica da função objetivo a ser minimizada e apresenta boa eficiência, considerando-se que não utiliza derivadas, além de ser facilmente implementável em computador, [1].

Apresentamos a seguir uma ideia geral de funcionamento do método.

Recordemos que um poliedro regular de $n+1$ vértices em R^n é um simplex. Por exemplo em R^2 o simplex regular é representado por um triângulo equilátero, em R^3 por um tetraedro regular, etc.

Dados $n+1$ pontos em R^n formando um simplex a função objetivo pode ser avaliada em cada um dos vértices, e daquele correspondente ao maior valor da função é feita uma reflexão através do centróide do simplex. O vértice que originou a reflexão pode ser substituído pelo novo ponto e um outro simplex obtido. Assim procedendo, sempre substituindo ou não o vértice que der origem ao maior valor da função pelo ponto obtido na reflexão, juntamente com processos adequados de redução gradativa do simplex e de evitar ciclagem nas vizinhanças do ponto de mínimo, tem-se um método de mini

mização sem derivadas de desempenho apenas razoável.

Algumas dificuldades de ordem prática, no processo acima descrito, como sejam a impossibilidade de acelerar a pesquisa do mínimo ou mesmo de continuá-la em certos casos motivaram a adoção de várias medidas destinadas a melhorar a atuação do algoritmo.

O trabalho de NELDER-MEAD foi exatamente o de introduzir tais melhoramentos. A alteração básica no processo foi a possibilidade de o simplex, durante a execução, sofrer variações em sua forma deixando, assim, de ser um simplex regular. Daí a origem da denominação mais sugestiva - POLIEDRO FLEXIVEL - pela qual é também o método conhecido.

O algoritmo de NELDER-MEAD minimiza uma função $f: R^n \rightarrow R$ usando $(n+1)$ vértices de um poliedro flexível, em R^n . Cada vértice pode ser definido por um vetor X . Aquele correspondente ao maior valor de $f(X)$ é projetado através do centróide dos vértices RESTANTES (e não do poliedro, como antes), originando um novo ponto em que $f(X)$ pode ou não ter um valor menor.

Sejam, em um estágio qualquer do algoritmo, X_i o i -ésimo vértice do poliedro, onde $X_i \in R^n$, $i=1,2,\dots,n+1$, e o valor da função em X_i igual a $f(X_i)$. Sejam ainda X_h e X_ℓ vértices do poliedro tais que

$$f(X_h) = \max\{f(X_1), \dots, f(X_{n+1})\} \text{ e}$$

$$f(X_\ell) = \min\{f(X_1), \dots, f(X_{n+1})\}, \text{ e}$$

X_{n+2} o centróide de $\{X_1, X_2, \dots, X_{n+1}\} = \{X_h\}$ calculável por:

$$5 \quad \bar{x}_{n+2,j} = \frac{1}{n} \left[\left(\sum_{i=1}^{n+1} \bar{x}_{ij} \right) - x_{hj} \right], \quad j=1, 2, \dots, n$$

Escolhe-se, em geral, como poliedro inicial um simplex regular mas não é obrigatório. A sequência de passos para se encontrar um ponto $X \in R^n$ onde $f(X)$ assuma um valor menor envolve, em linhas gerais, as seguintes operações:

1 - REFLEXÃO - X_h é refletido através do centróide obtendo-se o ponto X_{n+3} calculável por

$$6 \quad X_{n+3} = X_{n+2} + \alpha(X_{n+2} - X_h)$$

onde $\alpha > 0$ é o coeficiente de reflexão

X_{n+2} é o centróide calculado por (5) e

X_h é o vértice onde $f(X_i)$, $i=1, 2, \dots, n+1$, é máximo.

2 - EXPANSÃO - Se $f(X_{n+3}) \leq f(X_\ell)$, o vetor $(X_{n+3} - X_{n+2})$ é expandido, gerando X_{n+4} calculável por:

$$7 \quad X_{n+4} = X_{n+2} + \gamma(X_{n+3} - X_{n+2})$$

onde $\gamma > 1$ é o coeficiente de expansão.

Se $f(X_{n+4}) < f(X_\ell)$, X_h é substituído por X_{n+4} e a operação recomeça de 1, como um novo estágio. Caso contrário X_h é substituído por X_{n+3} voltando-se também para 1.

3 - CONTRAÇÃO - Se $f(X_{n+3}) > f(X_i)$, $\forall i \neq h$, é feita uma contração do vetor $(X_h - X_{n+2})$ gerando X_{n+5} calculável por:

$$8 \quad X_{n+5} = X_{n+2} + \beta (X_h - X_{n+2})$$

onde $0 < \beta < 1$ é o coeficiente de contração

X_h é substituído por X_{n+5} e o retorno é feito para 1 iniciando-se um novo estágio.

4 - REDUÇÃO - Se $f(X_{n+3}) > f(X_h)$ todos os vetores $(X_i - X_\ell)$, $i=1, 2, \dots, n+1$, são reduzidas à metade a partir de X_ℓ , o que é feito através da equação

$$X_i = X_\ell + \frac{1}{2} (X_i - X_\ell), \quad i=1, 2, \dots, n+1$$

Volta-se novamente para 1 afim de iniciar-se o estágio seguinte.

A diferença básica entre o simplex rígido e o poliedro flexível é que este possui a faculdade de ser auto adaptativo à topografia da função objetivo, alongando-se, contraindo-se ou reduzindo-se de tamanho, conforme a necessidade do problema.

Conforme visto, o coeficiente α é usado na reflexão, γ na expansão e β na contração do poliedro flexível.

Uma questão fundamental é pois estabelecer os valores de α , β e γ mais eficazes. Note-se que uma alteração do poliedro só é necessária quando ocorrer uma variação na topo

grafia da função. Assim, com $\alpha=1$ estaremos atendendo a essa invariância de forma do poliedro. Além disto, NELDER-MEAD demonstraram que para $\alpha=1$ um número muito menor de avaliações de $f(X)$ é requerido do que com $\alpha=1$.

- 1 - Um menor valor para 2 proporciona uma melhor adaptação do poliedro ao "terreno" da função objetivo, particularmente nos casos de estreitamento com mudanças de direção.
- 2 - Nas vizinhanças do mínimo o poliedro precisa ser reduzido e um valor grande para retarda a convergência. Destarte, $\alpha=1$ parece ser uma boa escolha.

Quanto a β e γ , NELDER-MEAD baseados em diversos testes com diferentes combinações entre os dois parâmetros chegaram à conclusão que $\beta=0.5$ e $\gamma=2$ constituem duas boas opções. Já PAVIANI sugeriu os seguintes valores para β e γ , (ver [1]),

$$0.4 \leq \beta \leq 0.6,$$

$$2.8 \leq \gamma \leq 3.0,$$

considerando que para $0 < \beta < 0.4$ existe a possibilidade de término antecipado e com $\beta > 0.6$ o algoritmo pode requerer um excessivo número de estágios e longo tempo de computador para atingir a solução do problema.

O método de NELDER-MEAD apresenta duas particularidades interessantes: não requer derivadas nem utiliza buscas unidirecionais.

Devemos acrescentar ainda que este processo é a base do método de Tolerância Flexível apresentado no capítulo VII e que a forma de geração dos vértices do poliedro regular inicial, a partir de um ponto dado, é ali também explicado.

ALGORITMO | 1 |

Sejam $\alpha=1$, $\beta=0.5$, $\gamma=2e$ e $\epsilon_2 \in R$ (ver VIII).

- P1. Escolha um ponto inicial $X_1 \in R^n$.
- P2. Calcule, a partir de X_1 , os demais vértices do poliedro inicial (ver VII. 1).
- P3. Calcule $f(X_i)$, $i=1,2,\dots,n+1$
- P4. Calcule X_h , X_ℓ e X_{n+2} .
- P5. REFLEXÃO - Calcule X_{n+3} pela equação (6) e $f(X_{n+3})$.
- P6. Se $f(X_{n+3}) > f(X_\ell)$ vá para P9. Caso contrário, vá para P7.
- P7. EXPANSÃO - Calcule X_{n+4} pela equação (7) e $f(X_{n+4})$.
- P8. Se $f(X_{n+4}) < f(X_\ell)$ faça $X_h = X_{n+4}$ e vá para P14.
Caso contrário, faça $X_h = X_{n+3}$ e vá para P14.
- P9. Se $f(X_{n+3}) > f(X_i)$, $\forall i \neq h$ vá para P10.
Caso contrário faça $X_h = X_{n+3}$ e vá para P14.
- P10. Se $f(X_{n+3}) > f(X_h)$ vá para P11. Caso contrário faça $X_h = X_{n+3}$ e vá para P11.
- P11. CONTRAÇÃO - Calcule X_{n+5} pela equação (8) e $f(X_{n+5})$.
- P12. Se $f(X_{n+5}) < f(X_h)$ faça $X_h = X_{n+5}$ e vá para P14.
Caso contrário vá para P13.
- P13. REDUÇÃO - Calcule X_i , $i=1,2,\dots,n+1$, pela equação (9).
- P14. Se $\left\{ \frac{1}{n} \sum [f(X_i) - f(X_{n+2})]^2 \right\}^{1/2} \leq \epsilon_2$ PARE.
Caso contrário vá para P4.

CAPÍTULO VIMÉTODOS DE MINIMIZAÇÃO VINCULADA COM DERIVADAS

No segundo capítulo formalizamos o problema de minimização vinculada (ver II. 1) e fizemos referência à larga faixa de aplicação que ocupa no campo da otimização.

Existem na literatura, vários métodos destinados a resolver o problema (II. 1.1), uns mais outros menos eficientes. Em nosso trabalho encontram-se três deles, dos quais dois exigem o cômputo da derivada da função objetivo e dos vínculos, ao passo que o terceiro necessita tão somente do cômputo dessas funções ponto a ponto.

A menos de poucas exceções os métodos existentes se dividem em dois grandes grupos: O primeiro formado por aqueles cujo desenvolvimento teórico independe do uso de derivadas, utilizando-as apenas como uma ferramenta de cálculo. O segundo constituído por aqueles que dependem conceitualmente de gradientes. A este pertence o método de direções viáveis apresentado na seção 2 e àquele o método de penalidades descrito na seção 1.

O método de penalidades resolve o problema geral de otimização (II. 1.1) e apresenta, de um modo geral, boa rapidez de convergência.

A restrição que se apresenta para o método de direções viáveis é não admitir a existência de vínculos não lineares do tipo igualdade. Como o método de penalidades, apresenta boa convergência.

Na descrição dos métodos, neste capítulo, basear-nos-emos sempre em |2|, |1|, |23| e |24|.

SEÇÃO 1 - MÉTODO DE PENALIDADES

O problema geral de otimização é

$$1 \quad (P) \quad \text{Minimizar } f(X)$$

$$X \in V$$

onde V é o conjunto viável definido por

$$2 \quad V = \{X \in \mathbb{R}^n / g(X) \leq 0, \quad h(X) = 0\}$$

onde $g: \mathbb{R}^n \rightarrow \mathbb{R}^m$

$$h: \mathbb{R}^n \rightarrow \mathbb{R}^{\ell}$$

A idéia fundamental do método de penalidades é reduzir a resolução do problema (1) a uma sequência de problemas de minimização desvinculada da forma

$$3 \quad (P_i) \quad \text{Min}\{f(X) + p_i(X) / X \in \mathbb{R}^n\}, \quad i=1,2,\dots$$

onde $p_i: \mathbb{R}^n \rightarrow \mathbb{R}$, $i=1,2,\dots$ são funções penalidades que adicionam a $f(X)$ um custo positivo se $X \notin V$ forçando as soluções dos problemas (P_i) a se aproximarem de V (caso de penalidades exteriores), ou forçam essas soluções a permanecerem no interior de V (caso de penalidades interiores).

O problema (3) pode ser construído basicamente de três maneiras, a depender da natureza das funções p_i . Estas podem ser exteriores ou interiores. No primeiro caso o processo de solução do problema (3) é dito de penalidades exteriores e no segundo, de penalidades interiores ou de barreira. Quando se usam os dois tipos de penalidades o processo

recebe a denominação de método misto.

Em nosso trabalho optamos pela implementação do método misto, que reúne as vantagens de ambos os métodos.

A seguir descreveremos, sucintamente, as noções fundamentais dos métodos de penalidades exteriores e interiores para, então, abordarmos o método misto.

MÉTODO DE PENALIDADES EXTERIORES

- 4 DEFINIÇÃO: Uma seqüência de funções contínuas -
 $p_i: R^n \rightarrow R, i=1,2,\dots$, é dita de penalidades exteriores para o problema (P_i) se:
- $p_i(X) = 0, \forall X \in V, i=1,2,\dots$,
 - $p_i(X) > 0, \forall X \notin V, i=1,2,\dots$,
 - $p_{i+1}(X) > p_i(X), \forall X \notin V, i=1,2,\dots$,
 - $p_i(X) \rightarrow \infty$ quando $i \rightarrow \infty, \forall X \notin V$

Seja $p_i(\cdot), i=1,2,\dots$ uma seqüência de funções penalidades exteriores para o conjunto V definido em (2). Uma seqüência de problemas de minimização desvinculada, penalizados, é definida por

$$5 \quad (PE)_i \quad \text{Min}\{f(X)+p_i(X)/X \in R^n\}$$

Para demonstrar a convergência do método é necessário garantir que os problemas penalizados $(PE)_i$ têm solução. Segue uma hipótese bastante restritiva enunciada em [2]. Uma hipótese menos restritiva encontra-se em [23].

6 HIPÓTESE

- V é fechado
- Existe $X' \in V$ tal que o conjunto
 $U = \{X/f(X) \leq f(X')\}$ é compacto.

7 Consideremos a seqüência de problemas definida em (5) e suponhamos que a hipótese em (6) é satisfeita.

Demonstra-se (ver |2|) que, para f contínua:

Se x_i é ótimo para $(PE)_i$, $i=1,2,\dots$, então a sequência $(x_i)_{i \in \mathbb{N}}$ é compacta e qualquer ponto de acumulação de (x_i) é ótimo para o problema (P).

- 8 FUNÇÕES PENALIDADES EXTERIORES - Existem várias maneiras de se determinarem as funções $p_i(\cdot)$ para o conjunto viável (2). Como exemplo temos a seguinte:

$$9 \quad p_i(x) = \alpha_i \left[\left(\sum_{i=1}^{\ell} (h_i(x))^2 \right)^{\beta/2} + f(x) + \sum_{i=1}^m (\max\{g_i(x), 0\})^{\beta} \right]$$

onde $\beta \geq 1$ e α_i é uma sequência estritamente crescente de números positivos tal que $\alpha_i \rightarrow \infty$ quando $i \rightarrow \infty$.

Desde que sejam h_i e g_i funções continuamente diferenciáveis, então p_i também o serão se $\beta \geq 2, |2|$.

MÉTODO DE PENALIDADES INTERIORES

Neste caso, o problema geral de otimização (1) é transformado em uma sequência de problemas de minimização desvinculada à semelhança do que se fez em penalidades exteriores, diferindo apenas quanto à natureza das penalidades que são agora interiores.

Suponhamos que o conjunto viável $V \subset \mathbb{R}^n$ seja definido por

$$10 \quad V = \{X/g_i(X) \leq 0, i=1,2,\dots,m\}.$$

11 HIPÓTESE

a. V é fechado

b. $\bar{V} = V \neq \emptyset$, ou seja, a aderência do interior de V é igual a V e não vazia.

12 DEFINIÇÃO - Uma sequência de funções contínuas -

$p_j^!: \overset{0}{V} \rightarrow \mathbb{R}, j=1,2,\dots$ é dita uma sequência de funções penalidades interiores para V se

a. $p_j^!(X) \rightarrow 0$ quando $j \rightarrow \infty, \forall X \in \overset{0}{V}$

b. Seja (X_i) uma sequência qualquer convergente para um ponto \hat{X} na fronteira de V . Nesse caso,

$$\lim_{i \rightarrow \infty} p_j^!(X_i) = +\infty, \text{ para } j=1,2,\dots$$

Consideremos agora a sequência de problemas de minimização

$$13 \quad (PI)_j \quad \text{Min}\{f(X) + p_j^!(X) / X \in \overset{\circ}{V}\}, \quad j=1,2,\dots$$

onde $p_j^!(.)$ são funções penalidades interiores. Hipótese semelhante a (6) garante que os problemas $(PI)_j$ têm solução em $\overset{\circ}{V}$ (ver [2]). Nesse caso mostra-se, [2], que qualquer ponto de acumulação da sequência de soluções (X_i) dos problemas $(PI)_j$ resolve o problema (P).

14 FUNÇÕES PENALIDADES INTERIORES - As funções $p_j^!(.)$ podem ser definidas conforme abaixo:

$$15 \quad p_j^!(X) = -\alpha_j \sum_{i=1}^m \frac{1}{g_i(X)}, \quad X \in \overset{\circ}{V}, \quad j=1,2,\dots$$

onde $\alpha_j, j=1,2,\dots$ é uma sequência estritamente decrescente de números positivos que tende para zero quando j tende para infinito.

RESOLUÇÃO DO SUBPROBLEMA - Tanto os problemas $(PE)_i$ quanto os $(PI)_j$ podem ser resolvidos por qualquer método desvinculado. Para os problemas $(PI)_j$, no entanto, como suas soluções estão em $\overset{\circ}{V}$, deve-se tomar o cuidado de não sair de $\overset{\circ}{V}$ durante as buscas unidirecionais. Uma técnica para evitar a obtenção de um ponto fora de $\overset{\circ}{V}$ é descrita no capítulo VIII.

MÉTODO MISTO

Este processo é uma combinação dos dois métodos vistos anteriormente e a sequência de problemas desvinculados em que é transformado o problema (1) utiliza ambas as formas de penalidades.

Seja $V=V' \cap V''$, onde V' satisfaz as condições (6) e V'' as condições (11). Utilizaremos penalidades exteriores com relação a V' e interiores com relação a V'' .

16 HIPÓTESE - Para ao menos um $\hat{X} \in V$, ótimo para o problema (P) qualquer vizinhança aberta $\hat{\beta}$ de \hat{X} satisfaz -
 $\hat{B} \cap V' \cap \overset{0}{V}'' \neq \emptyset$.

17 DEFINIÇÃO - Seja a sequência de problemas de minimização desvinculada definida por

$$(P)_i \quad \text{Min}\{f(X) + p_i(X) + p'_i(X) / X \in \overset{0}{V}''\}, \quad i=1, 2, \dots$$

onde $p_i(\cdot)$ são penalidades exteriores e $p'_i(\cdot)$ penalidades interiores respectivamente para os conjuntos V' e V'' .

Com as hipóteses feitas acima com relação aos conjuntos viáveis e funções penalidades, demonstra-se, |2|, que a sequência gerada pelos problemas $(P)_i$ satisfaz as condições de convergência como em (7).

O método misto é o que apresenta melhor comportamento na solução de problemas práticos, |2|, daí haver

seu funcionamento, em linhas gerais, é o seguinte:

Dado um ponto inicial $X_0 \in R^n$ para o problema (1) e V definido como

$$V = \{X \in R^n / g_i(X) \leq 0, i=1,2,\dots,m, h_j(X)=0, j=1,2,\dots,\ell\}$$

onde $g_i: R^n \rightarrow R$ e $h_j: R^n \rightarrow R$, podemos aplicar penalidades exteriores aos vínculos $h_j(X)=0$ e às restrições $g_i(X) \leq 0$, sempre que $g_i(X_0) \geq 0$ e penalidades interiores aos vínculos $g_i(X) \leq 0$ quando $g_i(X_0) < 0$.

Na aplicação do método é necessário que todas as funções envolvidas sejam continuamente diferenciáveis.

A convergência do algoritmo é tratada em [2].

Na resolução do subproblema as minimizações são truncadas em conformidade com o trabalho realizado por E.POLAK (ver [2]). A truncagem é feita com base no gradiente da função objetivo dos problemas desvinculados, aumentando-se a precisão das buscas à medida que crescem as penalidades.

ALGORITMO | 2 |

P1 . Escolha $X_0 \in R^n$, $\alpha, \alpha', \alpha'' \in (0, 0.5)$ e $\beta \in (0.5, 0.8)$

P2 . Faça $X = X_0$ e $k=0$.

P3 . Defina os conjuntos de índices

$$I = \{i \in \{1, 2, \dots, m\} / g_i(X) \geq 0\},$$

$$I' = \{i \in \{1, 2, \dots, m\} / g_i(X) < 0\}.$$

P4 . Defina as funções penalidades exteriores e interiores por

$$p(X) = \sum_{i=1}^{\ell} [h_i(X)]^2 + \sum_{i \in I} [\max\{0, g_i(X)\}]^2,$$

$$p'(X) = - \sum_{i \in I'} \frac{1}{g_i(X)}.$$

P5 . Se $k=0$, vá para P6. Caso contrário vá para P9.

P6 . Calcule $\nabla f(X)$, $\nabla p(X)$, $\nabla p'(X)$.

P7 . Faça $\epsilon = \|\nabla p(X)\| / \|\nabla f(X)\|$, $\epsilon' = \|\nabla f(X)\| / \|\nabla p'(X)\|$.

P8 . Escolha ϵ_k (0.1, 1).

P9 . Calcule

$$S(X) = - \left[\nabla f(X) + \frac{1}{\epsilon} \nabla p(X) + \epsilon' \nabla p'(X) \right].$$

P10. Se $\|S(X)\| > \epsilon_k$ vá para P11. Caso contrário faça

$$\epsilon_{k+1} = \alpha \epsilon_k, \quad \epsilon = \alpha \epsilon, \quad \epsilon' = \alpha' \epsilon', \quad X_{k+1} = X, \quad k = k+1 \quad \text{e vá para P3.}$$

P11. Resolva o problema $\min\{f(X) + \frac{1}{\epsilon} p(X) + \epsilon' p'(X)\}$ por qualquer método de minimização desvinculada com $\epsilon_4 = \epsilon_k$ e volte para P9; com o novo ponto X calculado.

SEÇÃO 2 - MÉTODO DE DIREÇÕES VIÁVEIS

O método de direções viáveis foi introduzido por ZOUTENDIJK, [24], em 1959 e se destina a resolver o seguinte problema de otimização:

19 Minimizar $f(X)$

$$X \in V$$

onde V é definido em (10) e as funções envolvidas, f e g_i , $i=1,2,\dots,m$ são continuamente diferenciáveis.

Um algoritmo que resolva (19) é dito método de direções viáveis se, dado um ponto X_i pertencente ao conjunto V , ele determina uma semi-reta $\{X/X = X_i + \mu S_i, \mu \geq 0\}$ passando pelo interior (relativo) de V , onde $S_i \in R^n$, e nessa semi-reta escolhe $X_{i+1} = X_i + \mu_i S_i$ tal que $f(X_{i+1}) < f(X_i)$. Assim, métodos de direções viáveis podem ser usados para resolver (19) apenas se V tem interior (relativo) não vazio, [2].

Desta forma, somente poderão ser admitidos vínculos do tipo igualdade se estes forem lineares. Em nosso tratamento consideraremos apenas vínculos do tipo desigualdade satisfazendo às seguintes:

20 HIPÓTESES

a. O interior do conjunto viável (10) é dado por

$$\overset{\circ}{V} = \{X \in R^n / g(X) < 0\}$$

b. $\overset{\circ}{V}$ é não vazio

c. Existe um ponto $X_0 \in V$ tal que o conjunto

$U = \{X \in \mathbb{R}^n / f(X) - f(X_0) \leq 0, g(X) \leq 0\}$ é compacto e tem interior não vazio.

DETERMINAÇÃO DA DIREÇÃO VIÁVEL - O método de direções viáveis é uma extensão do processo de CAUCHY. Neste procura-se diminuir a função objetivo caminhando na direção contrária à de gradiente (ver IV. 1). Naquele o objetivo é o mesmo, com a restrição de sempre permanecer na região viável.

Dado um ponto $X_j \in V$, uma direção $S_j \in \mathbb{R}^n$ é dita viável, a partir de X_j , se for possível determinar $\bar{\mu} > 0$ tal que para todo $\mu \in (0, \bar{\mu}]$ tem-se $g(X_j + \mu S_j) < 0$, isto é, $X_j + \mu S_j \in \overset{\circ}{V}$.

Em outras palavras, uma direção é viável, a partir de um ponto, se for possível "caminhar" pelo menos um pouco sobre ela sem sair do conjunto viável, supondo-se que o ponto de partida pertença a esse conjunto.

As variações de f e g_i , $i=1,2,\dots,m$, em X , na direção S podem ser medidas por

$$\langle \nabla f(X), S \rangle \text{ e}$$

$$\langle \nabla g_i(X), S \rangle, \quad i=1,2,\dots,m$$

A direção S é viável, a partir de X , se

$$21 \quad \langle \nabla g_i(X), S \rangle < 0, \quad i \in J_0(X) = \{i \mid g_i(X) = 0, i=1,2,\dots,m\}$$

e utilizável se (21) valer e

22 $\langle \nabla f(X), S \rangle < 0.$

Procura-se uma direção viável utilizável, isto é, que, simultaneamente reduza "bastante" f e penetre "bastante" em V . Isto não leva em conta vínculos quase violados cuja presença provoca ineficiência da busca e pode acarretar zig-zag. É neste ponto que entra o importante conceito de:

ϵ -atividade - Dado $\epsilon \geq 0$, $\epsilon \in \mathbb{R}$, $X \in V$, o conjunto de índices ϵ -ativos em X é definido por

23 $J_\epsilon(X) = \{i/g_i(X) + \epsilon \geq 0, i \in \{1, 2, \dots, m\}\}$

O conjunto $J_\epsilon(X)$ tem por objetivo evitar "engarrafamentos" do algoritmo. O valor de ϵ vai gradativamente diminuindo à medida em que o algoritmo se processa.

Considerando que a direção S seja de decréscimo tanto de f como de g_i , $i \in J_\epsilon(X)$, podemos dizer que as expressões (22) e (21) nos dão o decréscimo dessas funções na direção S .

Definamos o seguinte conjunto

24 $Z = \{S \in \mathbb{R}^n / |s_i| \leq 1, i=1, 2, \dots, n\}$

que, evidentemente contém a origem em seu interior.

Seja $s_\epsilon: V \rightarrow \mathbb{R}$ uma função definida por

25 $s_\epsilon = \max_{i \in J_\epsilon(X)} \{ \langle \nabla f(X), S \rangle, \langle \nabla g_i(X), S \rangle \}$

A expressão (25) nos fornece o valor do menor de crescimento inicial (maior crescimento inicial) entre todas as funções f e g_i , $i \in J_\epsilon(X)$. Ora, se esse decrescimento inicial ainda for bom, $s_\epsilon < 0$, significa que é possível de terminar um novo ponto onde os valores de todas as funções decresceram. Entretanto, se $s_\epsilon = 0$, o menor decrescimento inicial é nulo e não podemos garantir a determinação de um novo ponto, em $\overset{0}{V}$, no qual a função critério tenha diminuído seu valor.

Se para $\epsilon > 0$, tivermos $s_\epsilon = 0$, é feita uma redução no valor de ϵ e o processo segue normalmente.

Se para $\epsilon = 0$, tivermos $s_\epsilon = 0$ então, para certas condições de regularidade, mostra-se que as condições de KUHN-TUCKER são satisfeitas no ponto X , o que fornece uma regra de parada para o algoritmo.

O problema agora é determinar uma direção que permita decrescer todas as funções. Uma direção que fornece bons resultados é aquela segundo a qual é possível aumentar ao máximo o menor decrescimento obtido em (25), (minimizar o maior crescimento).

Definamos a função $s_\epsilon^0: V \rightarrow R$ por

26

$$s_\epsilon^0 = \min_{S \in Z} \max_{i \in J_\epsilon(X)} \{ \langle \nabla f(X), S \rangle, \langle \nabla g_i(X), S \rangle \}$$

Resolvendo o problema em (26) em relação a S , teremos determinado a direção que procuramos.

Podemos calcular o valor de s_ϵ^0 em (26) resolvendo o problema

$$27 \quad \text{Minimizar } s_\epsilon^0 \\ S \in Z$$

sujeito a

$$28 \quad s_\epsilon^0 = \max_{i \in J_\epsilon(X)} \{ \langle \nabla f(X), S \rangle, \langle \nabla g_i(X), S \rangle \}$$

Entretanto, (27)-(28) é equivalente a

$$29 \quad \text{Minimizar } s_\epsilon^0 \\ S \in Z$$

sujeito a

$$30 \quad s_\epsilon^0 \geq \langle \nabla f(X), S \rangle, \\ s_\epsilon^0 \geq \langle \nabla g_i(X), S \rangle, \quad i \in J_\epsilon(X).$$

Mostra-se facilmente que (29)-(30) equivale a

$$31 \quad \text{Minimizar } s_\epsilon^0 \\ \text{sujeito a} \\ -s_\epsilon^0 + \langle \nabla f(X), S \rangle \leq 0 \\ -s_\epsilon^0 + \langle \nabla g_i(X), S \rangle \leq 0, \quad i \in J_\epsilon(X)$$

$$s_i \leq 1 \quad i=1,2,\dots,n \\ -s_i \leq 1$$

O problema (31) é um problema de programação linear em R^{n+1} e pode ser resolvido pelo Método Simplex. **Abstrução**

será o par $(s_\epsilon^0, S_\epsilon)$, onde $s_\epsilon^0 \in \mathbb{R}$ e $S_\epsilon \in \mathbb{R}^n$, sendo $-s_\epsilon^0$ o menor decréscimo inicial verificado entre as funções f e g_i , $i \in J_\epsilon(X)$, na direção $S_\epsilon \in \mathbb{R}^n$, |2|.

O presente método não exige convexidade das funções envolvidas e sua convergência é tratada em |2|.

O algoritmo apresentado a seguir destina-se a resolver o problema (19) e pode ser encontrado em |2|.

32

ALGORITMO | 2 |

P1 . Escolha $\epsilon' > 0$, $\epsilon'' \in (0, \epsilon')$, $\alpha > 0$, $\beta' \in (0, 1)$, $\beta'' \in (0.5, 0.8)$
e um inteiro k tal que $5 \leq k \leq 10$.

P2 . Dado um ponto inicial $X_0 \in R^n$, verifique se $X_0 \in \overset{O}{V}$,
definido com V em (10). Se $X_0 \in \overset{O}{V}$ vá para P4.
Caso contrário vá para P3.

P3 . Faça $x_0 = \max\{g_i(X_0), i=1, 2, \dots, m\}$ e resolva o seguinte
problema (em R^{n+1}):
 $\min\{x_0 / -x_0 + g_i(X) \leq 0, i=1, 2, \dots, m\}$ para determinar um
ponto viável, $X' \in \overset{O}{V}$.

P4 . Faça $X_0 = X'$ e $i=0$.

P5 . Faça $\epsilon = \epsilon'$.

P6 . Faça $X = X_i$.

P7 . Defina o conjunto de índices

$$J_\epsilon(X) = \{j \in \{1, 2, \dots, m\} / g_j(X) + \epsilon \geq 0\}$$

P8 . Calcule o vetor $(s_\epsilon^O(X), S_\epsilon(X))$ (em R^{n+1}) resolvendo o se-
guinte problema

33

$$\text{Min } s_\epsilon^O$$

sujeito a

$$-s_\epsilon^O + \langle \nabla f(X), S \rangle \leq 0$$

$$-s_\epsilon^O + \langle \nabla g_i(X), S \rangle \leq 0$$

$$h_j \leq 1, j=1, 2, \dots, n$$

$$-h_j \leq 1$$

- P9 . Se $s_{\epsilon}^0 \leq -\alpha\epsilon$, faça $S(X)=S_{\epsilon}(X)$ e vá para P12.
 Caso contrário vá para P10.
- P10. Se $\epsilon \leq \epsilon''$ faça $\bar{\epsilon}=\epsilon$, resolva o problema (33) para $\epsilon=0$
 para determinar $(s_{\epsilon}^0, S_{\epsilon})$ e vá para P11.
 Caso contrário, faça $\epsilon=\beta'\epsilon$ e vá para P8.
- P11. Se $s_{\epsilon}^0=0$, PARE. Caso contrário faça $\epsilon=\beta'\epsilon$ e vá para P9.
- P12. Minimize $f(X)$ na direção $S(X)$, usando qualquer busca
 unidirecional, para determinar um novo ponto viável
 X_{i+1} e faça $i=i+1$.
- P13. Se $(i/k)=0$, módulo k , vá para P5.
 Caso contrário vá para P6.

CAPÍTULO VIIMÉTODO DE MINIMIZAÇÃO VINCULADA SEM DERIVADAS

Neste capítulo abordaremos um processo que foi introduzido em 1968 por PAVIANI e HIMMELBLAU, [25], que se intitula MÉTODO DE TOLERÂNCIA FLEXIVEL.

Durante a descrição do método estaremos sempre nos baseando em [1] e [25].

A característica principal deste processo é não necessitar de derivadas das funções critério e vínculos o que representa, em tempo, uma economia enorme na preparação dos dados, por parte do usuário. Uma comparação desta natureza é feita em [1], destacando o comportamento do método em estudo.

Além de apresentar facilidade na preparação, o Método de Tolerância Flexível apresenta bom desempenho em relação aos diversos tipos de problemas que lhe servem de teste, resultados também apresentados em [1].

A estratégia do processo é uma extensão da idéia de NELDER-MEAD, exposta no algoritmo descrito na seção V.3. Técnicas especiais foram criadas a fim de possibilitar a resolução de problemas sujeitos a vínculos tanto de desigualdade quanto de igualdade.

SEÇÃO 1 - MÉTODO DE TOLERÂNCIA FLEXIVEL

Este é um método que se destina a resolver o problema (II.1.1) repetido a seguir:

$$1 \quad \begin{aligned} & \text{Min } f(X), X \in R^n \\ & \text{sujeito a } g(X) \leq 0 \\ & h(X) = 0 \end{aligned}$$

Seu desenvolvimento teórico é baseado no processo de NELDER-MEAD (ver V. 3).

CRITÉRIO DE TOLERÂNCIA. Seja ϕ_k o critério de tolerância flexível de viabilidade, no k-ésimo estágio, definido por

$$2 \quad \phi_k = \min \left\{ \phi_{k-1}, \frac{\ell+1}{p+1} \sum_{i=1}^{p+1} \left| |x_i - x_{p+2}| \right| \right\}$$

sendo seu valor inicial ϕ_0 calculável através de:

$$3 \quad \phi_0 = 2(\ell+1)t.$$

Em (2) e (3):

ϕ_k = valor do critério de tolerância no estágio k.

ϕ_{k-1} = valor do critério de tolerância no estágio k-1.

ℓ = número de restrições de igualdade.

p = $n - \ell$ = número de graus de liberdade.

x_i = i-ésimo vértice do poliedro flexível.

x_{p+2} = centróide do poliedro flexível (ver V. 3).

t = tamanho inicial do poliedro (ver 12).

O critério de tolerância (2) é uma sequência positiva não crescente $(\phi_k)_{k \in \mathbb{N}}$. Os valores ϕ_k agem como um critério de tolerância de violação de vínculos durante toda a resolução do problema, bem como se presta a um critério de parada do algoritmo. Realmente vimos em (V. 3) que o poliedro flexível durante o processamento do algoritmo sofre contrações e expansões mas tende a diminuir de tamanho, em cada estágio, à medida que se aproxima do ótimo da função. A sequência (ϕ_k) como definida em (4) é então, claramente, positiva não crescente, isto é,

$$4 \quad \phi_0 \geq \phi_1 \geq \dots \geq \phi_k \geq 0.$$

Nas vizinhanças do ótimo de f , o poliedro tende para zero e o mesmo ocorre com ϕ_k , daí sua condição de regra de parada para o algoritmo, [1].

A FUNÇÃO T(.). Seja a função $T: \mathbb{R}^n \rightarrow \mathbb{R}$ definida por

$$5 \quad T(X) = \left[\sum_{i=1}^{\ell} (h_i(X))^2 + \sum_{i=1}^m u_i (g_i(X))^2 \right]^{1/2},$$

onde

u_i é o operador de Heaviside tal que $u_i=0$ para $g_i(X)<0$ e $u_i=1$ para $g_i(X) \geq 0$.

Como se observa, $T(.)$ é uma função não negativa para todo X de \mathbb{R}^n . Em particular, se $\sum_{i=1}^{\ell} (h_i(.))^2$ e $g_i(.)$ são convexas então $T(.)$ é convexa com um ponto de mínimo global X , viável, com $T(X)=0$. Por outro lado, $T(X)>0$ para todo X não viável. Assim, $T(.)$ pode ser usada para

saber se um ponto pertence ou não ao conjunto viável. Pode ocorrer entretanto, que $T(X) \approx 0$ e neste caso o ponto X estará muito próximo da região viável e daí o conceito de

QUASE-VIABILIDADE. A distinção entre pontos viáveis, não viáveis e quase-viáveis é dada por

- a. viável se $T(X)=0$,
- b. não viável se $T(X) > \phi_k$,
- c. quase-viável se $0 \leq T(X) \leq \phi_k$.

Desta forma, a região de quase viabilidade é definida por

$$6 \quad \{X/T(X) - \phi_k \leq 0\}$$

TRANSFORMAÇÃO DO PROBLEMA. A idéia básica do processo é transformar o problema (1) no problema abaixo:

$$7 \quad \text{Min } f(X), X \in R^n$$

sujeito a

$$T(X) - \phi_k \leq 0.$$

Durante sua aplicação, o algoritmo procura obter novos vértices para o poliedro flexível, de tal maneira que esses novos vértices sejam viáveis ou quase viáveis. Em cada estágio é pois bastante minimizar $T(X)$ até obter $T(X) \leq \phi_k$, o que pode ser feito por qualquer método de minimização desvinculada sem derivadas. No presente algoritmo, o processo utilizado é o de NELDER-MEAD (ver V. 3).

É preciso, no entanto, mostrar que (7) é equivalente a (1). Para tanto é suficiente observar o comportamento de ϕ .

Em virtude de ser (ϕ_k) uma sequência positiva não crescente, tal que $\phi_k=0$ apenas quando se atinge o ponto de ótimo para $f(\cdot)$, a região de quase-viabilidade (6) é gradualmente restringida em cada estágio do algoritmo. No limite, isto é, quando o poliedro se identificar com o ponto de ótimo, então o valor final de ϕ é igual a zero e apenas pontos viáveis podem satisfazer (6). Em outras palavras, se $\phi=0$, desde que $T(X)$ não pode ser negativa então só pode ser igual a zero o que requer que X , ponto de ótimo, seja viável, |1|.

CÁLCULO DOS VÉRTICES DO POLIEDRO INICIAL

Para se iniciar a busca são necessários $p+1$ pontos iniciais ($n+1$ quando $\ell=0$) que podem ou não formar um simplex regular em R^n . Os $p+1$ pontos devem ser escolhidos de tal maneira que qualquer subconjunto formado por p pontos seja linearmente independente. Para fins práticos é conveniente partir de um ponto inicial x_0 e construir um simplex regular. Os $p+1$ vetores de R^n são determinados por

$$8 \quad x_i = x_0 + D_i, \quad i=1,2,\dots,p+1$$

onde D_i representa a i -ésima linha de uma matriz $(p+1) \times n$, dada por

$$9 \quad D = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ u & v & v & \dots & v \\ v & u & v & \dots & v \\ \cdot & \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \cdot & \dots & \cdot \\ \cdot & \cdot & \cdot & \dots & \cdot \\ v & v & v & \dots & u \end{bmatrix}$$

onde

$$10 \quad u = (t/n\sqrt{2})(\sqrt{n+1} + n-1), \text{ e}$$

$$11 \quad v = (t/n\sqrt{2})(\sqrt{n+1} - 1).$$

Em (11) e (12), t é uma constante que determina o tamanho do simplex.

O valor inicial de t pode ser calculado em função do intervalo de variação esperado para as variáveis independentes.

Se os limites inferior e superior das componentes de x_0 podem ser estimadas, então t pode ser calculado por

$$12 \quad t = \min \left[(0.2/n) \sum_{i=1}^n L_i \right]$$

onde L_i é a diferença entre os valores final e inicial, esperada para a i -ésima componente de x_0 . Se tais diferenças não são conhecidas, qualquer valor razoável para t representa uma boa escolha, [25]. Em nosso trabalho optamos por fazer $t=1$ pois consideramos normalmente difícil estabelecer valores verdadeiros para L_i , a menos que já se conheça a solução do problema.

Na minimização de $T(\cdot)$, definida em (5), o valor de t é calculado empiricamente através da relação

$$13 \quad t = 0.05\phi_k$$

O método em estudo não exige convexidade das funções em (1) e sua convergência é comentada em [1].

O algoritmo que apresentamos a seguir destina-se a resolver o problema definido em (1), usando para a minimização de $T(X)$ o método desvinculado de NELDER-MEAD descrito em (V. 3).

ALGORITMO | 25 |

Coment.: A expressão satisfaça a equação (6) significa calcular $T(X)$ através de (5) e minimizá-la até que $T(X) \leq \phi_k$ obtendo-se, então um (novo) ponto X viável, ou quase-viável. ϕ_k é o valor de $\phi(2)$ no k -ésimo estágio.

P1 . Escolha $X_0 \in R^n$, $\epsilon \geq 0$, $\alpha=1$, $\beta=0.5$ e $\gamma=2$.

Calcule t por (12) ou faça t igual a um valor conveniente. Calcule ϕ_0 por (3), satisfaça a equação (6), construa um simplex regular a partir do ponto X_0 (viável) obtido e faça $k=0$.

P2 . Satisfaça a equação (6) para todos os vértices do poliedro flexível.

P3 . Calcule $f(X_i)$, $i=1,2,3,\dots,p+1$

P4 . Determine os vértices X_h e X_ℓ correspondentes, respectivamente ao maior e menor valor de $f(X_i)$, $i=1,2,\dots,p+1$

P5 . Calcule o centroide do poliedro flexível por

$$X_{p+2} = \frac{1}{p} \left(\sum_{i=1}^{p+1} X_i - X_h \right)$$

P6 . Calcule ϕ_k por (2).

P7 . REFLEXÃO - Calcule $X_{p+3} = X_{p+2} + \alpha(X_{p+2} - X_h)$ e satisfaça a equação (6). Calcule $f(X_{p+3})$.

P8 . Se $f(X_{p+3}) > f(X_\ell)$, vá para P11.

Caso contrário vá para P9.

- P9 . EXPANSÃO - Calcule $X_{p+4} = X_{p+2} + \gamma(X_{p+3} - X_{p+2})$. Satisfaça a equação (6) e calcule $f(X_{p+4})$.
- P10. Se $f(X_{p+4}) < f(X_\ell)$ substitua X_h e $f(X_h)$ por X_{p+4} e $f(X_{p+4})$, respectivamente, e vá para P17. Caso contrário substitua X_h por X_{p+3} , $f(X_h)$ por $f(X_{p+3})$ e vá para P17.
- P11. Se para algum $i \neq h$, $f(X_{p+3}) < f(X_i)$ substitua X_h por X_{p+3} , $f(X_h)$ por $f(X_{p+3})$ e vá para P17. Caso contrário vá para P12.
- P12 . Se $f(X_{p+3}) > f(X_h)$ vá para P14.
Caso contrário vá para P13.
- P13. Substitua X_h por X_{p+3} , $f(X_h)$ por $f(X_{p+3})$ e vá para P14.
- P14. CONTRAÇÃO - Calcule $X_{p+5} = X_{p+2} + \beta(X_h - X_{p+2})$, satisfaça a equação (6) e calcule $f(X_{p+5})$.
- P15. Se $f(X_{p+5}) < f(X_h)$ substitua X_h por X_{p+5} , $f(X_h)$ por $f(X_{p+5})$ e vá para P17. Caso contrário vá para P16.
- P16. REDUÇÃO - Calcule X_i , $i=1, 2, \dots, p+1$ por
 $X_i = X_\ell + 0.5(X_i - X_\ell)$ e $f(X_i)$, $i=1, 2, \dots, p+1$, para os novos valores de X_i . Faça $k=k+1$ e vá para P2.
- P17. Se $\phi_k \leq \epsilon$. PARE. Caso contrário faça $k=k+1$ e vá para P2.

CAPÍTULO VIIIMANIPULAÇÃO DE PRECISÕES

Todos os algoritmos estudados nos capítulos anteriores incluem a resolução de subproblemas: nos métodos desvinculados e direções viáveis, o subproblema é a busca unidirecional e nos de penalidades e tolerância flexível o subproblema é o algoritmo desvinculado.

Teoricamente, a resolução exata de tais subproblemas levaria tempo infinito. Para contornar este tipo de problema, os algoritmos são transformados de conceituais em implementáveis, através de truncamento, passando agora a levar apenas tempo finito em seu processamento.

Normalmente o truncamento é feito lançando-se mão de precisões obtidas heurísticamente. Em nosso caso pretendemos formalizar o estudo de tais precisões com o objetivo de calculá-las a partir de informações inerentes a cada problema a ser resolvido pelo algoritmo.

Polak apresenta em [2] modelos de algoritmos implementáveis que incluem a manipulação de precisões e demonstra a sua convergência. Na seção seguinte será estudado um modelo de algoritmo implementável, baseado em [2], cuja eficiência é comentada. Nas seções posteriores serão tratados outros assuntos relacionados à manipulação de preci-

sões, tais como convergência, intervalo inicial de busca e regras de parada para algoritmos.

No presente trabalho são utilizados quatro tipos de precisões:

- 1 ϵ_1 - Esta é uma precisão que deve ser fornecida em unidades de R^n . É utilizada para o estabelecimento das precisões ϵ e ϵ' empregadas em testes de parada das buscas de secção-áurea e Davies-Swann-Campey-Powell, respectivamente para o intervalo final que contém λ^* (ver III) e suficiente decréscimo da função. Essas regras serão comentadas adiante.
- 2 ϵ_2 - Esta precisão corresponde ao zero computacional de R^n e deve também ser fornecida em unidades compatíveis. É utilizada em teste de parada das buscas unidirecionais (ver III 3.4) através de comparações com o intervalo que contém λ^* . É também usada como regra de parada de algoritmos desvinculados por comparações com a norma da diferença entre pontos (vetores) consecutivos determinados pelos algoritmos.
- 3 ϵ_3 - Corresponde ao zero computacional do critério e deve ser fornecida em unidades da função. Sua utilização é feita em regra de parada de algoritmos desvinculados através de comparações com diferenças entre três valores consecutivos de f .
- 4 ϵ_4 - É a precisão que corresponde ao zero computacional do gradiente da função. É utilizado como regra de parada de métodos desvinculados que usam derivadas, fazendo-se sua comparação com a norma do gradiente de f em cada ponto obtido pelo algoritmo.

SEÇÃO 1 - MODELO IMPLEMENTÁVEL

O problema desvinculado é

1 Minimizar $f(X)$

$$X \in R^n$$

O modelo implementável, abaixo, se refere a métodos desvinculados que usam gradientes e se destina a resolver o problema (1). Sua base é o modelo conceitual apresentado em (II. 4.6). O algoritmo, em cada ponto $X_i \in R^n$, gera uma direção S_i a partir de $\nabla f(X_i)$, tal que $S_i=0$ se e somente se $\nabla f(X_i)=0$. A introdução de precisões para o modelo em (II.4.6) leva ao seguinte

2 ALGORITMO

P0 . Escolha $X_0 \in R^n$.

P1 . Faça $i=0$.

P2 . Obtenha uma direção de busca $S_i \in R^n$.

P3 . Calcule $\|S_i\|, \|\nabla f(X_i)\|$.

P4 . Se $\|S_i\| < \epsilon_1$, pare. Caso contrário vá para P5.

P5 . Faça $\epsilon = \epsilon_1 / \|S_i\|$ e $\epsilon' = \epsilon_1 \|\nabla f(X_i)\|$.

P6 . Calcule λ'_i tal que exista λ^* satisfazendo a

$$|\lambda'_i - \lambda^*| \leq \epsilon, \quad f(X_i + \lambda^* S_i) = \min_{\lambda \geq 0} \{f(X_i + \lambda S_i)\}$$

P7 . Se $f(X_i + \lambda'_i S_i) - f(X_i) > -\epsilon'$, faça $\epsilon_1 = \epsilon_1 / 2$, $\epsilon = \epsilon / 2$, $\epsilon' = \epsilon' / 2$ e vá para P6.

Caso contrário faça $X_{i+1} = X_i + \lambda'_i S_i$, $i=i+1$ e vá para P2.

O modelo acima difere do de Polak no ponto em que neste a função é comparada com $\varepsilon_1 \|\nabla f(X_i)\|$ enquanto que naquele deve-se usar $\alpha\varepsilon_1$, $\alpha > 0$ fixo. Em nosso caso a precisão para a função fica coerente pois corresponde ao decréscimo de f para variações de X_i : $\varepsilon_1 \nabla f(X_i) / \|\nabla f(X_i)\|$.

A modificação proposta levou a bons resultados, embora não se demonstre neste trabalho a convergência dos algoritmos adaptáveis ao modelo: esse tratamento teórico exigiria a manipulação de vários resultados sobre algoritmos, fugindo à finalidade desta tese.

Nas buscas em (III. 3 e 4) o teste no passo 6 foi incluído na própria busca o que pode ser visto em seus respectivos algoritmos. As buscas em (III. 1 e 2) não fazem uso desse tipo de teste em face de sua maneira própria de calcular o valor de λ' .

Em métodos em que a direção S_i é calculada a partir de gradientes, $\|S_i\|$ é aproximadamente da ordem de $\|\nabla f(X_i)\|$, o que permite usar $\|S_i\|$ em vez de $\|\nabla f(X_i)\|$ no passo 5 do modelo (2), levando a bons resultados.

CÁLCULO DE ρ - As buscas unidirecionais iniciam pela determinação de um intervalo inicial $[a_0, b_0]$ (ver III) que contenha um valor λ^* correspondente ao passo 6 em (2). Um bom valor inicial para λ pode evitar uma série de cálculos na determinação de $[a_0, b_0]$.

Baseados no fato de que para uma direção S a de

derivada direcional é nula no ponto de mínimo unidirecional, podemos aproximar o cálculo de λ inicial usando a seguinte idéia, ilustrada nas figuras 1 e 2.

Se a derivada segunda de f na direção S é constante, então para variações de X na direção S o decréscimo da função tende a se anular à medida que X se aproxima do ponto de mínimo unidirecional. É então possível calcular aproximadamente um valor inicial para λ . No algoritmo que apresentamos a seguir foram introduzidos testes adicionais com o fim de atender a casos onde o comportamento da função não permita o cálculo de $\rho(\lambda \text{ inicial})$ conforme se faz no passo 6 do algoritmo abaixo.



Fig. 1

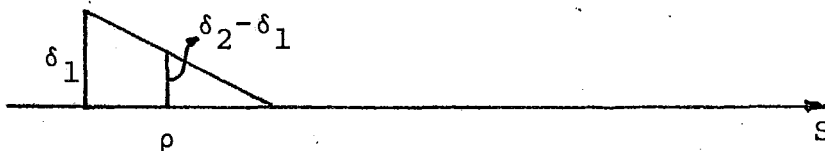


Fig. 2

3 ALGORITMO

P0 . Dados $\alpha=0.01$, $\epsilon_3 \in \mathbb{R}$, $S \in \mathbb{R}^n$, $X \in \mathbb{R}^n$.

P1 . Calcule $\delta_1 = \theta(\alpha, S) = f(X) - f(X + \alpha S)$.

P2 . Se $\delta_1 \leq 0$ faça $\rho = \alpha$ e PARE.

Caso contrário vá para P3.

P3 . Faça $\alpha = 2\alpha$ e calcule $\delta_2 = \theta(\alpha, S)$.

P4 . Se $\delta_2 < \delta_1$ faça $\rho = \alpha$ e PARE.

Caso contrário vá para P5.

P5 . Se $\delta_2 > 2\delta_1 - \epsilon_3$ faça $\delta_1 = \delta_2$ e vá para P3.

Caso contrário vá para P6.

P6 . Calcule $\rho = -\alpha\delta_1 / (\delta_2 - \delta_1)$ e PARE.

4 PARADA DAS BUSCAS

Após determinado o intervalo inicial as buscas em (III. 3 e 4) fazem o seu estreitamento testando em cada diminuição do intervalo, o seu tamanho com o valor de $\epsilon = \epsilon_1 / \|S_1\|$ (ver 2). Se o tamanho do intervalo for menor que ϵ , a primeira condição de parada está satisfeita e o teste entre o valor do decréscimo de f para o λ obtido, e ϵ' é efetuado. Se satisfeito, a busca retorna com sucesso, senão aumentam-se as precisões envolvidas (ver III. 3 e 4) e o intervalo volta a ser estreitado até se obter um bom decréscimo para f . Em caso de se obter uma precisão para o intervalo, tal que $\epsilon_1 \leq \epsilon_2$ então a busca acaba: com sucesso se o decréscimo for positivo ou insucesso se não for positivo.

5 CONVERGÊNCIA DE ALGORITMOS

O algoritmos desvinculados tem sua convergência comentada nas seções onde são expostos. Acúmulo de erros devidos a imprecisões de cálculo das buscas pode afetar a convergência dos métodos. O que se faz então é recompor o algoritmo sempre que a busca unidirecional não conseguir encontrar um ponto melhor, fazendo a direção de busca, contrária à do gradiente. Se a busca falhar novamente então não é possível encontrar um ponto melhor e o método acaba. Entretanto, em métodos de CAUCHY não é feito qualquer "resetamento" da direção de busca e o algoritmo para na primeira falha da

busca. Outra técnica empregada é "resetar" o algoritmo a cada grupo de xn iterações, automaticamente. Uma boa escolha para x parece ser $x=2$. Em nosso caso isto é feito apenas no algoritmo de FLETCHER-REEVES.

Assim agindo, garantimos que a convergência dos algoritmos não é afetada uma vez que o método de CAUCHY tem convergência demonstrada.

6 REGRAS DE PARADAS DOS ALGORITMOS

Os métodos desvinculados com derivadas possuem, em nosso caso, 3 regras de parada:

1. O número de iterações solicitado pelo usuário foi atingido.
2. O gradiente da função atinge um valor menor que ϵ_4 .
3. Três pontos consecutivos possuem diferença (em norma) menor que ϵ_2 e três valores consecutivos da função possuem diferença menor que ϵ_3 .

Para métodos desvinculados sem derivadas existem outras regras de parada que podem ser vistas nos correspondentes algoritmos.

SEÇÃO 2 - MÉTODO DE PENALIDADES

Conforme exposto no capítulo VI a resolução do problema (II. 1.1) pelo método de penalidades é obtida pela resolução de uma sequência de problemas desvinculados (ver VI.1). Cada problema desvinculado é resolvido independentemente pelos algoritmos desvinculados mas neste caso a regra de parada dos algoritmos e o intervalo inicial da busca são estabelecidos de forma diferente.

Em cada estágio do método de penalidades um novo problema desvinculado deve ser resolvido, e em cada estágio do subproblema há um segundo subproblema que é a determinação do mínimo unidirecional. Sabe-se que tal mínimo sempre se acha dentro da região viável e, portanto, é sempre possível determinar um intervalo inicial de busca em cada estágio do subproblema.

O algoritmo que apresentamos a seguir parte de um valor (ρ) calculado em (3) e determina um intervalo inicial de busca para cada estágio do subproblema do método de penalidades, sendo seu posterior refinamento feito pelas próprias buscas unidirecionais.

7 ALGORITMO

P0 . Dados $X \in R^n$ e $S \in R^n$

P1 . Faça $i=0$, $\lambda_{i-1}=0$, $\lambda_0=0$, $k=0$

P2 . Calcule $\rho > 0$ (ver 3) e faça $\lambda = \rho$

P3 . Se $X + \lambda S$ é viável, faça $i=i+1$, $\lambda_i = \lambda$ e vá para P4.

Caso contrário faça $\rho = \rho/2$, $\lambda = \lambda - \rho$, $k=1$ e vá para P3.

P4 . Se $f(X + \lambda_i S) \geq f(X + \lambda_{i-1} S)$ faça $a_0 = \lambda_{i-2}$, $b_0 = \lambda_i$ e PARE.

Caso contrário vá para P5.

P5 . Se $k=0$ faça $\rho = \lambda_i$, $\lambda = \lambda + \rho$ e vá para P3.

Caso contrário faça $\rho = \rho/2$, $\lambda = \lambda + \rho$ e vá para P3.

No caso especial das buscas de GOLDSTEIN e ARMIJO (ver III) devem ser feitas pequenas alterações envolvendo a inclusão de alguns testes. As modificações necessárias podem ser encontradas no capítulo XII.

Os algoritmos que resolvem os subproblemas podem não obedecer às mesmas regras de parada descritas em (6). Polak realizou um importante trabalho neste setor introduzindo um processo de truncamento para os subproblemas com relação aos gradientes. Uma precisão inicial para o gradiente da função penalizada é fornecida e em cada estágio do algoritmo de penalidades essa precisão vai sendo aumentada da mesma forma que as penalidades, garantindo-se com isto a convergência do método.

SEÇÃO 3 - MÉTODO DE DIREÇÕES VIÁVEIS

Como no método de penalidades, também neste caso a determinação do intervalo inicial é feita por um algoritmo especial. No método de direções viáveis entretanto, o sub-problema é a própria minimização unidirecional e a determinação do intervalo inicial é feita em cada estágio do algoritmo de direções viáveis. Diferentemente do método de penalidades pode ocorrer no entanto que a função não possua mínimo unidirecional dentro da região viável. Neste caso o ideal seria determinar um ponto para o qual existisse o maior número de vínculos violados possível.

O algoritmo que apresentamos a seguir tenta obter esse ponto da seguinte maneira: quando é determinado um ponto para o qual haja, pelo menos, um vínculo ϵ -violado, procura-se saber se para esse mesmo ponto existe um número maior de vínculos 3ϵ -violados. Se houver são feitas duas tentativas com a finalidade de torná-los ϵ -violados, voltando-se em seguida com λ' igual ao valor final obtido para λ . Note-se que neste caso não se determina intervalo inicial: o valor final de λ é o λ' procurado (ver III).

As mesmas considerações da seção anterior são válidas aqui com relação às buscas unidirecionais de GOLDSTEIN e ARMIJO.

8 ALGORITMO

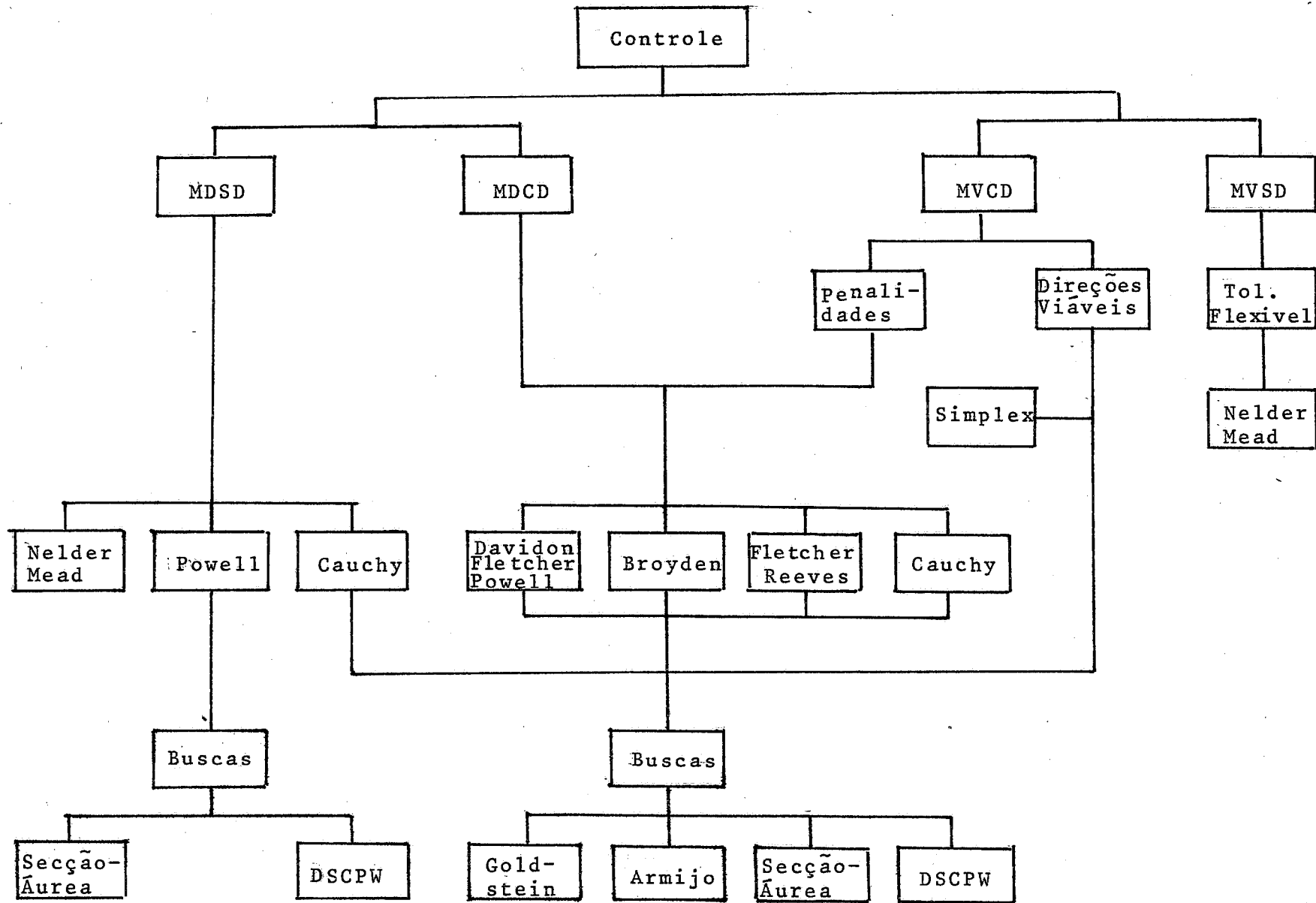
- P0 . Dados $X \in \mathbb{R}^n$ e $S \in \mathbb{R}^n$.
- P1 . Faça $i=0, j=0, k=0, \lambda_{-1}=0, \lambda_0=0$.
- P2 . Calcule $\rho > 0$ (ver 3) e faça $\lambda = \rho$.
- P3 . Se $X + \lambda S$ é viável, faça $i=i+1, \lambda_i = \lambda$ e vá para P4.
Caso contrário faça $\rho = \rho/2, \lambda = \lambda - \rho, k=1$ e vá para P3.
- P4 . Se $f(X + \lambda_i S) \geq f(X + \lambda_{i-1} S)$ faça $a_0 = \lambda_{i-2}, b_0 = \lambda_i$ e PARE.
Caso contrário vá para P5.
- P5 . Se $k=0$ faça $\rho = \lambda_i, \lambda = \lambda + \rho$ e vá para P3.
Caso contrário vá para P6.
- P6 . Se $X + \lambda S$ é ϵ -viável faça $\rho = \rho/2, \lambda = \lambda + \rho$ e vá para P3.
Caso contrário vá para P7.
- P7 . Calcule $n_1 =$ número de vínculos ϵ -violados e $n_2 =$ número de vínculos 3ϵ -violados.
- P8 . Se $n_2 = n_1$, faça $\lambda' = \lambda_i$ e PARE: acabou a busca.
Caso contrário vá para P9.
- P9 . Se $j < 2$, faça $\rho = \rho/2, \lambda = \lambda + \rho, j = j+1$ e vá para P3.
Caso contrário faça $\lambda' = \lambda_i$ e PARE: acabou a busca.

CAPÍTULO IXESTRUTURA DO SISTEMA

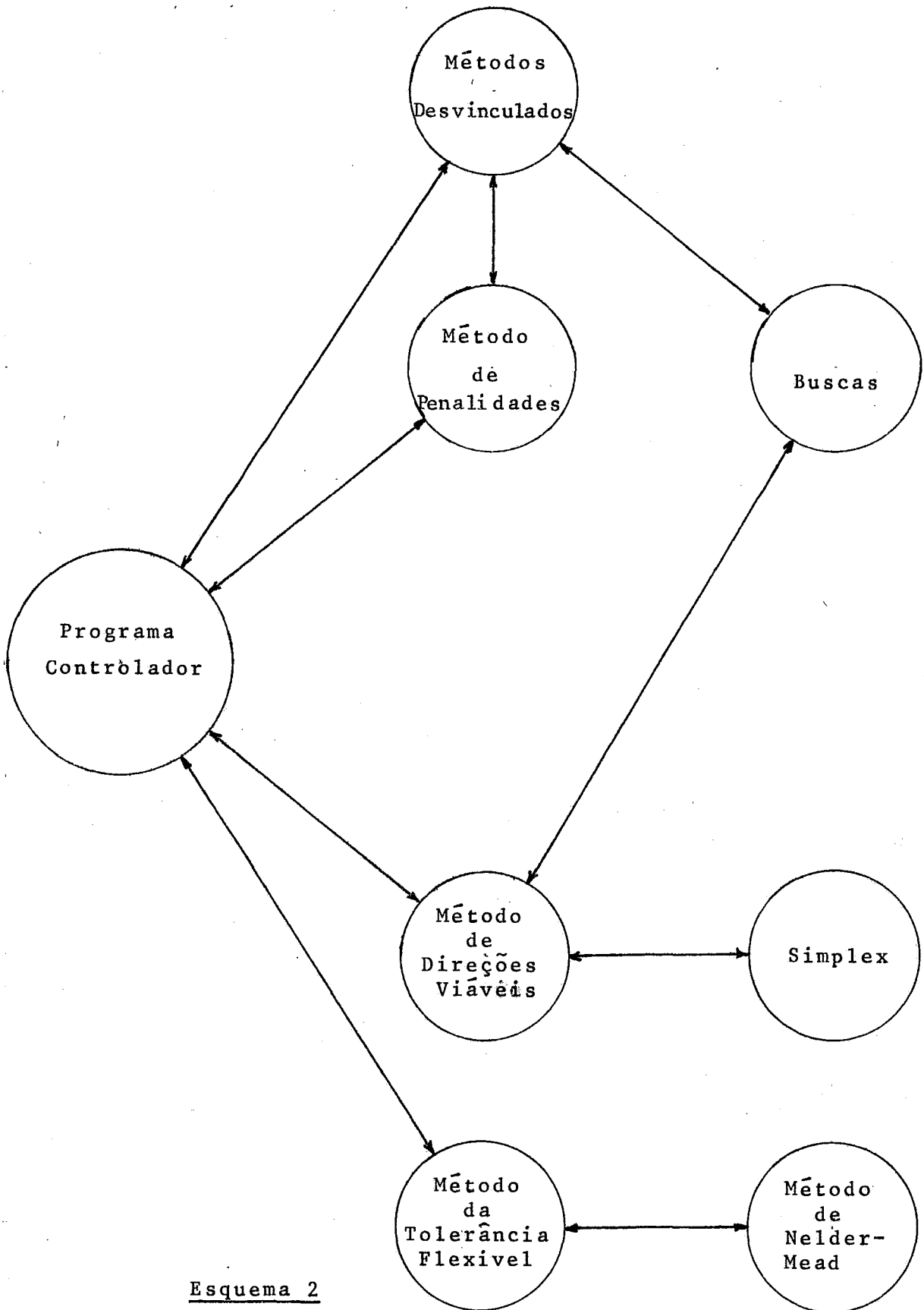
Nossa intenção aqui é apresentar um fluxograma de todo o sistema implementado, explicar como o programa principal e as subrotinas se interligam e como podem ser introduzidas novas subrotinas no sistema sem acarretar grandes modificações no conjunto.

Na interligação das rotinas usamos um método simples e sem sofisticação mas que consideramos eficiente e de fácil manejo pelo usuário. Maiores detalhes podem ser obtidos no capítulo X onde se explica a utilização de todos os algoritmos.

A seguir apresentamos dois esquemas, onde se pode observar o funcionamento de todo o sistema. O primeiro mostra o inter-relacionamento de todos os algoritmos que o compõem, e o segundo indica o funcionamento relativo dos métodos, obedecendo este às restrições impostas por aquele.



Esquema 1



Esquema 2

Afim de tentarmos explicar o funcionamento de cada rotina do conjunto, dividi-la-emos em quatro grupos principais

- 1 - Rotinas de uso geral.
- 2 - Rotinas de métodos desvinculados
- 3 - Rotinas de métodos vinculados
- 4 - Rotinas de programação linear

É conveniente esclarecer que rotina para nós indica qualquer programa, em FORTRAN, que faz parte do conjunto de listagens do capítulo XII.

Na classificação acima, rotina de uso geral é um título que enquadra todas aquelas que executam tarefas completamente genéricas, sem se prenderem especificamente a este ou àquele método.

Temos ao todo, no trabalho, 48 rotinas, cujos nomes contêm sempre 5 letras que, na medida do possível, procuram expressar o tipo de trabalho que executam ou a qual método se referem.

A seguir faremos a descrição sumária de todas elas, em ordem alfabética dentro da classificação acima. O programa controlador, no entanto, é colocado em primeiro lugar em face de sua posição hierárquica na estrutura do sistema.

1 ROTINAS DE USO GERAL

Nesta categoria encontramos:

1. Programa Controlador - Controla a leitura de dados, seleção de algoritmos escolhidos pelo usuário, número de problemas a executar, algumas impressões e escolhe algoritmos e valores de certos parâmetros quando o usuário não informa por qual método será resolvido o problema.
2. CRIVO - Examina os dados fornecidos pelo usuário, detectando possíveis enganos.
3. FUNCF - Na verdade esta rotina deve ser fornecida pelo usuário, mas foi incorporada com o objetivo de alertá-lo, quando se esquecer de incluí-la entre os dados de entrada, através do envio de uma mensagem adequada.
4. FUNGD - idem
5. FUNHI - idem
6. GRADF - idem
7. GRADV - idem
8. GRADX - Seleciona qual rotina de gradiente chamar de acôrdo com o algoritmo usado.
9. IMPRS - Imprime resultados intermediários e finais bem como outras mensagens de interesse para o usuário.

10. LEIAS - Faz a leitura de todos os dados necessários à solução do problema.
11. NORMA - Calcula a norma de um vetor determinado pelo algoritmo que a chama.
12. SELE1 - Seleciona o algoritmo indicado pelo usuário.
13. SELE2 - Seleciona a busca unidirecional escolhida pelo usuário.
14. SELE3 - Seleciona o algoritmo determinado pelo usuário para resolver o subproblema de métodos vinculados. No caso, apenas do método de penalidades.

2 ROTINAS DE MÉTODOS DESVINCULADOS

Nesta classe temos:

1. CONVE - Verifica as condições de convergência para os algoritmos desvinculados, determinando ou não a interrupção da execução.
2. COXIC - É o algoritmo de CAUCHY com derivadas.
3. COXIS - É o algoritmo de CAUCHY, sem derivadas.
4. DFLEP - É uma rotina comum aos métodos de DAVIDON-FLETCHER-POWELL e BROYDEN.
5. DIREC - É usada pelas buscas unidirecionais. Calcula um

novo ponto, dados o anterior e a direção de busca.

6. DSCPW - É a busca unidirecional de DAVIES-SWANN-CAMPEY-POWELL.
7. FRIVS - É o algoritmo de FLETCHER-REEVES.
8. GOSEC - É a busca unidirecional de seção-áurea (GOLDEN SECTION).
9. LAMBI - Calcula o λ inicial (p_0) para as buscas unidirecionais.
10. MDBRD - Atualiza a matriz direcional do método de -
BROYDEN.
11. MDDFP - Atualiza a matriz direcional do método de -
DAVIDON-FLETCHER-POWELL.
12. NEDMD - É o algoritmo de NELDER-MEAD.
13. POWEL - É o algoritmo de POWELL.
14. PLAK1 - É a busca unidirecional de GOLDSTEIN.
15. PLAK2 - É a busca unidirecional de ARMIJO.

3 ROTINAS DE MÉTODOS VINCULADOS

São as seguintes:

1. CENTR - Calcula o centróide e seleciona os vértices do

poliedro correspondentes ao maior e menor valor da função, no método de tolerância flexível.

2. CLCFI - Calcula o valor da função ϕ critério de tolerância do método de tolerância flexível.
3. CRITV - Verifica a violação de vínculos dos métodos de penalidades e direções viáveis e calcula o intervalo inicial.
4. DIRVS - É o algoritmo de direções viáveis.
5. FLEXT - É o algoritmo de tolerância flexível.
6. FUNCP - Calcula o valor da função do problema penalizado do método de penalidades.
7. GRADP - Calcula os gradientes das funções penalidades do método de penalidades.
8. INDIC - Determina o conjunto de índices de vínculos violados para os métodos de penalidades e de direções viáveis.
9. INTER - Faz, se necessário, a interpolação entre pontos interiores e exteriores à região viável, no método de tolerância flexível, quando o problema não apresenta restrições de igualdade.
10. OPERC - Calcula reflexão, expansão, contração e redução no método de tolerância flexível.
11. PENAF - Calcula o valor das funções penalidades do método

de penalidades.

12. PENAL - É o algoritmo do método de penalidades.
13. VERTS - Calcula os vértices do simplex regular nos métodos de NELDER-MEAD e tolerância flexível.
14. VIAVL - Verifica se os pontos calculados pelo método de tolerância flexível são viáveis ou quase-viáveis.
15. TESIS - Calcula o valor da função $T(X)$ do problema desvinculado do método de tolerância flexível e para o método de direções viáveis.

4 ROTINAS DE PROGRAMAÇÃO LINEAR

1. CAREX - Calcula o problema de programação linear ou as direções de busca do método de direções viáveis.
2. SINEX - Prepara os dados para a rotina anterior.
3. VETEX - Faz a transmissão de dados do método de direções viáveis para a rotina Carex.
4. VINEX - Faz a preparação de vetores para a transmissão de dados feita pela Vetex.

FUTURAS INCLUSÕES

Neste ponto conhecemos, mesmo superficialmente, todas as rotinas que compõem o método e seu respectivo funcionamento. Vejamos agora os procedimentos básicos a serem

seguidos em caso de inclusão de novos algoritmos. Existem certas particularidades no sistema que devem ser observadas:

- a) As buscas sempre recebem o ponto anterior λ' e a direção nas variáveis VY, XM e DR, respectivamente.
- b) Ao gradiente da função objetivo corresponde a variável GX.
- c) Aos gradientes dos vínculos de igualdade e desigualdade correspondem respectivamente as variáveis GH e GG.
- d) O novo ponto, o valor da função objetivo, o valor dos vínculos de igualdade e de desigualdade são colocados respectivamente nas variáveis VX, FX, HI e GD. VX também contém o ponto inicial.
- e) As variáveis inteiras IN e IO contêm os números relativos aos dispositivos de entrada e saída, respectivamente. Em nosso caso IN=8 e IO=5.

No caso de novas inclusões existem, basicamente, 3 rotinas a serem alteradas:

CRIVO - consistência de dados iniciais

IMPRS - impressão de resultados e mensagens

SELE1 - seleção do algoritmo principal

Contudo, é possível que haja necessidade de outras

alterações dependendo do caso.

A numeração dos algoritmos foi feita de tal maneira que as futuras inclusões, se efetuadas, não alterem a disposição dos métodos agrupados segundo o tipo (desvinculado, vinculado ou programação linear) e a natureza (com ou sem derivadas). Na rotina SELE1 há espaço reservado para novas inclusões, em número igual ao existente.

Quando o usuário não indica o algoritmo a ser utilizado o programa assume o seguinte:

DESVINCULADO

COM DERIVADA: DAVIDON-FLETCHER-POWELL com a busca de
DAVIES-SWANN-CAMPEY-POWELL.

DESVINCULADO

SEM DERIVADAS: POWELL com a busca de DAVIES-SWANN-CAMPEY-
POWELL.

VINCULADO

COM DERIVADAS: PENALIDADES com DAVIDON-FLETCHER-POWELL e
a busca de DSCPW.

CAPÍTULO XUTILIZAÇÃO DO SISTEMA

Este é um capítulo especialmente dedicado ao usuário, onde é exposta, detalhadamente, a utilização dos algoritmos implementados.

REQUISITOS - Os algoritmos foram programados em FORTRAN IV-G e o computador utilizado um IBM/360 modelo 40 com memória de 256K-bytes, instalado no Núcleo de Computação Eletrônica da UFRJ. A exigência mínima de memória para a utilização dos algoritmos é de aprox./120 KB. É nossa intenção, em futuro próximo, fazer uma adaptação do conjunto ao IBM-1130 afim de atender a uma faixa maior de possíveis usuários, fazendo mais uso de memória auxiliar.

Para maior eficiência na utilização do sistema é recomendável que o usuário possua, pelo menos, conhecimentos básicos de otimização muitas vezes necessários à correta interpretação de certos resultados. Evidentemente, conhecimentos de FORTRAN e do sistema IBM/360 são também indispensáveis.

DIMENSÕES DO PROBLEMA

- 1 Métodos desvinculados - Neste caso a formulação do problema é:

$$\text{Min } f(X), f: \mathbb{R}^n \rightarrow \mathbb{R}, X \in \mathbb{R}^n.$$

- 2 Métodos vinculados - O problema a ser resolvido é:

$$\text{Min } f(X), f: \mathbb{R}^n \rightarrow \mathbb{R}, X \in \mathbb{R}^n$$

sujeito a

$$g(X) \leq 0, g: \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$h(X) = 0, h: \mathbb{R}^n \rightarrow \mathbb{R}^\ell$$

- 3 Programação linear - Formulação:

$$\text{Min } C'X, C, X \in \mathbb{R}^n$$

sujeito a

$$AX = b, A, (m,m), b \in \mathbb{R}^m$$

$$X \geq 0$$

Em (1), (2) e (3): $n \leq 100$

Em (2) e (3) : $m \leq 20$

Em (2) : $\ell \leq 20$

O que acabamos de supor significa, em outras palavras, que:

$n=100$ é o número máximo de variáveis independentes do problema.

$m=20$ é o número máximo de restrições de desigualdade em (2) e total em (3).

$\ell=20$ é o número máximo de restrições de igualdade em (2).

CARTÕES DE CONTROLE - Na implementação do sistema procuramos adotar medidas que facilitassem, ao máximo, a tarefa do usuário. Para tanto criamos uma "procedure" intitulada OTIMIZAR que engloba a maior parte de cartões de controle, restando alguns que, pela própria natureza do sistema operacional do computador, não podem ser nela incluídos. A seguir analisaremos todos eles, separadamente, na ordem em que devem ser colocados na leitora.

1. CARTÃO JOB - É o cartão JOB comum e seu formato é, por exemplo, o seguinte:

```
//NOME JOB (identidade),MSGLEVEL=1,CLASS=G,TIME=3.
```

2. CARTÃO EXEC - Este é um cartão especial do programa pois através dele ordenamos a execução da "procedure" OTIMIZAR. Seu formato é:

```
//NOME EXEC OTIMIZAR
```

3. CARTÃO DD - É o cartão DD comum. Após este cartão são colocadas as subrotinas, fornecidas pelo usuário, referentes à função (ou funções) e gradiente(s) do problema. Seu formato é:

```
//FORT.SYSIN DD *
```

4. CARTÃO DELIMITADOR - Indica o fim das subrotinas de entrada. Seu formato:

```
/*
```

5. CARTÃO "GO" - É o cartão "GO" comum. Indica o início da etapa de execução. Em seguida a este cartão devem vir os dados. Seu formato:

```
//GO.SYSIN DD *
```

6. CARTÃO DELIMITADOR - Indica o fim dos dados. Seu formato:

```
/*
```

São pois, no total 6 cartões de controle que acreditaremos não constituir qualquer dificuldade ao usuário. A seguir damos uma idéia de conjunto dos cartões de controle:

```
//NOME JOB (identidade),MSGLEVEL=1,CLASS=G,TIME=3
```

```
//NOME EXEC OTIMIZAR
```

```
//FORT.SYSIN DD *
```

4 - subrotinas

```
/*
```

```
//GO.SYSIN DD *
```

- dados

```
/*
```

SUBROTINAS DE ENTRADA - Existem, ao todo, 5 subrotinas a serem fornecidas pelo usuário, sendo incluídas em cada execução apenas aquelas exigidas pelo tipo de problema.

Antes, porém, de as descrevermos definamos dois conjuntos de cartões A e B conforme abaixo:

5 CONJUNTO A - É composto de 2 cartões de formato fixo, usados

em todas as subrotinas de entrada, na ordem em que se apre
sentam:

1. IMPLICIT REAL*8 (A-H,O-Z)
2. COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),HI(20),
FX

6 CONJUNTO B - É também composto de 2 cartões por demais conhe
cidos:

1. RETURN
2. END

Vejaçmos agora as subrotinas.

7 FUNCF - Contem a expressão analítica da função objetivo e
calcula o seu valor. VX contem o ponto e FX o valor da
função no ponto:

1. SUBROUTINE FUNCF
2. Conjunto A
3. $FX = VX(1)*VX(1)+VX(2)*VX(2)+...$
4. Conjunto B

8 GRADF - Contém a expressão analítica do gradiente da função
objetivo e calcula o seu valor. VX contém o ponto e GX o
valor do gradiente no ponto:

1. SUBROUTINE GRADF
2. Conjunto A
3. $GX(1)=2.*VX(1)+...$
4. $GX(2)=2.*VX(2)+...$
5. \vdots
5. Conjunto B

- 9 FUNGD - Contêm as expressões analíticas dos vínculos de desigualdade e calcula seus valores. VX contém o ponto e GD o valor das funções:
1. SUBROUTINE FUNGD
 2. Conjunto A
 3. $GD(1) = VX(1)*VX(1) - VX(2)*VX(2) + \dots$
 4. $GD(2) = VX(1)*VX(1)*VX(1) - VX(2)*VX(2)*VX(2) + \dots$
 5. \vdots
 5. Conjunto B
- 10 FUNHI - Contêm as expressões analíticas dos vínculos de igualdade e calcula seus valores. VX contém o ponto e HI o valor das funções:
1. SUBROUTINE FUNHI
 2. Conjunto A
 3. $HI(1) = VX(1)*VX(2) + VX(2)*VX(2) + \dots$
 4. $HI(2) = VX(1)*VX(2) - VX(2)*VX(2) + \dots$
 5. \vdots
 5. Conjunto B
- 11 GRADV - Contêm as expressões analíticas dos gradientes dos vínculos de desigualdade e igualdade e calcula seus valores. VX contém o ponto, GG o gradiente dos vínculos de desigualdade e GH dos vínculos de igualdade.
1. SUBROUTINE GRADV
 2. Conjunto A
 3. $GG(1,1) = 2.*VX(1) + \dots$
 4. $GG(1,2) = -2.*VX(2) + \dots$
 5. \vdots
 5. $GG(2,1) = 3.*VX(1)*VX(1) + \dots$

6. $GG(2,2) = -3.*VX(2)*VX(2)+...$
 \vdots
 \vdots
7. $GH(1,1) = VX(2)+...$
8. $GH(1,2) = VX(1)+2.*VX(2)+...$
 \vdots
 \vdots
9. $GH(2,1) = VX(2)+...$
10. $GH(2,2) = VX(1)-2.*VX(2)+...$
 \vdots
 \vdots
11. Conjunto B.

DADOS - Para cada execução do programa existem basicamente , três conjuntos de dados a serem fornecidos pelo usuário. São colocados após o 5º cartão de controle, na ordem em que serão apresentados a seguir:

- 12 Informações gerais para o programa - Estes dados, num total de 10, são todos perfurados em um único cartão. Todos eles são variáveis inteiras lidas no formato I5. O ajuste é a direita dos respectivos campos. São eles:

11. Indica o tipo de problema. É uma informação obrigatória. Seus valores estão no quadro 1.
12. Indica o algoritmo que se quer usar. Seus valores estão indicados no quadro 3. Quando o usuário deseja que o programa escolha o algoritmo deve colocar 0 ou 1. Zero é desvinculado e 1 vinculado.
13. Indica a busca escolhida. Seus valores podem ser encontrados no quadro 4. Zero ou um devem ser os valores colocados quando se deseja que o algoritmo usado seja determinado pelo programa. Zero é sem derivadas e um com

derivadas. Para programação linear, zero significa que não há restrição de sinal e um que há restrição de sinal em X .

- I4. Indica o algoritmo para resolver o subproblema do método de penalidades. Colocar zero se é deixado ao programa a liberdade de escolher o método. Para programação linear, esta variável contém o número de restrições de $\leq (m_1)$ (ver 16).
- I5. Indica o número de variáveis ao programa. Depende do problema. É uma informação obrigatória para o programa.
- I6. Indica o número de vínculos de desigualdade em problemas vinculados. É uma informação obrigatória para o programa. Para métodos desvinculados colocar zero. Para programação linear esta variável contém o número de restrições de $\geq (m_3)$ (ver 16).
- I7. Indica o número de vínculos de igualdade em problemas vinculados. É também uma informação obrigatória para o programa. Para métodos desvinculados colocar zero. Para programação linear esta variável contém o número de restrições de $= (m_2)$ (ver 16).
- I8. Colocar 1 para a impressão de resultados intermediários e 0 para resultados finais. Em métodos vinculados colocar 2 para resultados intermediários apenas do algoritmo principal.

- I9. Representa o número máximo de iterações. Se por exemplo o valor fornecido for igual a 100 a execução é interrompida automaticamente quando o algoritmo atinge este número de iterações. Se zero for o valor colocado o programa assume 100. Para programação linear esta variável conterà 1 se for minimização e 0 se for maximização (ver 16).
- Il0. Deve ser 0 ou 1. Se for 1 significa que o mesmo problema será resolvido com outros parâmetros e o programa, após a execução retornará à leitura de novos dados. Se for zero o programa "entende" que aquele é o último problema a resolver.

A seguir são apresentados os quadros 1,2,3,4 e 5 .

São os seguintes os respectivos conteúdos:

- . Quadro 1: valores da variável I1.
- . Quadro 2: valores da variável I2.
- . Quadro 3: valores da variável I3.
- . Quadro 4: Resumo dos três anteriores. O primeiro número (I1) indica o tipo do método. O segundo número indica o algoritmo, ou seja, valor I2. O terceiro número (I3) indica a busca. Os quadriculos com X indicam a não existência daquela configuração. O sinal (?) significa que dependente do algoritmo desvinculado escolhido pelo usuário.
- . Quadro 5: Apresenta um resumo geral dos valores das variáveis, I1,I2,...,Il0. O sinal (?) significa que o valor depende do problema.

VALORES DE I1					
"DEFAULT"	MDCD	MDSO	MVCD	MVSD	PL
0	1	2	3	4	5

QUADRO 1.

MDCD = métodos desvinculados com derivadas

MDSO = métodos desvinculados sem derivadas

MVCD = métodos vinculados com derivadas

MVSD = métodos vinculados sem derivadas

VALORES DE I2										
MDCD				MDS D			MVCD		MVSD	PL
Davidon-Fletcher-Powell	Broyden	Fletcher-Reeves	Cauchy	Cauchy	Powell	Nelder-Mead	Penalidades ^o	Direções Viáveis	Tolerância Flexível	
1	2	3	4	1	2	3	1	2	1	1

Quadro 2

VALORES DE I3				
Buscas				
Nenhuma	Goldstein	Armijo	Secção-Áurea	Davies-Swann-Campey-Powell
0	1	2	3	4

Quadro 3

		MDCD				MDS D			MVCD		MVSD	
Algoritmos Buscas	Davidon Fletcher Powell	Broyden	Fletcher Reeves	Cauchy	Cauchy	Powell	Nelder Mead	Penali dades	Direções Viáveis	Tolerân cia Fle xivel	PL	
	Goldstein	1-1-1	1-2-1	1-3-1	1-4-1	2-1-1	X	2-3-0	3-1-1-?	3-2-1	4-1-0	5-1-0
Armijo	1-1-2	1-2-2	1-3-2	1-4-2	2-1-2	X	2-3-0	3-1-2-?	3-2-2	4-1-0	5-1-0	
Secção- Áurea	1-1-3	1-2-3	1-3-3-	1-4-3	2-1-3	2-2-3	2-3-0	3-1-3-?	3-2-3	4-1-0	5-1-0	
Davies- Swann- Campey- Powell	1-1-4	1-2-4	1-3-4	1-4-4	2-1-4	2-2-4	2-3-0	3-1-4-?	3-2-4	4-1-0	5-1-0	

Quadro 4

I casos	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10
"DEFAULT"	0	0-1	0-1	0	?	?	?	0-1	?	0-1
MDCD	1	1-4	1-4	0	?	0	0	0-1	?	0-1
MDSD	2	1-3	1-4	0	?	0	0	0-1	?	0-1
MVCD	3	1-2	1-4	1-4	?	?	?	0-1	?	0-1
MVSD	4	1	0	0	?	?	?	0-1	?	0-1
PL	5	1	0-1 sinal	? m ₁	? n	? m ₃	? m ₂	0-1	0-1	0-1

Quadro 5

OBSERVAÇÕES:

. O sinal (?) significa que o valor depende do problema .

Um hífen entre dois números substitui a preposição a .

Por exemplo 1-4 quer dizer: 1 a 4.

- 13 PRECISÕES - São 4 valores também perfurados em um único cartão. São variáveis reais lidas no formato D20.13:
- Z1. Precisão em R^n . É um valor escolhido pelo usuário em função do seu problema. Corresponde ao zero computacional para a norma de um vetor, isto é, considera-se $X=Y$ se $\|X-Y\| \leq Z1$.
- Z2. Precisão para a função objetivo. Também é um valor que depende do problema. Corresponde ao zero computacional para comparação de valores do critério.
- Z3. Precisão para as buscas unidirecionais. A Unidade é a mesma de Z1. Apenas o valor fica a critério do usuário (ver capítulo VIII).
- Z4. Precisão para o gradiente da função objetivo. O valor fica a critério do usuário, e corresponde ao zero computacional para valores do gradiente do critério.
- 14 PONTO INICIAL - São perfurados 4 componentes do vetor, por cartão, no formato D20.13. Num problema com 8 variáveis, por exemplo, são utilizados 2 cartões para a entrada do ponto inicial.
- 15 IDENTIFICAÇÃO DE ALGORITMOS E BUSCAS - Os quadros 2 e 3 fornecem os números atribuídos respectivamente aos algoritmos e às buscas. O quadro 4 estabelece o interrelacionamento entre os dois anteriores.

- 16 PROGRAMAÇÃO LINEAR - Este é um caso especial e, por isto, é analisado separadamente.

Conforme vimos (12), os valores de n, m_1, m_2 e m_3 são colocados nos campos do cartão correspondentes, respectivamente, às variáveis I5, I4, I7 e I6, enquanto que a indicação da existência (1) ou não (0) de restrição de sinal é colocada na variável I3 e se é problema de minimização (1) ou maximização (0), na variável I9.

A leitura da matriz de restrições é feita como se segue. O problema a ser resolvido é:

17
$$\text{Min } C_1' X_0$$

sujeito a

$$D_1 X_0 \geq b_1, D_1, (m_1, n)$$

$$D_2 X_0 = b_2, D_2, (m_2, n)$$

$$D_3 X_0 \geq b_3, D_3, (m_3, n)$$

$$X_0 \geq 0$$

O problema (17) pode ser reescrito, sem particularizarmos:

18
$$\text{Min } C_1' X_0$$

sujeito a

$$D_1 X_0 \leq b_1$$

$$D_3 X_0 \geq b_3$$

$$D_2 X_0 = b_2$$

$$X_0 \geq 0$$

Consideremos, de (18), a matriz

$$19 \quad D = (d_1, d_2, d_3, \dots, d_n) = \begin{bmatrix} D_1 \\ D_3 \\ D_2 \end{bmatrix} \text{ e o vetor } b = \begin{bmatrix} b_1 \\ b_3 \\ b_2 \end{bmatrix}$$

onde

$$d_i \in R^m, \quad i=1,2,\dots,n$$

$$b \in R^m$$

$$\text{com } m = m_1 + m_2 + m_3$$

Definamos agora a matriz de restrições A , $(m+1, n+1)$, cuja primeira coluna é o vetor $\begin{bmatrix} b \\ 0 \end{bmatrix}$ e última linha o vetor $(0, c_1)$. Desenvolvendo a matriz A temos:

$$20 \quad A = (a_1, a_2, \dots, a_{n+1}) = \begin{bmatrix} b_1 & d_{11} & d_{12} & \dots & d_{1n} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ b_m & d_{m1} & d_{m2} & \dots & d_{mn} \\ 0 & c_1 & c_2 & \dots & c_n \end{bmatrix}$$

onde

$$a_i \in R^{m+1}, \quad i=1,2,\dots,n+1$$

$$a_1 = \begin{bmatrix} b \\ 0 \end{bmatrix}, \quad b \in R^m \text{ e } 0 \in R$$

$$a_j = \begin{bmatrix} d_{j-1} \\ c_{j-1} \end{bmatrix}, \quad d_{j-1} \in R^m, c_{j-1} \in R, \quad j=2,3,\dots,n+1$$

A leitura da matriz A é feita por colunas como em (14), isto é, perfurando-se quatro componentes do vetor

a_i , $i=1,2,\dots,n+1$, por cartão, obedecendo-se o formato - D20.13. Nunca é demais repetir que as primeiras m linhas da matriz A devem obedecer à disposição $\leq, \geq, =$, rigorosamente.

21 EXEMPLOS

Consideremos, como exemplos, os seguintes problemas:

22 Problema 1 - $\text{Min } f(X) = x_1^2 + x_2^2 + 1$

23 Problema 2 - $\text{Min } f(X) = x_1^2 + x_2^2 + 1$

sujeito a

$$g_1(X) = x_1^2 + x_1 x_2 \leq 0$$

$$g_2(X) = x_1 x_2 + x_2^2 \leq 0$$

$$h_1(X) = x_1^2 - x_1 x_2 = 0$$

$$h_2(X) = x_1 x_2 - x_2^2 = 0$$

Em (22) e (23):

. Ponto inicial: $X_0 = (10, 10)$

.. Precisões: $Z_1 = 10^{-6}, Z_2 = 10^{-6}, Z_3 = 1, Z_4 = 10^{-3}$

24 Problema 3 - $\text{Min } C'X = x_1 - 2x_2 + 3x_3$

sujeito a

$$3x_1 + x_2 - 5x_3 \leq 30$$

$$4x_1 + x_2 = 20$$

$$x_1 - 2x_2 + 2x_3 \geq 10$$

$$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0$$

Mostraremos dois exemplos para o problema 1, dois para o problema 2 e um para o problema 3.

25 EXEMPLO 1 - Resolver o problema (1) pelos métodos de FLETCHER-REEVES e BROYDEN usando, respectivamente as buscas de GOLDSTEIN e DAVIES-SWANN-CAMPEY-POWELL.

A codificação será:

```
//NOME JOB (identidade),MSGLEVEL=1,CLASS=G,TIME=3
//NOME EXEC OTIMIZAR
//FORT.SYSIN DD *
  SUBROUTINE FUNCF
    IMPLICIT REAL*8(A-H,O-Z)
    COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),HI(20),FX
    FX = VX(1)*VX(1)+VX(2)*VX(2)+1.
    RETURN
  END
  SUBROUTINE GRADF
    IMPLICIT REAL*8(A-H,O-Z)
    COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),HI(20),FX
    GX(1)=2.*VX(1)
    GX(2)=2.*VX(2)
    RETURN
  END
/*
//GO.SYSIN DD *
  1   3   1   0 | 2   0   0   1 | 50   1 |
      0.000001 | 0.000001 | 1.   | 0.001 |
      10.      | 10.      |     |     |
  1   2   4   0 | 2   0   0   1 | 50   0 |
      0.000001 | 0.000001 | 1.   | 0.001 |
      10.      | 10.      |     |     |
                                20      40      60      80
                                COLUNAS
```

EXEMPLO 2 - Resolver o problema 1 pelos métodos de POWELL e NELDER-MEAD, usando a busca de secção-âurea.

NOTA: O método de NELDER-MEAD não usa busca unidirecionais.

A codificação será:

```
//NOME JOB (identidade),MSGLEVEL=1,CLASS=G,TIME=3
//NOME EXEC OTIMIZAR
//FORT.SYSIN DD *
  SUBROUTINE FUNCF
    IMPLICIT REAL*8(A-H,O-Z)
    COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),HI(20),FX
    FX = VX(1)*VX(1)+VX(2)*VX(2)+1.
    RETURN
  END
/*
//GO.SYSIN DD *
  2   2   3   0 | 2   0   0   1 | 50   1 |
      0.000001 | 0.000001 | 1.   | 0.001 |
      10.      | 10.      |     |     |
  2   3   0   0 | 2   0   0   0 | 50   0 |
      0.000001 | 0.000001 | 1.   | 0.001 |
      10.      | 10.      |     |     |
                                20   40   60   80
                                COLUNAS
```

27 EXEMPLO 3 - Resolver o problema 2 pelo método de PENALIDADES usando o método de CAUCHY para resolver o subproblema, e a busca de ARMIJO.

A codificação será:

```
//NOME JOB (identidade),MSGLEVEL=1,CLASS=G,TIME=3
//NOME EXEC OTIMIZAR
//FORT.SYSIN DD *
  SUBROUTINE FUNCF
    IMPLICIT REAL*8(A-H,O-Z)
    COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),HI(20),FX
    FX = VX(1)*VX(1)+VX(2)*VX(2)+1.
    RETURN
  END
  SUBROUTINE GRADF
    IMPLICIT REAL*8(A-H,O-Z)
    COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),HI(20),FX
    GX(1) = 2.*VX(1)
    GX(2) = 2.*VX(2)
    RETURN
  END
  SUBROUTINE FUNGD
    IMPLICIT REAL*8(A-H,O-Z)
    COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),HI(20),FX
    GD(1) = VX(1)*VX(1)+VX(1)*VX(2)
    GD(2) = VX(1)*VX(2)+VX(2)*VX(2)
    RETURN
  END
  SUBROUTINE FUNHI
    IMPLICIT REAL*8(A-H,O-Z)
    COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),HI(20),FX
    HI(1) = VX(1)*VX(1)-VX(1)*VX(2)
    HI(2) = VX(1)*VX(2)-VX(2)*VX(2)
    RETURN
  END
```

```

SUBROUTINE GRADV
IMPLICIT REAL*8(A-H,O-Z)
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),HI(20),FX
GG(1,1) = 2.*VX(1)+VX(2)
GG(1,2) = VX(1)
GG(2,1) = VX(2)
GG(2,2) = VX(1)+2.*VX(2)
GH(1,1) = 2.*VX(1)-VX(2)
GH(1,2) = -VX(1)
GH(2,1) = VX(2)
GH(2,2) = VX(1)-2.*VX(2)
RETURN
END

```

```

/*
//GO.SYSIN DD *

```

```

      3      1      2      4      |      2      2      2      1      |      50      0      |
      0.000001      |      0.000001      |      1.      |      0.001      |
      10.      |      10.      |      |      |
/*
      10      40      60      80
      COLUNAS

```

28 EXEMPLO 4 - Resolver o problema 2 pelo método de TOLERÂNCIA FLEXIVEL.

A codificação será:

```

//NOME JOB (identidade),MSGLEVEL=1,CLASS=G,TIME=3
//NOME EXEC OTIMIZAR
//FORT.SYSIN DD *
SUBROUTINE FUNCF
IMPLICIT REAL*8(A-H,O-Z)

```


CAPÍTULO XICONCLUSÕES E SUGESTÕES

Os resultados que obtivemos na resolução de problemas clássicos, através dos algoritmos implementados parecem promissoras embora tenhamos razões para acreditar que sua aplicação a problemas reais possa trazer resultados ainda mais significativos. Realmente, a manipulação de precisões, abordada no capítulo VIII, é de alta relevância em problemas práticos em face de considerações fundamentais sobre as Unidades Físicas em que o problema é formulado, quanto a precisões, regras de parada, etc.

Dentre os testes efetuados apresentaremos no quadro 1 os resultados obtidos na minimização desvinculada das funções abaixo, pelos métodos de DAVIDON-FLETCHER-POWELL e de FLETCHER-REEVES, utilizando a busca unidirecional de DAVIES-SWANN-CAMPEY - POWELL.

Função 1 - $f(X) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$

Ponto inicial: $X_0 = (-1.2, 1)$

Função 2 - $f(X) = 100(x_2 - x_1^3)^2 + (1 - x_1)^2$

Ponto inicial: $X_0 = (-1.2, 1)$

Ponto de ótimo: $X^* = (1.0, 1.0)$

Em comparação com os resultados apresentados em [1], obteve-se uma significativa diferença entre os números de iterações dos algoritmos, embora o número de cálculos da função esteja em desvantagem. Acreditamos contudo que pequenas alterações na implementação das buscas podem reduzir sensivelmente o total de cálculos da função. Tais alterações deverão ser efetuadas doravante obedecendo a um processo de evolução requerido pela própria natureza do trabalho.

Nos quadros que se seguem, os símbolos usados têm o seguinte significado:

DFP = método de DAVIDON-FLETCHER-POWELL

FR = método de FLETCHER-REEVES

F = funções

X_0 = ponto inicial

NI = número de iterações do algoritmo

NF = número de vezes que a função foi calculada

X^* = ponto de ótimo

$f(X^*)$ = ótimo da função

N = método de Newton

P = método de penalidades

DV = método de direções viáveis

SUMT = sequential unconstrained minimization technique

TL = tolerância flexível

		DFP				FR			
F	X_0	NI	NF	X^*	$f(X^*)$	NI	NF	X^*	$f(X^*)$
1	-1.2 1.0	12	362	0.999995 0.999990	3.6×10^{-11}	11	279	0.999992 0.999985	5×10^{-11}
2	-1.2 1.0	13	452	1.000001 0.999980	1.0×10^{-9}	12	397	0.999067 0.996882	1×10^{-6}

Quadro 1

No quadro 2, a seguir repetimos os resultados apresentados em [1], para efeito comparativo. Como não há elementos sobre o método de FLETCHER-REEVES fizemos sua substituição pelo método de NEWTON.

		DFP				N			
F	X_0	NI	NF	X^*	$f(X^*)$	NI	NF	X^*	$f(X^*)$
1	-1.2 1.0	23	120	-	1×10^{-12}	17	98	-	3×10^{-13}
2	-1.2 1.0	21	120	-	3×10^{-12}	25	127	-	4×10^{-15}

Quadro 2

Para métodos vinculados apresentamos no quadro 3 a seguir os resultados obtidos pela aplicação dos métodos de penalidades e direções viáveis ao seguinte problema

$$\text{Min } f(X) = 4x_1 - x_2^2 - 12$$

$$\text{sujeito a } g_1(X) = x_1^2 - 10x_1 + x_2^2 - 10x_2 + 34 \leq 0$$

$$g_2(X) = -x_1 \leq 0$$

$$g_3(X) = -x_2 \leq 0$$

$$g_4(X) = x_1^2 + x_2^2 - 25 \leq 0$$

$$h_1(X) = x_1^2 + x_2^2 - 25 = 0$$

Ponto inicial: $X_0 = (1,1)$

Ponto de ótimo: $X^* = (1.001, 4.898)$

OBSERVAÇÕES: No método de direções viáveis não foi incluída a restrição de igualdade. O subproblema do método de penalidades foi resolvido pelo algoritmo de Davidon-Fletcher-Powell e a busca utilizada foi de secção-áurea para os dois métodos.

P				* DV		
X_0	NI	X^*	$f(X^*)$	NI	X^*	$f(X^*)$
1.0	9	1.0009	-31.98	7	1.0021	-31.93
1.0		4.8985			4.8929	

Quadro 3

* - ver página seguinte.

Os resultados de |1|, para o mesmo problema podem ser vistos no quadro 4.

SUMT				** TL		
X_0	NI	X^*	$f(X^*)$	NI	X^*	$f(X^*)$
1.0	23	1.073	-31.80	23	1.001	-31.99
1.0		4.909			4.899	

Quadro 4

* - O método de direções viáveis em nossa implementação requer ainda que o ponto inicial seja viável e o ponto inicial para os resultados do quadro 3 foi $X_0=(2.5,3.5)$. A modificação para que X_0 seja qualquer está sendo efetuada mas ainda não se encontrava completa quando da obtenção dos resultados acima.

** - O método de tolerância flexível de nosso sistema resolveu o mesmo problema em 9 iterações com $X^*=(1.050,4.888)$ e $f(X^*)=-31.66$. É bom esclarecer que em |1|, os mesmos resultados foram obtidos em 9 iterações.

Evidentemente os resultados apresentados acima não expressam o real desempenho dos algoritmos implementados e nem mesmo podem servir de base para conclusões definitivas em relação a comparações com outros resultados existentes na literatura. Vários outros testes necessitam ser feitos e é nosso pensamento efetuá-los oportunamente. Aí então poderemos tirar conclusões mais concretas sobre nosso sistema.

Durante a realização deste trabalho observamos alguns fatos que poderão ser estudados ou pesquisados por interessados na área:

- a) Existem outros métodos de busca unidirecional (ver [4]) cuja implementação seja talvez interessante e apresenta resultados compensadores.
- b) Conforme comentamos em (V.3) o método de NELDER-MEAD não utiliza buscas unidirecionais. Um estudo interessante seria talvez adaptá-lo ao uso de buscas conseguindo-se assim uma possível melhoria em seu desempenho.
- c) O subproblema do método de Tolerância Flexível, em nosso caso, é resolvido pelo processo de NELDER-MEAD. Uma sugestão seria aplicar os métodos de Powell e Cauchy sem derivadas na solução desse subproblema.
- d) O tratamento de precisões é um assunto altamente importante no estudo da rapidez de convergência de algoritmos e achamos que uma pesquisa na área poderia trazer importantes contribuições à rapidez de convergência inicial dos algoritmos.

CAPÍTULO XII

LISTAGENS DOS PROGRAMAS

```

//CARUX JOB (3032,2410),MSGLEVEL=1,CLASS=C,TIME=5
C**** PROCEDURE
// EXEC PGM=IEBUPDTE
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=SYS1.PROCLIB,DISP=SHR
//SYSUT2 DD DSN=SYS1.PROCLIB,DISP=SHR
//SYSIN DD DATA
./ REPL NAME=OTIMIZAR,LIST=ALL
./ NUMBER NEW1=10,INCR=10
//FORT EXEC PGM=IEYFCRT,REGION=100K
//SYSPRINT DD SYSOUT=A
//SYSPUNCH DD SYSOUT=B
//SYSLIN DD DSNAME=&LOADSET,DISP=(MGD,PASS),UNIT=SYSSQ,
// SPACE=(80,(200,100),RLSE),DCB=BLKSIZE=80
//LKED EXEC PGM=IEWL,REGION=96K,COND=(4,LT,FORT)
//SYSLIB DD DSNAME=SYS1.FCRTLIB,DISP=SHR
// DD DSN=CLOVIS,DISP=SHR
//SYSLMOD DD DSNAME=&GCSET(MAIN),DISP=(NEW,PASS),UNIT=SYSDA,
// SPACE=(1024,(20,10,1),RLSE),DCB=BLKSIZE=1024
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10),RLSE),DCB=BLKSIZE=1024,
// DSNAME=&SYSUT1
//SYSLIN DD DSN=PROCC,DISP=SHR
// DD DSN=&LOADSET,DISP=(OLD,DELETE)
//GO EXEC PGM=*.LKED.SYSLMOD,COND=((4,LT,FORT),(4,LT,LKED))
//GO.FT05F001 DD SYSOUT=A,DCB=(LRECL=133,BLKSIZE=133,RECFM=FA)
//GO.FT06F001 DD SYSOUT=A
//FT08F001 DD DDNAME=SYSIN
//GO.FT20F001 DD DSN=&&CLOVIS,DISP=(NEW,PASS),UNIT=SYSDA,
// SPACE=(8,(30000,10))
/*
//CLOVIS EXEC FORTGC
//FORT.SYSLIN DD DSN=PROCC,UNIT=2314,VOL=SER=LIX001,DISP=(NEW,CATLG),
// DCB=BLKSIZE=80,SPACE=(80,(200,100),RLSE),LABEL=EXPDT=75365
/*
C**** PROGRAMA CONTROLADOR
IMPLICIT REAL*8(A-H,O-Z)
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
C**** AS VARIÁVEIS V1 A V7 E J1 A J7 FORAM INCLUIDAS PARA
C**** USO FUTURO. NÃO FORAM USADAS EM PARTE ALGUMA.
DEFINE FILE 20(30000,2,U,IO)
IN=8
IO=5

```

```

I=0
10 CONTINUE
I=I+1
WRITE(10,1) I
1 FORMAT(//1X,130('*')//60X,'PROBLEMA NO. ',15//)
FI=0.
FL=0.
FH=0.
IT=0
KL=0
NI=0
NU=0
NV=0
NX=0
ID=0
NF=0
IK=0
ND=0
IC=0
EP=1000.
C**** LEITURA VERIFICACAO E IMPRESSAO DOS DADOS
CALL LEIAS
IF(NV.GT.0) RETURN
IF(JN.LT.0..OR.JN.GT.2) JN=0
IF(ZR.GT.EP.OR.ZR.EG.0.) ZR=EP
DO 20 K=1,NC
VD(K)=0.
VE(K)=0.
VF(K)=0.
20 CONTINUE
IF(NK.LE.0) NK=100
C**** CONTROLE PASSADO AO ALGORITMO ESCOLHIDO PELO USUARIO
CALL SELE1
IF(ID.NE.7) GO TO 30
ID=2
CALL IMPRS
30 CONTINUE
CALL IMPRS
40 IF(NP.EG.1) GO TO 10
CALL EXIT
END

/*
//CLOVIS EXEC FORTGCL,PARM.LKED='NCAL,LET'
//FORT.SYSIN DD *
SUBROUTINE LEIAS
IMPLICIT REAL*8(A-H,G-Z)
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,

```

```

4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KB,LF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7

```

```

C**** FAZ A LEITURA DOS DADOS INICIAIS

```

```

C**** ID=1: DESVINCULADO COM DERIVADA

```

```

C**** ID=2: DESVINCULADO SEM DERIVADA

```

```

C**** ID=3: VINCULADO COM DERIVADA

```

```

C**** ID=4: VINCULADO SEM DERIVADA

```

```

C**** ID=5: PROGRAMACAO LINEAR

```

```

WRITE(IO,11)

```

```

11 FORMAT(///1X,130('-'))//20X,'DADOS FORNECIDOS PELO USUARIO'//)

```

```

READ(IN,12,ERR=30) ID,N1,N2,N3,NC,NG,NH,JN,NK,NP

```

```

WRITE(IO,12) ID,N1,N2,N3,NC,NG,NH,JN,NK,NP

```

```

MX=ID

```

```

NA=N1

```

```

NB=N2

```

```

12 FORMAT(10I5)

```

```

READ(IN,13,ERR=30) ZN,ZF,ZR,ZE

```

```

13 FORMAT(4D20.13)

```

```

WRITE(IO,15) ZN,ZF,ZR,ZE

```

```

15 FORMAT(1X,4D20.13)

```

```

READ(IN,13,ERR=30) (VX(I),I=1,NC)

```

```

WRITE(IO,15) (VX(I),I=1,NC)

```

```

IF(ID.NE.5) CALL CRIVC

```

```

IF(NV.EQ.1) RETURN

```

```

IF(NV.EQ.0) RETURN

```

```

30 WRITE(IO,14)

```

```

14 FORMAT(//20X,'SK. USUARIO,'//20X,'REEXAMINE COM ATENCAO OS ',

```

```

1'CARTOES DE DADOS. CONSTATADA A EXISTENCIA DE ERRO(S) ',

```

```

2'NAQUELAS INFORMACOES.')
```

```

NV=1

```

```

RETURN

```

```

END

```

```

//LKED.SYSLMOD DD DSN=CLGVIS(LEIAS),DISP=(NEW,CATLG),UNIT=2314,

```

```

// VOL=SER=LIX001,SPACE=(CYL,(2,2,15),,MXIG),DCB=BLKSIZE=7294,

```

```

// LABEL=EXPDT=75365

```

```

//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'

```

```

//FORT.SYSIN DD *

```

```

SUBROUTINE IMPRS

```

```

IMPLICIT REAL*8(A-H,O-Z)

```

```

COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),

```

```

1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),

```

```

2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,

```

```

3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,

```

```

4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),

```

```

5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,

```

```

6NS,N1,N2,N3,N4,IC,IA,IB,KB,LF,NK,NP,NU,NV,NX,IT,JN,

```

```

7MX,J1,J2,J3,J4,J5,J6,J7

```

```

C**** IMPRIME RESULTADOS E MENSAGENS

```

C****

```

IF(JN.EQ.2.AND.IT.EQ.2) RETURN
IF(ID.GT.4) GO TO 5
IB=NU-NX
IF(IT.EQ.1) WRITE(IO,1)
IF(IT.EQ.2) WRITE(IC,2)
1 FORMAT(/1X,130('-')//10X,'* PROBLEMA PRINCIPAL */)
2 FORMAT(/1X,130('-')//10X,'* SUBPROBLEMA */)
5 GO TO (10,20,30,40,50,60,70,80,90,100),ID

```

C****

C**** VALORES OTIMOS: METODOS DESVINCULADOS

C****

```

10 WRITE(IO,11) (VX(I),I=1,NC)
11 FORMAT(10X,'* SOLUCAO DO PROBLEMA *///10X,
1'PONTO DE OTIMO:*/(6(2X,D20.13)))
IF(IK.EQ.1) WRITE(IO,12) (GX(I),I=1,NC)
12 FORMAT(/10X,'GRADIENTE DA FUNCAO OBJETIVO: ',
1//6(2X,D20.13)))
WRITE(IO,13) FX,NI,NF
13 FORMAT(/10X,'VALOR OTIMO DA FUNCAO OBJETIVO: ',D20.13//10X,
1'TOTAL DE ITERACOES DO ALGORITMO PRINCIPAL: ',I10//10X,
2'TOTAL DE CALCULOS DA FUNCAO OBJETIVO: ',I10)
IF(NU.GT.0) WRITE(IC,14) NU
14 FORMAT(/10X,'TOTAL DE ITERACOES DA BUSCA ',
1'UNIDIRECIONAL: ',I10)
GO TO 120

```

C****

C**** VALORES INTERMEDIARIOS: METODOS DESVINCULADOS

C****

```

20 CONTINUE
IF(JN.EQ.2) RETURN
IF(IK.NE.1) GO TO 25
ID=3
CALL NORMA
25 WRITE(IO,21) NI
21 FORMAT(10X,'* VALORES INTERMEDIARIOS *///10X,
1'ITERACAO NC.:',I10)
WRITE(IO,22) (VX(I),I=1,NC)
22 FORMAT(/10X,'VETOR X*/(6(2X,D20.13)))
IF(IK.EQ.1) WRITE(IO,12) (GX(I),I=1,NC)
IF(IK.EQ.1) WRITE(IO,23) XN
23 FORMAT(/10X,'NORMA DO GRADIENTE DA FUNCAO: ',D20.13)
WRITE(IO,24) FX,NF
24 FORMAT(/10X,'VALOR DA FUNCAO OBJETIVO EM X: ',D20.13//10X,
1'TOTAL ACUMULADO DO NO. DE CALCULOS DA ',
2'FUNCAO OBJETIVO: ',I10)
IF(NU.GT.0) WRITE(IO,26) IB,NU
26 FORMAT(/10X,'ITERACOES DA BUSCA UNIDIRECIONAL ',
1'NESTE ESTAGIO: ',I5//10X,'TOTAL ACUMULADO DE ',
2'ITERACOES DA BUSCA UNIDIRECIONAL: ',I10)

```

GO TO 120

C****

C**** VALORES OTIMOS: METODOS VINCULADOS

C****

```

30 WRITE(IO,11) (VX(I),I=1,NC)
   IF(IK.EQ.3) WRITE(IO,12) (GX(I),I=1,NC)
   WRITE(IO,31) FX
31 FORMAT(/10X,'VALOR OTIMO DA FUNCAO OBJETIVO: ',D20.13)
   IF(NH.NE.0) WRITE(IC,32) (HI(I),I=1,NH)
32 FORMAT(/10X,'VINCULOS DE IGUALDADE'//(6(2X,D20.13)))
   IF(NG.NE.0) WRITE(IC,33) (GD(I),I=1,NG)
33 FORMAT(/10X,'VINCULOS DE DESIGUALDADE'//(6(2X,D20.13)))
   IF(NA.EQ.2) NI=0
   WRITE(IO,36) NX,NI,NF
36 FORMAT(/10X,'TOTAL DE ITERACOES DO ALGORITMO PRINCIPAL: ',
1110//10X,'TOTAL DE ITERACOES DO SUBPROBLEMA: ',
2110//10X,'TOTAL DE CALCULOS DA FUNCAO OBJETIVO: ',110)
GO TO 120

```

C****

C**** VALORES INTERMEDIARIOS: METODOS VINCULADOS

C****

```

40 CONTINUE
50 CONTINUE
   IF(IK.NE.3) GO TO 45
   ID=3
   CALL NORMA
45 WRITE(IO,21) NX
   WRITE(IO,22) (VX(I),I=1,NC)
   IF(IK.EQ.3.AND.ND.NE.4) WRITE(IO,41) (GX(I),I=1,NC)
   IF(IK.EQ.3.AND.ND.EQ.4) WRITE(IO,12) (GX(I),I=1,NC)
41 FORMAT(/10X,'GRADIENTE DA FUNCAO PENALIZADA',
1//6(2X,D20.13)))
   IF(IK.EQ.3) WRITE(IC,23) XN
   WRITE(IO,24) FX,NF
   WRITE(IO,42) NI
   IF(NH.NE.0) WRITE(IC,32) (HI(I),I=1,NH)
   IF(NG.NE.0) WRITE(IC,33) (GD(I),I=1,NG)
42 FORMAT(/10X,'TOTAL ACUMULADO DE ITERACOES DO ',
1'SUBPROBLEMA: ',110)
GO TO 120

```

C****

C**** MENSAGENS DE INTERESSE DO USUARIO

C****

```

60 WRITE(IC,61)
61 FORMAT(/20X,'SR. USUARIO,'//20X,'IMPOSSIVEL ',
1'ENCONTRAR UM PONTO MELHOR.')
   NV=1
   RETURN
70 CONTINUE
   WRITE(IO,71)

```

```

71 FORMAT(/20X,'SR. USUARIO,'//20X,'EXECUCAO INTERROMPIDA ',
1'POR HAVER SIDO ATINGIDO O NUMERO DE ITERACOES ',
2'SOLICITADO.'//20X,'ULTIMOS RESULTADOS OBTIDOS:')
RETURN
80 CONTINUE
WRITE(IO,81)
81 FORMAT(/10X,'SR. USUARIO,'//10X,'POSSIVELMENTE ',
1'TRATA-SE DE PROBLEMA CUJA SOLUCAO E ILIMITADA.')
NV=1
RETURN
90 CONTINUE
100 CONTINUE
120 WRITE(IC,121)
121 FORMAT(/1X,120('-'))
RETURN
END

/*
//LKED.SYSLMED DD DSN=CLOVIS(IMPRS),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *
SUBROUTINE CRIVO
IMPLICIT REAL*8(A-H,C-Z)
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),UR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,IX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NF,
6NS,N1,N2,N3,N4,IC,IA,IE,KL,IF,NK,NP,NL,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
C**** TESTA A VALIDADE DOS DADOS FORNECIDOS PELO USUARIO
C**** TOTAL DE METODOS DESVINCULADOS COM DERIVADAS
M1=4
C**** TOTAL DE METODOS DESVINCULADOS SEM DERIVADAS
M2=3
C**** TOTAL DE METODOS VINCULADOS COM DERIVADAS
M3=2
C**** TOTAL DE METODOS VINCULADOS SEM DERIVADAS
M4=1
C**** TOTAL DE BUSCAS UNIDIRECIONAIS
M5=4
M6=MAX0(M1,M2,M3,M4)
CALL FUNCF
IF(ID.EQ.1.OR.ID.EQ.3) CALL GRADF
IF(ID.LE.2) GO TO 40
CALL FUNGD
IF(NA.NE.2) CALL FUNHI
IF(ID.NE.4) CALL GRADV
40 CONTINUE
IF(ID.LT.0.OR.ID.GT.5.OR.NA.LE.0.OR.NA.GT.M6.OR.NB.LT.0.

```

```

1OR,NB,GT,M5,OR,N3,LT,0,OR,N3,GT,M1) NV=2
  IF(ID,EQ,0,AND,NA,LE,0,OR,ID,EQ,0,AND,NA,GT,5) NV=2
  IF(ID,EQ,1,AND,NA,GT,M1,OR,ID,EQ,2,AND,NA,GT,M2,OR,ID,EQ,3,
1AND,NA,GT,M3,OR,ID,EQ,4,AND,NA,GT,M4) NV=2
  IF(NC,LE,0,OR,NG,LT,0,OR,NH,LT,0,OR,NP,LT,0,OR,NP,GT,1,
1OR,ZN,LT,0,OR,ZF,LT,0,OR,ZR,LT,0,OR,ZE,LT,0) NV=2
  IF(ID,GT,2,AND,NG,EQ,0,AND,NH,EQ,0) NV=2
  IF(MX,EQ,3,AND,NA,EQ,2,AND,NH,NE,0) NV=2
  RETURN
  END

```

```
/*
```

```
//LKED,SYSLMOD DD DSN=CLOVIS(CRIVO),DISP=OLD
```

```
//
```

```
//CARIX JOB (3032,2410),MSGLEVEL=1,CLASS=C,TIME=3
```

```
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
```

```
//FORT,SYSIN DD *
```

```
  SUBROUTINE SELE1
```

```
  IMPLICIT REAL*8(A-H,O-Z)
```

```
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KB,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
```

```
C**** SELEÇÃO O ALGORITMO PRINCIPAL ESCOLHIDO PELO USUARIO
```

```
C**** 'DEFAULT'
```

```
  IF(ID,NE,0) GO TO 50
```

```
  ID=NA
```

```
  IF(NA,GT,2) NA=1
```

```
  NB=4
```

```
  N3=1
```

```
C**** SELEÇÃO DO TIPO DE PROBLEMA
```

```
  50 GO TO (100,200,300,400,500),ID
```

```
C**** DESVINCULADOS COM DERIVADAS
```

```
  100 GO TO (110,110,130,140,150,160,170,180),NA
```

```
C**** DESVINCULADOS SEM DERIVADAS
```

```
  200 GO TO (210,220,230,240,250,260),NA
```

```
C**** VINCULADOS COM DERIVADAS
```

```
  300 GO TO (310,320,330,340),NA
```

```
C**** VINCULADOS SEM DERIVADAS
```

```
  400 GO TO (410,420),NA
```

```
C**** METODOS DESVINCULADOS COM DERIVADAS
```

```
  110 CALL DFLEP
```

```
  RETURN
```

```
  130 CALL FRIVS
```

```
  RETURN
```

```
  140 CALL COXIC
```

```
  RETURN
```



```

C**** RESERVADO PARA FUTURAS INCLUSOES
  150 CONTINUE
  160 CONTINUE
  170 CONTINUE
  180 CONTINUE
      RETURN
C**** METODOS DESVINCULADOS SEM DERIVADAS
  210 CALL COXIS
      RETURN
  220 CALL POWEL
      RETURN
  230 CALL NEDMD
      RETURN
C**** RESERVADO PARA FUTURAS INCLUSOES
  240 CONTINUE
  250 CONTINUE
  260 CONTINUE
      RETURN
C**** METODOS VINCULADOS COM DERIVADAS
  310 CALL PENAL
      RETURN
  320 CALL DIRVS
      RETURN
C**** RESERVADO PARA FUTURAS INCLUSOES
  330 CONTINUE
  340 CONTINUE
      RETURN
-----
C**** METODOS VINCULADOS SEM DERIVADAS
  410 CALL FLEXT
      RETURN
C**** RESERVADO PARA FUTURAS INCLUSOES
  420 CONTINUE
      RETURN
C**** PROGRAMACAO LINEAR
  500 CALL CAREX
      RETURN
      END

```

```
/*
```

```

//LKED.SYSLMOD DD DSN=CLOVIS(SELE1),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *

```

```

      SUBROUTINE SELE2
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,ID,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,

```

```

7MX,J1,J2,J3,J4,J5,J6,J7
  IF(NB.EQ.0) GO TO 800
  GO TO (500,600,700,800),NB
500 CALL PLAK1
  RETURN
600 CALL PLAK2
  RETURN
700 CALL G0SEC
  RETURN
800 CALL DSCPW
  RETURN
  END

```

```
/*
```

```

//LKED.SYSLMOD DD DSN=CLOVIS(SELE2),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *

```

```

  SUBROUTINE SELE3
  IMPLICIT REAL*8(A-H,O-Z)
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KB,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7

```

```

  IF(NA.EQ.0) GO TO 20
  GO TO (10,10,20,30,40,50,60),NA
10 CALL DFLEP
  RETURN
20 CALL FRIVS
  RETURN
30 CALL COXIC
  RETURN
40 CALL COXIS
  RETURN
50 CALL POWEL
  RETURN
60 CALL NEDMD
  RETURN
  END

```

```
/*
```

```

//LKED.SYSLMOD DD DSN=CLOVIS(SELE3),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *

```

```

  SUBROUTINE LAMBI
  IMPLICIT REAL*8(A-H,O-Z)
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,

```

```

3FL, FH, TX, EP, E1, E2, P1, P2, Q1, Q2, V1, V2, V3, V4, V5, V6, V7,
4VA(100), VB(100), VC(100), VD(100), PX(100), PY(100),
5IX(20), IN, IO, NI, ID, IK, NC, NB, NA, NG, NR, IL, NT, ND, NH,
6NS, N1, N2, N3, N4, IC, IA, IB, KL, NF, NK, NP, NU, NV, NX, IT, JN,
7MX, J1, J2, J3, J4, J5, J6, J7

```

```

C**** CALCULA O LAMBDA INICIAL PARA QUALQUER METODO

```

```

IA=0
XM=0.01
FY=FX
CALL DIREC
IF(ND.EQ.0.OR.ND.EQ.5) GO TO 10
CALL FUNGD
CALL INDIC
IF(IL.GT.0) GO TO 60

```

```

10 CONTINUE
TE=FY-FX
IF(TE.LE.0.) GO TO 60

```

```

20 CONTINUE
XM=2.*XM
CALL DIREC
IF(ND.EQ.0.OR.ND.EQ.5) GO TO 30
CALL FUNGD
CALL INDIC
IF(IL.GT.0) GO TO 60

```

```

30 CONTINUE
TA=FY-FX
IF(TA.LT.0) GO TO 60
P2=2.*TE-ZF
IF(TA.LE.P2) GO TO 40
TE=TA
GO TO 20

```

```

40 CONTINUE
XM=XM*TE/(2.*TE-TA)
60 CONTINUE
FX=FY

```

```

C**** VERIFICACAO DA VIOLACAO DE VINCULOS

```

```

CALL CRITV

```

```

70 CONTINUE

```

```

C**** ATUALIZACAO DE ZR

```

```

ID=4
CALL NORMA
EP=ZR/XN
IF(XM.GT.EP) RETURN
ZR=XM*XN

```

```

80 CONTINUE
RETURN
END

```

```

/*

```

```

//LKED,SYSLMOD DD DSN=CLOVIS(LAMBI),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'

```

```
//FORT.SYSIN DD *
```

```
  SUBROUTINE CRITV
```

```
  IMPLICIT REAL*8(A-H,O-Z)
```

```
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
 1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
 2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
 3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
 4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
 5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
 6NS,N1,N2,N3,N4,IC,IA,IB,KB,LF,NK,NP,NU,NV,NX,IT,JN,
 7MX,J1,J2,J3,J4,J5,J6,J7
```

```
  IF(ND.EQ.0) GO TO 200
```

```
  GO TO (100,300,300,100,200),ND
```

```
C**** INTERVALO INICIAL PARA PENALIDADES E/OU DIRECOES VIAVEIS
```

```
100 CONTINUE
```

```
  S1=0.
```

```
  S2=0.
```

```
  RO=XM
```

```
  J=0
```

```
  K=0
```

```
  KN=ND
```

```
110 CONTINUE
```

```
  NU=NU+1
```

```
  ND=2
```

```
  CALL DIREC
```

```
  IF(KN.EQ.1) ND=1
```

```
  CALL INDIC
```

```
  IF(IL.EQ.0) GO TO 120
```

```
  RO=0.5*RO
```

```
  XM=XM-RO
```

```
  K=1
```

```
  GO TO 110
```

```
120 CONTINUE
```

```
  FZ=FX
```

```
  CALL FUNCF
```

```
  IF(KN.EQ.1) CALL FUNCP
```

```
  IF(FX.LT.FZ) GO TO 130
```

```
  P1=S1
```

```
  P2=XM
```

```
  ND=KN
```

```
  RETURN
```

```
130 CONTINUE
```

```
  RO=0.5*RO
```

```
  IF(K.EQ.0) RO=XM
```

```
  IF(KN.EQ.4) GO TO 160
```

```
  S1=S2
```

```
  S2=XM
```

```
  XM=XM+RO
```

```
  GO TO 110
```

```
C**** BUSCA UNIDIRECIONAL PARA DIRECOES VIAVEIS
```

```
160 CONTINUE
    DO 170 I=1,NG
    GD(I)=GD(I)+EP
170 CONTINUE
    ND=4
    CALL INDIC
    IF(IC.EQ.0) GO TO 190
    S1=S2
    S2=XM
    RO=0.5*RO
    XM=XM+RO
    GO TO 110
190 N1=IC
    DO 210 I=1,NG
    GD(I)=GD(I)+2.*EP
210 CONTINUE
    CALL INDIC
    N2=IC
    IF(N2.EQ.N1.OR.J.GE.2) GO TO 220
    S1=S2
    S2=XM
    RO=0.5*RO
    XM=XM+RO
    J=J+1
    GO TO 110
220 CONTINUE
    ND=KN
    CALL DIREC
    IA=1
    RETURN
200 CONTINUE
    CALL DIREC
    RETURN
300 IF(TX.LE.FI) GO TO 400
    DO 10 K=1,NC
    VA(K)=DV(K)
    VB(K)=DR(K)
    VC(K)=VY(K)
    VE(K)=HM(K)
    VF(K)=DG(K)
    PY(K)=VZ(K)
10 CONTINUE
    FZ=FL
    FW=FW
    NJ=NI
    CALL VIAVL
    IF(IC.EQ.1) RETURN
    FL=FZ
    FH=FW
    NI=NJ
```

```

N1=NR+1
NT=NC*N1
ND=3
DO 30 K=1,NC
DV(K)=VA(K)
DR(K)=VB(K)
VY(K)=VC(K)
HM(K)=VE(K)
DG(K)=VF(K)
VZ(K)=PY(K)
30 CONTINUE
400 CALL FUNCF
RETURN
END

```

```
/*
```

```
//LKED.SYSLMOD DD DSN=CLOVIS(CRITV),DISP=GLD
```

```
//
```

```
//CARIX JOB (3032,2410),MSGLEVEL=1,CLASS=C,TIME=6
```

```
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
```

```
//FORT.SYSIN DD *
```

```
  SUBROUTINE INDIC
```

```
  IMPLICIT REAL*8(A-H,O-Z)
```

```
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
```

```
C**** ESTABELECE O CONJUNTO DE VINCULOS VIOLADOS.
```

```
C**** SE IX(I)=1 O VINCULO I ESTA' VIOLADO.
```

```
C**** SE IX(I)=0 O VINCULO I NAO ESTA' VIOLADO.
```

```
C**** IC E IL CONTEM O NUMERO DE VINCULOS VIOLADOS.
```

```
  IL=0
```

```
  IF(ND.NE.4) CALL FUNGD
```

```
  IF(ND.EQ.1) GO TO 50
```

```
  IC=0
```

```
  DO 40 K=1,NG
```

```
  IF(GD(K).LT.0) GO TO 20
```

```
  IX(K)=1
```

```
  IC=IC+1
```

```
  IL=IL+1
```

```
  GO TO 40
```

```
20 IX(K)=0
```

```
40 CONTINUE
```

```
  GO TO 80
```

```
50 CONTINUE
```

```
  IL=0
```

```
  DO 80 K=1,NG
```

```

      IF(IX(K).EQ.0.AND.GD(K).GE.0.) GO TO 60
      VC(K)=0.
      GO TO 80
60 CONTINUE
      VC(K)=1.
      IL=IL+1
80 CONTINUE
90 CONTINUE
      RETURN
      END

```

```

/*

```

```

//LKED.SYSLMOD DD DSN=CLOVIS(INDIC),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *

```

```

      SUBROUTINE DIREC
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
      1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
      2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
      3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
      4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
      5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
      6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
      7MX,J1,J2,J3,J4,J5,J6,J7
      DO 10 K=1,NC
      VX(K)=VY(K)+XM*DR(K)
-----
10 CONTINUE
      IF(ND.EQ.2) RETURN
      NF=NF+1
      IF(ND.EQ.5) GO TO 20
      CALL FUNCF
      IF(ND.EQ.1) CALL FUNCP
      RETURN
20 CALL TEXIS
      RETURN
      END

```

```

/*

```

```

//LKED.SYSLMOD DD DSN=CLOVIS(DIREC),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *

```

```

      SUBROUTINE NORMA
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
      1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
      2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
      3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
      4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
      5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
      6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
      7MX,J1,J2,J3,J4,J5,J6,J7

```

```

XN=0.
DO 50 K=1,NC
GO TO (10,20,30,40),ID
10 XN=XN+PX(K)*PX(K)
GO TO 50
20 XN=XN+PY(K)*PY(K)
GO TO 50
30 XN=XN+GX(K)*GX(K)
GO TO 50
40 XN=XN+DR(K)*DR(K)
50 CONTINUE
XN=DSQRT(XN)
RETURN
END

```

```
/*
```

```

//LKED.SYSLMOD DD DSN=CLOVIS(NORMA),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *

```

```

SUBROUTINE CONVE

```

```

IMPLICIT REAL*8(A-H,O-Z)

```

```

COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7

```

```

C**** TESTA REGRA DE PARADA P/ METODOS DESVINCULADOS S/ DERIVADAS

```

```

IA=0

```

```

IF(ND.NE.5) GO TO 10

```

```

IF(FX.GT.0.) RETURN

```

```

IA=-1

```

```

RETURN

```

```

10 CONTINUE

```

```

IF(NI.GE.NK) GO TO 60

```

```

DO 20 K=1,NC

```

```

VD(K)=VE(K)

```

```

VE(K)=VF(K)

```

```

VF(K)=VX(K)

```

```

PX(K)=VE(K)-VD(K)

```

```

PY(K)=VF(K)-VE(K)

```

```

20 CONTINUE

```

```

ID=1

```

```

CALL NORMA

```

```

XM=XN

```

```

ID=2

```

```

CALL NORMA

```

```

EP=DABS(XM-XN)

```

```

FI=FL

```



```

FL=FX
FH=FX
P1=FI-FL
P2=FL-FH
TE=DABS(P1-P2)
C**** TESTE PARA VALORES DE X E F(X)
IF(EP,LE,ZN,AND,TE,LE,ZF) GO TO 80
ID=3
CALL NORMA
C**** TESTE PARA O GRADIENTE DE F(X)
IF(XN,LE,ZE) GO TO 80
RETURN
60 ID=7
80 IA=-1
RETURN
END

/*
//LKED,SYSLMOD DD DSN=CLOVIS(CONVE),DISP=OLD
//
//CAREX JOB (3032,2410),MSGLEVEL=1,CLASS=C,TIME=8
//CLOVIS EXEC FORTGCL,PARM,LKED='NCAL,LET'
//FORT,SYSLMOD DD *
SUBROUTINE PLAK1
IMPLICIT REAL*8(A-H,O-Z)
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
C**** FUNCAO PARA O CALCULO DOS TETAS (EM BAIXO E EM CIMA)
TETA(A,B,C,D,E)=A-B-C*D*E
NX=NU
NU=NU+1
FZ=FX
C**** A SUBROTINA LAMBI CALCULA O LAMBDA INICIAL
CALL LAMBI
IF(IA,EQ,1) RETURN
FY=FZ
10 CONTINUE
XN=XM/2.
ID=0
AF=0.6
PE=0.
DO 30 K=1,NC
PE=PE+GX(K)*DR(K)
30 CONTINUE
IF(PE,LT,0.) GO TO 20

```

```

KL=KL+1
FX=FZ
RETURN
20 CONTINUE
KL=0
IF(MX.EQ.1) GO TO 65
C**** TESTES PARA PENALIDADES E DIRECOES VIAVEIS
TB=TETA(FX,FY,P2,AF,PE)
IF(TB.LT.0.) GO TO 80
AF=1.-AF
TC=TETA(FX,FY,P1,AF,PE)
IF(TC.GT.0.) GO TO 70
TC=TETA(FX,FY,P2,AF,PE)
IF(TC.LE.0.) RETURN
AF=1.-AF
TB=TETA(FX,FY,P1,AF,PE)
IF(TB.LT.0.) GO TO 55
GO TO 70
55 A0=P1
AF=1.-AF
GO TO 50
65 CONTINUE
CALL DIREC
TB=TETA(FX,FY,XM,AF,PE)
IF(TB.GT.ZF) GO TO 40
IF(TB.GE.0.) RETURN
XM=2.*XM
NU=NU+1
GO TO 20
40 AF=1.-AF
TC=TETA(FX,FY,XM,AF,PE)
IF(TC.LE.0.) RETURN
A0=XM/2.
IF(A0.EQ.XN) A0=0.
C**** DETERMINACAO DO VALOR OTIMO DE LAMBDA NO INTERVALO (B0-A0)
50 B0=XM
C**** XM ESTA AGORA ENTRE A0 E B0
60 AF=1.-AF
XM=(A0+B0)/2.
CALL DIREC
TB=TETA(FX,FY,XM,AF,PE)
AF=1.-AF
TC=TETA(FX,FY,XM,AF,PE)
IF(TB.GE.0. AND TC.LE.0.) RETURN
NU=NU+1
IF(TB.GT.0.) GO TO 50
A0=XM
GO TO 60
70 XM=P1
80 CALL DIREC

```

RETURN
END

```

/*
//LKED,SYSLMOD DD DSN=CLOVIS(PLAK1),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM,LKED='NCAL,LET'
//FORT,SYSDIN DD *
  SUBROUTINE PLAK2
    IMPLICIT REAL*8(A-H,O-Z)
    COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
    IHI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
    2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
    3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
    4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
    5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
    6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
    7MX,J1,J2,J3,J4,J5,J6,J7
C**** FUNCAO PARA O CALCULO DE TETA (EM CIMA)
  TETA(A,B,C,D,E)=A-B-C*D*E
  NX=NU
  NU=NU+1
  FZ=FX
C**** A SUBROTINA LAMBI CALCULA O LAMBDA INICIAL
  CALL LAMBI
  IF(IA.EQ.1) RETURN
  FY=FZ
10 CONTINUE
  IO=0
  AF=0.4
  BT=0.7
  PE=0.
  DO 30 K=1,NC
  PE=PE+GX(K)*DR(K)
30 CONTINUE
  IF(PE.LT.0.) GO TO 20
  KL=KL+1
  FX=FZ
  RETURN
20 CONTINUE
  KL=0
  CALL DIREC
  TC=TETA(FX,FY,XM,AF,PE)
  IF(TC.LE.0.) RETURN
  XM=BT*XM
  NU=NU+1
  GO TO 20
  END
/*
//LKED,SYSLMOD DD DSN=CLOVIS(PLAK2),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM,LKED='NCAL,LET'
//FORT,SYSDIN DD *

```

SUBROUTINE GOSEC

IMPLICIT REAL*8(A-H,O-Z)

```
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
```

C**** BUSCA UNIDIRECIONAL DE SECCAO AUREA

NX=NU

FZ=FX

BT=0.1

F1=0.38

F2=0.62

P1=0.

C**** A SUBROTINA LAMBI CALCULA O LAMBDA INICIAL

CALL LAMBI

IF(IA.EQ.1) GO TO 160

C**** EP E' A PRECISAO DA BUSCA

Z1=ZR

EP=ZR/XN

Z2=EP

C**** E1 E' A PRECISAO DA FUNCAO

E1=ZR*XN

Z3=E1

C**** E2 E' UM VALOR MENOR QUE E1

E2=BT*E1

Z4=E2

FY=FZ

IF(ND.EQ.0.OR.ND.EQ.5) GO TO 10

XN=0.

GO TO 40

10 CONTINUE

XN=2.*XM

20 CONTINUE

CALL DIREC

IF(FX.GE.FY) GO TO 40

XM=2.*XM

P1=XM/4.

FY=FX

NU=NU+1

GO TO 20

C**** XM ESTA AGORA ENTRE A0 E B0

40 CONTINUE

IF(XM.EQ.XN) P1=0.

A0=P1

B0=XM

A1=A0

```

      B1=B0
C**** DETERMINACAO DO VALOR OTIMO DE LAMBDA NO INTERVALO (B1-A1)
      50 CONTINUE
         NU=NU+1
         DL=B1-A1
C**** TESTE DE SUFICIENTE DECRESCIMO DO INTERVALO
      IF(DL.LE.EP) GO TO 100
      A0=A1
      A1=A1+F1*DL
      B0=B1
      B1=A0+F2*DL
      XM=A1
      CALL DIREC
      FY=FX
      XM=B1
      CALL DIREC
      IF(FY.GE.FX) GO TO 80
      A1=A0
      GO TO 50
      80 B1=B0
         GO TO 50
C**** VERIFICACAO DE SUFICIENTE DECRESCIMO DA FUNCAO
      100 CONTINUE
         XM=(A1+B1)/2.
         CALL DIREC
         TE=FZ-FX
         IF(TE.GT.E1) GO TO 160
C**** TESTE ENTRE A PRECISAO DA BUSCA E A DE RN
      IF(ZR.LT.ZN) GO TO 140
      IF(TE.LE.E2) GO TO 120
      ZR=0.5*ZR
      E1=0.5*E1
      E2=0.5*E2
      EP=0.5*EP
      GO TO 50
      120 CONTINUE
         ZR=BT*ZR
         EP=BT*EP
         E1=E2
         E2=BT*E2
         GO TO 50
      140 CONTINUE
         IF(TE.GT.0.) GO TO 160
         KL=KL+1
         FX=FZ
         ZR=Z1
         EP=Z2
         E1=Z3
         E2=Z4
         RETURN

```

160 CONTINUE

KL=0

RETURN

END

/*

//LKED,SYSLMOD DD DSN=CLOVIS(GOSEC),DISP=OLD

//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'

//FORT.SYSIN DD *

SUBROUTINE DSCPW

IMPLICIT REAL*8(A-H,O-Z)

COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
 1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
 2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
 3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
 4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
 5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
 6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
 7MX,J1,J2,J3,J4,J5,J6,J7

NX=NU

FZ=FX

BT=0.1

IG=0

X1=0.

T1=0.

X2=0.

T2=0.

C**** CALCULO DO RO INICIAL

CALL LAMBI

IF(IA.EQ.1) GO TO 300

C**** EP E' A PRECISAO DA BUSCA

Z1=ZR

EP=ZR/XN

Z2=EP

C**** E1 E' A PRECISAO DA FUNCAO

E1=ZR*XN

Z3=E1

C**** E2 E' UM VALOR MENOR QUE E1

E2=BT*E1

Z4=E2

IF(ND.EQ.0.OR.ND.EQ.5) GO TO 10

C**** DETERMINACAO DE 3 PONTOS EQUIDISTANTES PARA METODOS VINCULADOS

CALL DIREC

T3=FZ-FX

X3=XM

XM=P1

CALL DIREC

T1=FZ-FX

X1=XM

XM=0.5*(P1+P2)

CALL DIREC

```

T2=FZ-FX
X2=XM
RO=X2
GO TO 80
10 CONTINUE
RO=XM
XM=2.*RO
X3=XM
CALL DIREC
T3=FZ-FX
IF(T3.GT.T1) GO TO 20
C**** INTERVALO INICIAL DETERMINADO: LAMBDA=2*RO
X2=X1+RO
XM=X2
CALL DIREC
T2=FZ-FX
C**** DESVIO PARA INTERPOLACAO
GO TO 80
C**** FUNCAO CONTINUA DECRESCENDO
20 CONTINUE
NU=NU+1
X1=X2
T1=T2
X2=X3
T2=T3
X3=X3+2.*RO
XM=X3
CALL DIREC
T3=FZ-FX
IF(T3.LT.T2) GO TO 40
RO=2.*RO
GO TO 20
C**** INTERVALO INICIAL DETERMINADO, CONSTITUIDO DE 3 PARTES IGUAIS
40 CONTINUE
XM=X2+RO
CALL DIREC
TM=FZ-FX
C**** ELIMINACAO DO PONTO DE MENOR DECRESCIMO (TETA)
IF(TM.GT.T2) GO TO 60
X3=XM
T3=TM
GO TO 80
60 CONTINUE
X1=X2
T1=T2
X2=XM
T2=TM
C**** INTERPOLACOES
80 CONTINUE
NU=NU+1

```

```

IF(IG.EQ.1) GO TO 100
C**** INTERPOLACAO QUADRATICA DE DAVIES-SWANN-CAMPEY
P1=T1-2.*T2+T3
IF(P1.EQ.0.) GO TO 250
XM=X2+0.5*RO*(T1-T3)/P1
P1=(X1-XM)*(XM-X3)
IF(P1.LT.0.) GO TO 250
IG=1
GO TO 130
100 CONTINUE
P1=T1*(X2-X3)+T2*(X3-X1)+T3*(X1-X2)
IF(P1.EQ.0.) GO TO 160
C**** INTERPOLACAO QUADRATICA DE POWELL
P2=0.5*(T1*(X2*X2-X3*X3)+T2*(X3*X3-X1*X1)+T3*(X1*X1-X2*X2))
XM=P2/P1
120 CONTINUE
P1=(X1-XM)*(XM-X3)
IF(P1.LT.0.) GO TO 250
130 CONTINUE
XN=X3-X1
C**** TESTE DE SUFICIENTE DECRESCIMO DO INTERVALO
IF(XN.GT.EP) GO TO 180
C**** TESTE DE SUFICIENTE DECRESCIMO DA FUNCAO
IF(T2.GT.E1) GO TO 280
C**** TESTE DA PRECISAO DA BUSCA COM A DE RN
IF(ZR.LT.ZN) GO TO 160
C**** REDUCAO DAS PRECISOES
IF(T2.LE.E2) GO TO 140
EP=0.5*EP
E1=0.5*E1
E2=0.5*E2
ZR=0.5*ZR
GO TO 180
140 CONTINUE
EP=BT*EP
ZR=BT*ZR
E1=E2
E2=BT*E2
GO TO 180
C**** PRECISAO DA BUSCA ESTA MAIOR QUE A DE RN
160 CONTINUE
IF(T2.GT.0.) GO TO 280
GO TO 260
C**** SELECAO DOS NOVOS TRES PONTOS ENTRE: X1,X2,X3 E XM
180 CONTINUE
CALL DIREC
TM=FZ-FX
IF(XM.LT.X2) GO TO 220
C**** XM ESTA A DIREITA DE X2
IF(TM.GE.T2) GO TO 200

```



```

      X3=XM
      T3=TM
      GO TO 80
200  CONTINUE
      X1=X2
      T1=T2
      X2=XM
      T2=TM
      GO TO 80
C**** XM ESTA 'A' ESQUERDA DE X2
220  CONTINUE
      IF(TM.GE.T2) GO TO 240
      X1=XM
      T1=TM
      GO TO 80
240  CONTINUE
      X3=X2
      T3=T2
      X2=XM
      T2=TM
      GO TO 80
250  CONTINUE
      XM=0.5*(X1+X2)
      GO TO 130
C**** A BUSCA FALHOU
260  CONTINUE
      KL=KL+1
      FX=FZ
      ZR=Z1
      EP=Z2
      E1=Z3
      E2=Z4
      RETURN
C**** A BUSCA TEVE SUCESSO
280  CONTINUE
      XM=X2
      CALL DIREC
300  CONTINUE
      KL=0
      RETURN
      END

/*
//LKED.SYSLMOD DD DSN=CLOVIS(DSCPW),DISP=OLD
//
//CARIX JOB (3032,2410),MSGLEVEL=1,CLASS=C,TIME=5
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *
      SUBROUTINE DFLEP
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),

```

```

1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KB,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
  IT=1
  IF(MX.EQ.3) IT=2
  NV=0
  KL=0
  NI=-1
10 CONTINUE
  ID=1
  DO 40 K=1,NC
  DO 20 J=1,NC
  DR(J)=0.
20 CONTINUE
  DR(K)=1.
  WRITE(20,ID) (DR(I),I=1,NC)
40 CONTINUE
50 CONTINUE
  NI=NI+1
  IF(KL.EQ.1) GO TO 110
  CALL CONVE
  IF(ID.EQ.7) RETURN
  IF(IA.EQ.-1) GO TO 700
  IF(JN.EQ.0.OR.JN.EQ.2) GO TO 60
  ID=2
  IK=1
  CALL IMPRS
60 CONTINUE
  DO 100 K=1,NC
  VY(K)=VX(K)
  VZ(K)=GX(K)
100 CONTINUE
110 CONTINUE
  ID=1
  DO 120 K=1,NC
  READ(20,ID) (VA(I),I=1,NC)
  DR(K)=0.
  DO 120 J=1,NC
  DR(K)=DR(K)-VA(J)*VZ(J)
120 CONTINUE
150 CONTINUE
  CALL SELE2
C**** BUSCA NA DIRECAO CONTRARIA AA DO GRADIENTE
  IF(KL.EQ.1) GO TO 10
C**** BUSCA UNIDIRECIONAL FALHOU COMPLETAMENTE
  IF(KL.GT.1) GO TO 800

```

```

CALL GRADX
DO 160 K=1,NC
DG(K)=GX(K)-VZ(K)
DV(K)=VX(K)-VY(K)
160 CONTINUE
ID=1
DO 200 K=1,NC
READ(20,ID) (VA(I),I=1,NC)
DR(K)=0.
DO 200 J=1,NC
DR(K)=DR(K)+VA(J)*DG(J)
200 CONTINUE
GO TO (500,600),NA
500 CALL MDDFP
IF(NV,NE,1) GO TO 50
KL=KL+1
GO TO 10
600 CALL MDBRD
IF(NV,NE,1) GO TO 50
KL=KL+1
GO TO 10
700 ID=1
IK=1
RETURN
800 CONTINUE
ID=2
CALL IMPRS
ID=6
RETURN
END

```

/*.

```

//LKED,SYSLMOD DD DSN=CLOVIS(DFLEP),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT,SYSLMOD DD *
SUBROUTINE MDDFP
IMPLICIT REAL*8(A-H,O-Z)
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
ID=1
P1=0.
P2=0.
DO 220 K=1,NC
P1=P1+DR(K)*DG(K)
P2=P2+DV(K)*DG(K)

```

```

220 CONTINUE
   IF(P2.EQ.0.) GO TO 300
   DO 260 K=1,NC
   READ(20'ID) (VC(I),I=1,NC)
   DO 240 J=1,NC
   VY(J)=DR(K)*DR(J)
   VZ(J)=DV(K)*DV(J)
   VC(J)=VC(J)-(VY(J)/P1)+(VZ(J)/P2)
240 CONTINUE
   ID=ID-NC
   WRITE(20'ID) (VC(I),I=1,NC)
260 CONTINUE
   RETURN
300 CONTINUE
   NV=1
   RETURN
   END

```

```
/*
```

```

//LKED.SYSLMOD DD DSN=CLOVIS(MDDFP),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *

```

```

SUBROUTINE MDBRD
  IMPLICIT REAL*8(A-H,O-Z)
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
  1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
  2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
  3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
  4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
  5IX(20),IN,IO,NI,ID,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
  6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
  7MX,J1,J2,J3,J4,J5,J6,J7
  ID=1
  P1=0.
  DO 20 K=1,NC
  DV(K)=DV(K)-DR(K)
  P1=P1+DV(K)*DG(K)
20 CONTINUE
  IF(P1.EQ.0.) GO TO 80
30 CONTINUE
  DO 60 K=1,NC
  READ(20'ID) (VC(I),I=1,NC)
  DO 40 J=1,NC
  VC(J)=VC(J)+DV(K)*DV(J)/P1
40 CONTINUE
  ID=ID-NC
  WRITE(20'ID) (VC(I),I=1,NC)
60 CONTINUE
  RETURN
80 CONTINUE
  NV=1

```

```
RETURN
END
```

```
/*
```

```
//LKED,SYSLMOD DD DSN=CLOVIS(MDBRD),DISP=OLD
```

```
//CLOVIS EXEC FORTGCL,PARM, LKED='NCAL,LET'
```

```
//FORT,SYSLMOD DD *
```

```
  SUBROUTINE FRIVS
```

```
  IMPLICIT REAL*8(A-H,O-Z)
```

```
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
  1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
  2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
  3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
  4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
  5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
  6NS,N1,N2,N3,N4,IC,IA,IB,KB,LF,NK,NP,NU,NV,NX,IT,JN,
  7MX,J1,J2,J3,J4,J5,J6,J7
```

```
C**** METODO DE FLETCHER REEVES
```

```
  IT=1
```

```
  IF(MX.EQ.3) IT=2
```

```
  GO=1.
```

```
  I1=2*NC
```

```
  NI=-1
```

```
  IB=0
```

```
10 CONTINUE
```

```
  NI=NI+1
```

```
  I2=MOD(NI,I1)
```

```
C**** 'RESETEADO' A CADA 2N ITERACOES: DIRECAO=-GRADIENTE
```

```
  IF(I2.EQ.0) IB=0
```

```
  G1=0.
```

```
  KL=0
```

```
  DO 20 K=1,NC
```

```
    VY(K)=VX(K)
```

```
    G1=G1+GX(K)*GX(K)
```

```
20 CONTINUE
```

```
  XL=G1/GO
```

```
  IF(IB.NE.0) GO TO 30
```

```
  XL=0.
```

```
  IB=1
```

```
30 CONTINUE
```

```
  IF(JN.EQ.0.OR.JN.EQ.2) GO TO 70
```

```
  ID=2
```

```
  IK=1
```

```
  CALL IMPRS
```

```
70 CONTINUE
```

```
  CALL CONVE
```

```
  IF(ID.EQ.7) RETURN
```

```
  IF(IA.EQ.-1) GO TO 80
```

```
40 CONTINUE
```

```
  DO 50 K=1,NC
```

```
    DR(K)=-GX(K)+XL*DR(K)
```

```

50 CONTINUE
   CALL SELE2
C**** BUSCA UNIDIRECIONAL FALHOU COMPLETAMENTE
      IF(KL.GT.1) GO TO 100
      IF(KL.NE.1) GO TO 60
C**** BUSCA NA DIRECAO CCNTRARIA AA DO GRADIENTE
      XL=0.
      GO TO 40
60 CONTINUE
   CALL GRADX
      GO=G1
      GO TO 10
80 ID=1
      IK=1
      RETURN
100 CONTINUE
      ID=2
      CALL IMPRS
      ID=6
      RETURN
      END

/*
//LKED,SYS LMOD DD DSN=CLOVIS(FRIVS),DISP=GLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT,SYS IN DD *
SUBROUTINE COXIC
  IMPLICIT REAL*8(A-H,O-Z)
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
  1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
  2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
  3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
  4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
  5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
  6NS,N1,N2,N3,N4,IC,IA,IB,IL,NF,NK,NP,NU,NV,NX,IT,JN,
  7MX,J1,J2,J3,J4,J5,J6,J7
C**** METODO DE CAUCHY COM DERIVADAS
      IT=1
      IF(MX.EQ.3) IT=2
10 CONTINUE
      IF(JN.EQ.0.OR.JN.EQ.2) GO TO 30
      ID=2
      IK=1
      CALL IMPRS
30 CONTINUE
      CALL CONVE
      IF(ID.EQ.7) RETURN
      IF(IA.EQ.-1) GO TO 40
      DO 20 K=1,NC
      VY(K)=VX(K)
      DR(K)=-GX(K)

```

```

20 CONTINUE
   CALL SELE2
   IF(KL.GT.0) GO TO 60
   CALL GRADX
   NI=NI+1
   GO TO 10
40 CONTINUE
   ID=1
   IK=1
   RETURN
60 CONTINUE
   ID=2
   CALL IMPRS
   ID=6
   RETURN
   END

```

```
/*
```

```

//LKED.SYSLMOD DD DSN=CLOVIS(COXIC),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *

```

```

SUBROUTINE POWEL
IMPLICIT REAL*8(A-H,O-Z)
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7

```

```
C**** METODO DE POWELL
```

```

IT=1
IF(MX.EQ.3) IT=2
K1=NC*NC
K2=2*NC
K4=K1-NC+1
K5=NC-1
FO=FX
ID=1

```

```
C**** DETERMINACAO DAS DIRECOES INICIAIS SEGUNDO OS EIXOS
```

```

DO 30 K=1,NC
DO 10 J=1,NC
DR(J)=0.
10 CONTINUE
DR(K)=1.
WRITE(20,ID) (DR(I),I=1,NC)
VY(K)=VX(K)
30 CONTINUE
40 CONTINUE
DO 45 K=1,NC

```

```

45 VA(K)=VX(K)
   IF(JN.EQ.0.OR.JN.EQ.2) GO TO 50
C**** IMPRESSAO DE RESULTADOS INTERMEDIARIOS
   ID=2
   IK=0
   CALL IMPRS
50 CONTINUE
   IF(NI.GE.NK) GO TO 320
   NI=NI+1
   ID=1
C**** DETERMINACAO DOS VETORES X(I), I=1,N
   DO 80 K=1,NC
   READ(20,ID) (DR(I),I=1,NC)
   IB=ID
   FI=FX
   CALL SELE2
   IF(K.EQ.1) FH=FI-FX
   FL=FI-FX
   IF(FL.LT.FH) GO TO 60
   FH=FL
   K3=K
60 CONTINUE
   ID=IB
   DO 80 K=1,NC
   VY(K)=VX(K)
80 CONTINUE
   F1=FX
C**** DETERMINACAO DO VETOR X(N+1)
   DO 100 K=1,NC
   VB(K)=VX(K)
   VX(K)=2.*VX(K)-VA(K)
   VY(K)=VX(K)
   DR(K)=VB(K)-VA(K)
100 CONTINUE
   CALL FUNCF
   P1=(F0-2.*F1+FX)*(F0-F1-FH)*(F0-F1-FH)
   P2=0.5*FH*(F0-FX)*(F0-FX)
   IF(FX.LT.F0.OR.P1.LT.P2) GO TO 160
C**** AS DIRECOES PERMANCEM AS MESMAS
   IF(F1.GE.FX) GO TO 280
C**** O NOVO PONTO X(0)=X(N)
   DO 120 K=1,NC
   VY(K)=VB(K)
   VX(K)=VB(K)
120 CONTINUE
   FX=F1
   F0=FX
   GO TO 280
C**** DETERMINACAO DO MINIMO NA DIRECAO X(N)-X(0)
160 CONTINUE

```



```

      FX=F0
      KL=0
      DO 180 K=1,NC
180   VY(K)=VA(K)
      CALL SELE2
      IF(KL.GT.0) GO TO 300
      FO=FX
C**** ATUALIZACAO DAS DIRECOES DE BUSCA
      IF(K3.LT.NC) GO TO 200
      ID=K4
      GO TO 260
200   CONTINUE
      ID=K3
      DO 240 K=K3,K5
      ID=ID+NC
      READ(20,ID) (VC(I),I=1,NC)
      ID=ID-K2
      WRITE(20,ID) (VC(I),I=1,NC)
240   CONTINUE
260   CONTINUE
      WRITE(20,ID) (DR(I),I=1,NC)
280   CONTINUE
      ID=4
      CALL NORMA
C**** TESTE DE PARADA
      IF(XN.GT.ZN) GO TO 40
-----
      ID=1
      RETURN
300   CONTINUE
      ID=2
      CALL IMPRS
      ID=6
      RETURN
320   CONTINUE
      ID=7
      CALL IMPRS
      ID=2
      RETURN
      END

/*
//LKED.SYSLMOD DD DSN=CLOVIS(POWEL),DISP=CLD
//
//CAREX JOB (3032,2410),MSGLEVEL=1,CLASS=C,TIME=8
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *
      SUBROUTINE FUNCF
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,

```

```

3FL, FH, TX, EP, E1, E2, P1, P2, Q1, Q2, V1, V2, V3, V4, V5, V6, V7,
4VA(100), VB(100), VC(100), VD(100), PX(100), PY(100),
5IX(20), IN, IO, NI, ID, IK, NC, NB, NA, NG, NR, IL, NT, ND, NH,
6NS, N1, N2, N3, N4, IC, IA, IB, KL, NF, NK, NP, NU, NV, NX, IT, JN,
7MX, J1, J2, J3, J4, J5, J6, J7

```

```
WRITE(IO,1)
```

```
1 FORMAT(//20X, 'SR. USUARIO, '//20X, 'A SUBROTINA FUNC F NAO',
1' FOI INCLUIDA ENTRE OS DADOS DE ENTRADA. '/20X,
2' FACA A CORRECAO E RECOMECE O PROBLEMA.')
```

```
NV=1
```

```
RETURN
```

```
END
```

```
/*
```

```
//LKED. SYSLMOD DD DSN=CLOVIS(FUNCF), DISP=OLD
```

```
//CLOVIS EXEC FORTGCL, PARM. LKED='NCAL, LET'
```

```
//FORT. SYSIN DD *
```

```
SUBROUTINE GRADF
```

```
IMPLICIT REAL*8(A-H, O-Z)
```

```
COMMON GG(20,100), GH(20,100), VX(100), GX(100), GD(20),
1HI(20), FX, VF(100), VY(100), DR(100), VE(100), VZ(100),
2HM(100), DG(100), DV(100), FY, ZR, ZE, XN, XM, ZN, ZF, TE, FI,
3FL, FH, TX, EP, E1, E2, P1, P2, Q1, Q2, V1, V2, V3, V4, V5, V6, V7,
4VA(100), VB(100), VC(100), VD(100), PX(100), PY(100),
5IX(20), IN, IO, NI, ID, IK, NC, NB, NA, NG, NR, IL, NT, ND, NH,
6NS, N1, N2, N3, N4, IC, IA, IB, KL, NF, NK, NP, NU, NV, NX, IT, JN,
7MX, J1, J2, J3, J4, J5, J6, J7

```

```
WRITE(IO,1)
```

```
1 FORMAT(//20X, 'SR. USUARIO, '//20X, 'A SUBROTINA GRADF NAO',
1' FOI INCLUIDA ENTRE OS DADOS DE ENTRADA. '/20X,
2' FACA A CORRECAO E RECOMECE O PROBLEMA.')
```

```
NV=1
```

```
RETURN
```

```
END
```

```
/*
```

```
//LKED. SYSLMOD DD DSN=CLOVIS(GRADF), DISP=OLD
```

```
//CLOVIS EXEC FORTGCL, PARM. LKED='NCAL, LET'
```

```
//FORT. SYSIN DD *
```

```
SUBROUTINE FUNHI
```

```
IMPLICIT REAL*8(A-H, O-Z)
```

```
COMMON GG(20,100), GH(20,100), VX(100), GX(100), GD(20),
1HI(20), FX, VF(100), VY(100), DR(100), VE(100), VZ(100),
2HM(100), DG(100), DV(100), FY, ZR, ZE, XN, XM, ZN, ZF, TE, FI,
3FL, FH, TX, EP, E1, E2, P1, P2, Q1, Q2, V1, V2, V3, V4, V5, V6, V7,
4VA(100), VB(100), VC(100), VD(100), PX(100), PY(100),
5IX(20), IN, IO, NI, ID, IK, NC, NB, NA, NG, NR, IL, NT, ND, NH,
6NS, N1, N2, N3, N4, IC, IA, IB, KL, NF, NK, NP, NU, NV, NX, IT, JN,
7MX, J1, J2, J3, J4, J5, J6, J7

```

```
WRITE(IO,1)
```

```
1 FORMAT(//20X, 'SR. USUARIO, '//20X, 'A SUBROTINA FUNHI NAO',
1' FOI INCLUIDA ENTRE OS DADOS DE ENTRADA. '/20X,
```

```
2'FACA A CORRECAO E RECOMECE O PROBLEMA.' )
```

```
NV=1
RETURN
END
```

```
/*
```

```
//LKED,SYSLMOD DD DSN=CLOVIS(FUNHI),DISP=OLD
```

```
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
```

```
//FORT,SYSIN DD *
```

```
  SUBROUTINE FUNGD
```

```
  IMPLICIT REAL*8(A-H,O-Z)
```

```
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
  1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
  2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
  3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
  4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
  5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
  6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
  7MX,J1,J2,J3,J4,J5,J6,J7
```

```
  WRITE(IO,1)
```

```
1 FORMAT(//20X,'SR. USUARIO,'//20X,'A SUBROTINA FUNGD NAO',
1' FOI INCLUIDA ENTRE OS DADOS DE ENTRADA.'/20X,
2'FACA A CORRECAO E RECOMECE O PROBLEMA.' )
```

```
  NV=1
  RETURN
  END
```

```
/*
```

```
//LKED,SYSLMOD DD DSN=CLOVIS(FUNGD),DISP=OLD
```

```
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
```

```
//FORT,SYSIN DD *
```

```
  SUBROUTINE GRADV
```

```
  IMPLICIT REAL*8(A-H,O-Z)
```

```
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
  1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
  2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
  3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
  4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
  5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
  6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
  7MX,J1,J2,J3,J4,J5,J6,J7
```

```
  WRITE(IO,1)
```

```
1 FORMAT(//20X,'SR. USUARIO,'//20X,'A SUBROTINA GRADV NAO',
1' FOI INCLUIDA ENTRE OS DADOS DE ENTRADA.'/20X,
2'FACA A CORRECAO E RECOMECE O PROBLEMA.' )
```

```
  NV=1
  RETURN
  END
```

```
/*
```

```
//LKED,SYSLMOD DD DSN=CLOVIS(GRADV),DISP=OLD
```

```
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
```

```
//FORT,SYSIN DD *
```

SUBROUTINE COXIS

IMPLICIT REAL*8(A-H,O-Z)

```
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,ID,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,IF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
```

C**** METODO DE CAUCHY SEM DERIVADAS

FZ=FX

AF=0.4

BT=8.

KL=0

ID=2

IT=1

IF(MX.EQ.3) IT=2

IK=0

CALL IMPRS

10 CONTINUE

IF(NI.GE.NK) GO TO 170

NI=NI+1

20 CONTINUE

DO 30 J=1,NC

VY(J)=VX(J)

30 CONTINUE

50 CONTINUE

DO 80 K=1,NC

DO 40 J=1,NC

HM(J)=0.

40 CONTINUE

HM(K)=1.

DO 60 J=1,NC

VX(J)=VY(J)+ZR*HM(J)

60 CONTINUE

CALL FUNCF

NF=NF+1

DR(K)=-{FX-FZ}/ZR

GX(K)=DR(K)

80 CONTINUE

ID=4

CALL NORMA

90 CONTINUE

DO 100 K=1,NC

VX(K)=VY(K)+BT*ZR*DR(K)/XN

100 CONTINUE

CALL FUNCF

NF=NF+1

DX=FX-FZ

```

110 CONTINUE
    IF(DX.GE.0.) GO TO 140
    FX=FZ
    CALL SELE2
    IF(KL.GT.0) GO TO 160
    TE=FX-FZ
    ID=4
    CALL NORMA
    P1=-XN*ZR
120 CONTINUE
    IF(TE.GT.P1) GO TO 140
    FZ=FX
    IF(JN.EQ.0.OR.JN.EQ.2) GO TO 10
    ID=2
    IK=0
    CALL IMPRS
    GO TO 10
140 CONTINUE
    ZR=0.5*ZR
    IF(ZR.GT.ZN) GO TO 50
    IF(FZ.LT.FX) FX=FZ
    ID=1
    IK=0
    RETURN
160 CONTINUE
    ID=2
    CALL IMPRS
    ID=6
    RETURN
170 CONTINUE
    ID=7
    CALL IMPRS
    ID=2
    RETURN
END

```

```
/*
```

```
//LKED,SYSMOD DD DSN=CLOVIS(COXIS),DISP=OLD
```

```
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
```

```
//FORT,SYSIN DD *
```

```
  SUBROUTINE NEDMD
```

```
  IMPLICIT REAL*8(A-H,O-Z)
```

```
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
```

```
C**** METODO DE NELDER-MEAD
```

```

C**** SE ND=0, 'NEDMD' FOI CHAMADA PELO 'PROGRAMA PRINCIPAL'
C**** SE ND=2, 'NEDMD' FOI CHAMADA PELA SUBROTINA 'VIAVL'
  ID=2
  IT=1
  IF(MX.EQ.4) IT=2
  IK=0
  CALL IMPRS
  AF=1.
  BT=0.5
  GA=2.
  NI=NC+1
  IF(ND.EQ.0) TE=1
  CALL VERTS
30 CONTINUE
  IF(NI.GE.NK) GO TO 400
  NI=NI+1
  DO 20 K=1,NC
  HM(K)=0.
20 CONTINUE
  IK=0
  ID=1
C**** DETERMINACAO DE FMAX E FMIN E PONTOS CORRESPONDENTES
  DO 110 K=1,N1
  READ(20,ID) (VX(I),I=1,NC)
  DO 40 J=1,NC
  HM(J)=HM(J)+VX(J)
40 CONTINUE
  CALL TEXIS
50 CONTINUE
  IK=IK+1
  DG(IK)=FX
  IF(K.GT.1) GO TO 70
  FH=FX
  FL=FX
  DO 60 J=1,NC
  DR(J)=VX(J)
  DV(J)=VX(J)
60 CONTINUE
  GO TO 80
70 CONTINUE
  IF(FX.LE.FH) GO TO 90
  IF(FL.EQ.FH) FY=FH
  FH=FX
  DO 80 J=1,NC
  IF(FL.EQ.FH) VD(J)=DR(J)
  DR(J)=VX(J)
80 CONTINUE
  L=NC*(K-1)
  GO TO 110
90 CONTINUE

```

```

      IF(FX. GE. FL) GO TO 105
      FY=FL
      FL=FX
      DO 100 J=1, NC
      VD(J)=DV(J)
      DV(J)=VX(J)
100  CONTINUE
      GO TO 110
105  CONTINUE
      IF(FY. GT. FL. AND. FX. GE. FY) GO TO 110
      FY=FX
      DO 110 J=1, NC
      VD(J)=VX(J)
110  CONTINUE
      IF(ND. EQ. 2. AND. FL. LE. FI) GO TO 370
C**** REFLEXAO
      DO 120 K=1, NC
      HM(K)=HM(K)-DR(K)
      HM(K)=HM(K)/NC
      VY(K)=HM(K)+AF*(HM(K)-DR(K))
      VX(K)=VY(K)
120  CONTINUE
      CALL TEXIS
      IF(FX. LE. FL) GO TO 240
      DO 140 K=1, N1
      IF(DG(K). EQ. FH) GO TO 140
      IF(FX. LE. DG(K)) GO TO 280
140  CONTINUE
      IF(FX. GE. FH) GO TO 160
      DO 160 K=1, NC
      DR(K)=VY(K)
160  CONTINUE
C**** CONTRACAO
      DO 170 K=1, NC
      VX(K)=HM(K)+BT*(DR(K)-HM(K))
170  CONTINUE
      CALL TEXIS
175  CONTINUE
      IF(FX. LE. FH) GO TO 200
C**** REDUCAO
      ID=1
      DO 190 K=1, N1
      READ(20, ID) (PX(I), I=1, NC)
      DO 180 J=1, NC
      PX(J)=DV(J)+0.5*(PX(J)-DV(J))
180  CONTINUE
      ID=ID-NC
      WRITE(20, ID) (PX(I), I=1, NC)
190  CONTINUE
      GO TO 310

```

```

200 DO 220 K=1,NC
    DR(K)=VX(K)
220 CONTINUE
    GO TO 300
C**** EXPANSAO
240 DO 260 K=1,NC
    VZ(K)=HM(K)+GA*(VY(K)-HM(K))
    VX(K)=VZ(K)
260 CONTINUE
    CALL TESIS
    IF(FX.GE.FL) GO TO 280
    DO 270 K=1,NC
    DR(K)=VZ(K)
270 CONTINUE
    GO TO 300
280 CONTINUE
    DO 300 K=1,NC
    DR(K)=VY(K)
300 CONTINUE
C**** ATUALIZACAO DO CONJUNTO DE VERTICES
    ID=L+1
    WRITE(20,ID) (DR(I),I=1,NC)
310 CONTINUE
C**** TESTE DE CONVERGENCIA E PARADA
    DO 320 K=1,NC
    VX(K)=HM(K)
320 CONTINUE
    CALL TESIS
330 CONTINUE
    XN=0.
    ID=1
    FH=FX
    DO 360 K=1,N1
    READ(20,ID) (VX(I),I=1,NC)
    CALL TESIS
    XN=XN+(FX-FH)**2
360 CONTINUE
    XN=XN/N1
    XN=DSQRT(XN)
    IF(XN.LE.ZF) GO TO 380
    IF(JN.EQ.0.OR.JN.EQ.2) GO TO 30
C**** IMPRESSAO DE RESULTADOS INTERMEDIARIOS
    ID=2
    IK=0
    CALL IMPRS
    GO TO 30
380 CONTINUE
    ID=1
    IK=0
    FX=FL

```



```

370 CONTINUE
    TX=FL
    DO 390 K=1,NC
390 VX(K)=DV(K)
    RETURN
400 CONTINUE
    ID=7
    CALL IMPRS
    ID=2
    RETURN
    END

/*
//LKED,SYSLMOD DD DSN=CLOVIS(NEDMD),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT,SYSLMOD DD *
//
//CAREX JOB (3032,2410),MSGLEVEL=1,CLASS=C,TIME=5
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT,SYSLMOD DD *
    SUBROUTINE PENAL
    IMPLICIT REAL*8(A-H,O-Z)
    COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
    1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
    2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
    3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
    4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
    5IX(20),IN,IO,NI,ID,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
    6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
    7MX,J1,J2,J3,J4,J5,J6,J7
C**** METODO DE PENALIDADES
    NJ=0
    ZD=ZE
    E0=2.
    AF=0.4
    A1=0.3
    A2=0.2
    BT=0.65
    ID=4
    IT=1
    IK=3
    CALL IMPRS
10 CONTINUE
    IF(NX.GE.NK) GO TO 100
    NX=NX+1
    CALL INDIC
    CALL PENAF
    IF(NX.GT.1) GO TO 40
    CALL GRADF
    CALL GRADP
    DO 20 K=1,NC

```

```
PX(K)=VA(K)
PY(K)=VB(K)
20 CONTINUE
  ID=1
  CALL NORMA
  Q1=XN
  ID=3
  CALL NORMA
  Q1=Q1/XN
  Q2=XN
  ID=2
  CALL NORMA
  Q2=Q2/XN
40 CONTINUE
  ND=1
  NA=N3
  Z1=ZR
  ZE=E0
  K1=NX
  NU=0
  NI=0
  NX=0
  CALL FUNCF
  NF=NF+1
  CALL FUNCP
  CALL GRADX
  DO 60 K=1,NC
60 DR(K)=GX(K)
  CALL SELE3
  NJ=NJ+NI
  NX=K1
  ZR=Z1
  E0=AF*E0
  IF(E0.LE.ZD) GO TO 80
  Q1=A1*Q1
  Q2=A2*Q2
  ND=2
  CALL FUNCP
  CALL GRADX
  IF(JN.EQ.0) GO TO 10
  NI=NJ
  ID=4
  IT=1
  IK=3
  CALL IMPRS
  GO TO 10
80 CONTINUE
  ND=2
  CALL FUNCP
  CALL GRADX
```

```

NI=NJ
ID=3
IT=1
IK=3
RETURN
100 CONTINUE
ID=7
CALL IMPRS
ID=4
RETURN
END

```

```

/*
//LKED,SYSMOD DD DSN=CLOVIS(PENAL),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT,SYSIN DD *
SUBROUTINE GRADX
IMPLICIT REAL*8(A-H,O-Z)
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7

```

C**** CALCULA O GRADIENTE DA FUNCAO OBJETIVO OU PENALIZADA

```

IF(ND.EQ.5) GO TO 50
CALL GRADF
IF(ND.EQ.0) RETURN
CALL GRADP
DO 40 K=1,NC
GX(K)=GX(K)+VA(K)/Q1+Q2*VB(K)
40 CONTINUE
RETURN
50 CONTINUE
DO 60 K=1,NC
60 GX(K)=0.
DO 80 K=1,NG
IF(GD(K).LE.0.) GO TO 80
DO 70 J=1,NC
70 GX(J)=GX(J)+2.*GD(K)*GG(K,J)
80 CONTINUE
RETURN
END

```

```

/*
//LKED,SYSMOD DD DSN=CLOVIS(GRADX),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT,SYSIN DD *
SUBROUTINE GRADP
IMPLICIT REAL*8(A-H,O-Z)

```

```

COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KB,LF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7

```

```

C**** CALCULA OS GRADIENTES DAS FUNCOES PENALIDADES USANDO OS VINCULOS

```

```

DO 40 K=1,NC
VA(K)=0.
VB(K)=0.
40 CONTINUE
IF(ND.NE.1) GO TO 60
CALL FUNGD
CALL FUNHI
60 CONTINUE
CALL GRADV
IF(NH.EQ.0) GO TO 90
DO 80 K=1,NH
DO 80 J=1,NC
VA(J)=VA(J)+2.*HI(K)*GH(K,J)
80 CONTINUE
90 CONTINUE
IF(NG.EQ.0) RETURN
DO 140 K=1,NG
IF(IX(K).EQ.0) GO TO 120
IF(GD(K).LT.0.) GO TO 140
DO 100 J=1,NC
VA(J)=VA(J)+2.*GD(K)*GG(K,J)
100 CONTINUE
GO TO 140
120 CONTINUE
DO 140 J=1,NC
VB(J)=VB(J)+(1./GD(K)**2)*GG(K,J)
140 CONTINUE
160 CONTINUE
RETURN
END

```

```

/*

```

```

//LKED,SYSLMOD DD DSN=CLOVIS(GRADP),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT,SYSIN DD *

```

```

SUBROUTINE PENAF
IMPLICIT REAL*8(A-H,O-Z)
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),

```

```

5IX(20), IN, IO, NI, ID, IK, NC, NB, NA, NG, NR, IL, NT, ND, NH,
6NS, N1, N2, N3, N4, IC, IA, IB, KL, NF, NK, NP, NU, NV, NX, IT, JN,
7MX, J1, J2, J3, J4, J5, J6, J7

```

```

C**** CALCULA AS PENALIDADES P1 E P2, INTERIORES E EXTERIORES

```

```

P1=0.
P2=0.
10 CONTINUE
IF(NH.EQ.0) GO TO 40
CALL FUNHI
DO 40 K=1, NH
P1=P1+HI(K)*HI(K)
40 CONTINUE
IF(NG.EQ.0) RETURN
CALL FUNGD
DO 80 K=1, NG
IF(IX(K).EQ.0) GO TO 60
IF(GD(K).LT.0.) GD(K)=0.
P1=P1+GD(K)*GD(K)
GO TO 80
60 P2=P2-1./GD(K)
80 CONTINUE
90 CONTINUE
RETURN
END

```

```

/*

```

```

//LKED.SYSLMOD DD DSN=CLOVIS(PENAF), DISP=OLD

```

```

//CLOVIS EXEC FORTGCL, PARM=LKED='NCAL,LET'

```

```

//FORT.SYSIN DD *

```

```

SUBROUTINE FUNCP
IMPLICIT REAL*8(A-H,O-Z)
COMMON GG(20,100), GH(20,100), VX(100), GX(100), GD(20),
1HI(20), FX, VF(100), VY(100), DR(100), VE(100), VZ(100),
2HM(100), DG(100), DV(100), FY, ZR, ZE, XN, XM, ZN, ZF, TE, FI,
3FL, FH, TX, EP, E1, E2, P1, P2, Q1, Q2, V1, V2, V3, V4, V5, V6, V7,
4VA(100), VB(100), VC(100), VD(100), PX(100), PY(100),
5IX(20), IN, IO, NI, ID, IK, NC, NB, NA, NG, NR, IL, NT, ND, NH,
6NS, N1, N2, N3, N4, IC, IA, IB, KL, NF, NK, NP, NU, NV, NX, IT, JN,
7MX, J1, J2, J3, J4, J5, J6, J7

```

```

C**** CALCULA A FUNCAO PENALIZADA

```

```

CALL PENAF
FX=FX+P1/Q1+Q2*P2
NF=NF+1
20 CONTINUE
RETURN
END

```

```

/*

```

```

//LKED.SYSLMOD DD DSN=CLOVIS(FUNCP), DISP=OLD

```

```

//CLOVIS EXEC FORTGCL, PARM=LKED='NCAL,LET'

```

```

//FORT.SYSIN DD *

```

```

SUBROUTINE FLEXT

```

```

IMPLICIT REAL *8(A-H,O-Z)
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7

```

```
ID=4
```

```
IT=1
```

```
IK=2
```

```
CALL IMPRS
```

```
NS=0
```

```
AF=1.0
```

```
BT=0.5
```

```
GA=2.0
```

```
K1=NC*(NC+1)+1
```

```
NR=NC-NH
```

```
IF(NR.LT.2) NR=2
```

```
N1=NR+1
```

```
NT=NC*N1
```

```
TE=1.0
```

```
C**** CALCULO DE FI(0)
```

```
FI=2.0*(NH+1)*TE
```

```
ND=3
```

```
20 CONTINUE
```

```
CALL TESIS
```

```
IF(TX.LE.FI) GO TO 30
```

```
TE=0.05*FI
```

```
NQ=NK
```

```
NK=10000
```

```
NX=NX+1
```

```
NS=NK
```

```
NI=0
```

```
CALL VIAVL
```

```
IF(IC.EQ.1) RETURN
```

```
NX=NS
```

```
NJ=NJ+NI
```

```
NK=NQ
```

```
30 CONTINUE
```

```
CALL VERTS
```

```
40 CONTINUE
```

```
IF(JN.EQ.0) GO TO 50
```

```
NI=NJ
```

```
ID=4
```

```
IT=1
```

```
IK=2
```

```
CALL IMPRS
```

```
50 CONTINUE
```

```

IF(NX.GE.NK) GO TO 140
NX=NX+1
ID=K1
DO 70 K=1,N1
READ(20,ID) (VX(I),I=1,NC)
CALL TEXIS
IF(TX.LE.FI) GO TO 70
NQ=NK
NK=10000
NS=NX
NI=0
CALL CRITV
IF(IC.EQ.1) RETURN
NX=NS
NJ=NJ+NI
NK=NQ
ID=ID-NC
WRITE(20,ID) (VX(I),I=1,NC)
70 CONTINUE
ND=3
TE=0.05*FI
C**** CALCULO DO CENTROIDE E REFLEXAO
CALL CENTR
CALL CLCFI
CALL OPERC
IF(NI.GE.1000) FI=0.5*FI
IF(FI.LE.ZN) GO TO 100
DO 80 K=1,NC
80 VX(K)=DV(K)
FX=FL
GO TO 40
100 CONTINUE
DO 120 K=1,NC
120 VX(K)=DV(K)
FX=FL
ID=3
IT=1
IK=2
RETURN
140 CONTINUE
ID=7
CALL IMPRS
ID=4
RETURN
END

/*
//LKED.SYSLMOD DD DSN=CLOVIS(FLEXT),DISP=OLD
//
//CARIX JOB (3032,2410),MSGLEVEL=1,CLASS=C,TIME=3
//CLOVIS EXEC FORTGCL,PARM.LKED='NCAL,LET'

```

```

//FORT,SYSIN DD *
  SUBROUTINE VIAVL
  IMPLICIT REAL*8(A-H,O-Z)
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
  1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
  2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
  3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
  4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
  5IX(20),IN,IO,NI,ID,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
  6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
  7MX,J1,J2,J3,J4,J5,J6,J7
  NI=0
  N1=NC+1
  NS=0
  ND=2
  EP=1.0D-7
  10 CONTINUE
  20 CONTINUE
C**** MINIMIZACAO DE TX
  CALL NEDMD
  IF(TX.LE.FI) GO TO 180
C**** CALCULO DE A(S)
  DO 40 K=1,NC
  VY(K)=VX(K)
  VX(K)=HM(K)
  40 CONTINUE
  ID=1
  CALL TESIS
  FH=TX
  DO 80 K=1,N1
  READ(20,ID) (VX(I),I=1,NC)
  FL=0.
  CALL TESIS
  TX=TX-FH
  TX=TX*TX
  FL=FL+TX
  80 CONTINUE
  FL=DSQRT(FL)
  FL=FL/N1
  IF(FL.GT.EP) GO TO 20
C**** MINIMIZACAO UNIDIRECIONAL SEGUNDO OS EIXOS
  CALL TESIS
  DO 120 K=1,NC
  DO 100 J=1,NC
  100 DR(J)=0.
  DR(K)=1.
  FX=TX
  CALL DSCPW
  CALL TESIS
  IF(TX.LE.FI) RETURN

```



```

120 CONTINUE
   NS=NS+1
   IF(NS.LT.3) GO TO 10
   ID=6
   CALL IMPRS
   IC=1
   RETURN
180 IF(TX.GT.ZR) RETURN
   IF(NH.NE.0) RETURN
   CALL INTER
   RETURN
   END

/*
//LKED.SYSLMOD DD DSN=CLOVIS(VIAVL),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *
   SUBROUTINE CENTR
   IMPLICIT REAL*8(A-H,O-Z)
   COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,TX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KB,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
   AF=1.
   N1=NR+1
   DO 20 K=1,NC
   HM(K)=0.
20 CONTINUE
   IK=0
   ID=NC*(NC+1)+1
C**** DETERMINACAO DE FAAX E FMIN E PONTOS CORRESPONDENTES
   DO 100 K=1,N1
   READ(20,ID) (VX(I),I=1,NC)
   DO 40 J=1,NC
   HM(J)=HM(J)+VX(J)
40 CONTINUE
   CALL FUNCF
50 CONTINUE
   IK=IK+1
   DG(IK)=FX
   IF(K.GT.1) GO TO 70
   FH=FX
   FL=FX
   DO 60 J=1,NC
   DR(J)=VX(J)
   DV(J)=VX(J)
60 CONTINUE

```

```

      GO TO 80
70 CONTINUE
   IF(FX.LE.FH) GO TO 90
   FH=FX
   DO 80 J=1,NC
   DR(J)=VX(J)
80 CONTINUE
   IL=NC*(K-1)
   GO TO 100
90 CONTINUE
   IF(FX.GE.FL) GO TO 100
   FL=FX
   DO 100 J=1,NC
   DV(J)=VX(J)
100 CONTINUE
C**** REFLEXAO
   DO 120 K=1,NC
   HM(K)=HM(K)-DR(K)
   HM(K)=HM(K)/NR
   VY(K)=HM(K)+AF*(HM(K)-DR(K))
   VX(K)=VY(K)
120 CONTINUE
   CALL TESIS
   CALL CRITV
   RETURN
   END

```

```

/*
//LKED.SYSLMOD DD DSN=CLOVIS(CENTR),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *
   SUBROUTINE OPERC
   IMPLICIT REAL*8(A-H,O-Z)
   COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
   ND=3
   BT=0.5
   GA=2.
   K1=NC*(NC+1)+1
   IF(FX.LE.FL) GO TO 240
   DO 140 K=1,N1
   IF(DG(K).EQ.FH) GO TO 140
   IF(FX.LE.DG(K)) GO TO 280
140 CONTINUE
   IF(FX.GE.FH) GO TO 160

```

```

DO 160 K=1,NC
DR(K)=VY(K)
160 CONTINUE
C**** CONTRACAO
DO 170 K=1,NC
VX(K)=HM(K)+BT*(DR(K)-HM(K))
170 CONTINUE
CALL TESIS
CALL CRITV
IF(FX0LE0FH) GO TO 200
C**** REDUCAO
ID=K1
DO 190 K=1,N1
READ(20'ID) (PX(I),I=1,NC)
DO 180 J=1,NC
PX(J)=DV(J)+0.5*(PX(J)-DV(J))
180 CONTINUE
ID=ID-NC
WRITE(20'ID) (PX(I),I=1,NC)
190 CONTINUE
RETURN
200 DO 220 K=1,NC
DR(K)=VX(K)
220 CONTINUE
GO TO 300
C**** EXPANSAO
240 DO 260 K=1,NC
VZ(K)=HM(K)+GA*(VY(K)-HM(K))
VX(K)=VZ(K)
260 CONTINUE
CALL TESIS
CALL CRITV
IF(FX0GE0FL) GO TO 280
DO 270 K=1,NC
DR(K)=VZ(K)
270 CONTINUE
GO TO 300
280 CONTINUE
DO 300 K=1,NC
DR(K)=VY(K)
300 CONTINUE
C**** ATUALIZACAO DO VETOR PX
ID=K1+IL
WRITE(20'ID) (DR(I),I=1,NC)
RETURN
END

/*
//LKED0SYSLMOD DD DSN=CLOVIS(OPERC),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM0LKED='NCAL,LET'
//FORT0SYSIN DD *

```

```

SUBROUTINE INTER
  IMPLICIT REAL*8(A-H,O-Z)
  COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
  1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
  2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
  3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
  4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
  5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
  6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
  7MX,J1,J2,J3,J4,J5,J6,J7
  IJ=0
  IK=2
  DO 20 K=1,NC
    GX(K)=VD(K)-VX(K)
20 CONTINUE
  ID=3
  CALL NORMA
  DO 40 K=1,NC
    VY(K)=VX(K)
    VX(K)=VD(K)
40 CONTINUE
  CALL FUNGD
  GV=0.
  DO 60 K=1,NG
    IF(GD(K).LE.ZR) GO TO 60
    GV=GV+GD(K)*GD(K)
  IJ=IJ+1
  IX(IJ)=K
60 CONTINUE
  CF=0.5
  DG(3)=GV
80 CONTINUE
  GV=0.
  DO 100 K=1,NC
100 VX(K)=VY(K)+CF*XN*GX(K)
  CALL FUNGD
  DO 120 K=1,NG
  DO 120 J=1,IJ
    IF(K.EQ.IX(J)) GV=GV+GD(K)*GD(K)
120 CONTINUE
  DG(IK)=GV
  IK=IK-1
  CF=0.
  IF(IK.GE.1) GO TO 80
  AF=DG(1)-2.*DG(2)+DG(3)
  BT=3.*DG(1)-4.*DG(2)+DG(3)
  XM=BT*BT-8.*AF*DG(1)
  IF(XM.LT.0.) XM=0.
  XM=DSQRT(XM)
  CF=(BT+XM)/(4.*AF*XN)

```

```

      DO 140 K=1,NC
140  VX(K)=VX(K)+CF*GX(K)
150  CONTINUE
      RETURN
      END

```

```
/*
```

```

//LKED.SYSLMOD DD DSN=CLOVIS(INTER),DISP=GLD
//CLOVIS EXEC FORTGCL,PARM,LKED='NCAL,LET'
//FORT.SYSIN DD *

```

```

      SUBROUTINE CLCFI
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KB,LF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
      XN=N1
      ID=NC*(NC+1)+1
      XM=NH+1
      CF=XM/XN
      XN=0.
      DO 40 K=1,N1
      READ(20, ID) (PX(I), I=1,NC)
      XM=0.
      DO 20 J=1,NC
20  XM=XM+(PX(J)-HM(J))*(PX(J)-HM(J))
40  XN=XN+XM
      XN=DSQRT(XN)
      XN=CF*XN
50  CONTINUE
      FI=DMIN1(FI,XN)
      RETURN
      END

```

```
/*
```

```

//LKED.SYSLMOD DD DSN=CLOVIS(CLCFI),DISP=GLD
//CLOVIS EXEC FORTGCL,PARM,LKED='NCAL,LET'
//FORT.SYSIN DD *

```

```

      SUBROUTINE VERTS
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KB,LF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7

```

```

ID=1
IF(ND.GT.2) ID=NC*(NC+1)+1
C**** ARMAZENA EM DISCO D PCNTO INICIAL
WRITE(20,ID) (VX(I),I=1,NC)
XM=NC+1
XN=NC
CF=TE/(XN*SQRT(2.))
D1=CF*(DSQRT(XM)+XN-1)
D2=CF*(DSQRT(XM)-1)
NP=NR
IF(ND.LE.2) NP=NC
DO 40 K=1,NP
DO 20 J=1,NC
20 GX(J)=VX(J)+D2
GX(K)=VX(K)+D1
C**** ARMAZENA EM DISCO OS VERTICES GERADOS
WRITE(20,ID) (GX(I),I=1,NC)
40 CONTINUE
RETURN
END

```

```
/*
```

```

//LKED.SYSLMOD DD DSN=CLOVIS(VERTS),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *

```

```

SUBROUTINE TESIS
IMPLICIT REAL*8(A-H,O-Z)
COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7
IF(ND.EQ.0) GO TO 90
TX=0.
IF(NH.EQ.0) GO TO 30
CALL FUNHI
DO 10 K=1,NH
10 TX=TX+HI(K)*HI(K)
30 IF(NG.EQ.0) GO TO 70
CALL FUNGD
DO 50 K=1,NG
IF(GD(K).LE.0.) GO TO 50
TX=TX+GD(K)*GD(K)
50 CONTINUE
70 IF(ND.NE.5) TX=DSQRT(TX)
FX=TX
RETURN
90 CALL FUNCF

```

```

NF = NF + 1
RETURN
END

```

```

/*

```

```

//LKED,SYSLMOD DD DSN=CLOVIS(TEXTS),DISP=OLD

```

```

//

```

```

//CAREX JOB (3032,2410),MSGLEVEL=1,CLASS=C,TIME=3

```

```

//CLOVIS EXEC FORTGCL,PARM, LKED='NCAL,LET'

```

```

//FORT,SYSDIN DD *

```

```

SUBROUTINE DIRVS

```

```

IMPLICIT REAL*8(A-H,O-Z)

```

```

COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
5IX(20),IN,IO,NI,IO,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
6NS,N1,N2,N3,N4,IC,IA,IB,KL,IF,NK,NP,NU,NV,NX,IT,JN,
7MX,J1,J2,J3,J4,J5,J6,J7

```

```

C**** METODO DE DIRECOES VIAVEIS

```

```

NH=0

```

```

ND=5

```

```

NA=N3

```

```

CALL TEXTS

```

```

CALL GRADX

```

```

CALL SELE3

```

```

CALL FUNCF

```

```

CALL GRADF

```

```

CALL FUNGD

```

```

CALL GRADV

```

```

NI=0

```

```

NX=0

```

```

NV=0

```

```

E1=0.2

```

```

E2=0.1

```

```

AF=0.4

```

```

B1=0.5

```

```

B2=0.6

```

```

N4=0

```

```

IZ=8

```

```

NJ=NC+1

```

```

20 CONTINUE

```

```

EP=E1

```

```

40 CONTINUE

```

```

ND=4

```

```

IF(JN.EQ.0) GO TO 50

```

```

ID=4

```

```

IT=1

```

```

IK=3

```

```

CALL IMPRS

```

```
50 CONTINUE
   IF(NX.GE.NK) GO TO 160
   NX=NX+1
   CALL VINEX
   NH=0
   HD=DR(1)
   IF(IC.NE.1) GO TO 60
   ID=6
   CALL IMPRS
   RETURN
60 CONTINUE
   TE=-AF*EP
   IF(HO.GT.TE) GO TO 100
   DO 80 K=2,NJ
   J=K-1
   DR(J)=DR(K)
   VY(J)=VX(J)
80 CONTINUE
   ND=4
   TA=EP
   TB=E1
   TC=E2
   K1=NX
   NA=2
   CALL FUNCF
   CALL GRADF
   CALL SELE2
   NX=K1
   EP=TA
   E1=TB
   E2=TC
90 CONTINUE
   ID=0
   I=MOD(NX,IZ)
   IF(I.EQ.0) GO TO 20
   GO TO 40
100 CONTINUE
   IF(EP.GT.E2) GO TO 120
   TA=EP
   EP=0.
   CALL VINEX
   NH=0
   HD=DABS(H0)
   IF(H0.EQ.0.) GO TO 140
   EP=TA
120 CONTINUE
   IF(EP.LE.ZN) GO TO 140
   EP=B1*EP
   GO TO 40
140 CONTINUE
```



```

ID=3
IT=1
IK=3
CALL IMPRS
RETURN
160 CONTINUE
ID=7
CALL IMPRS
ID=4
RETURN
END

```

```

/*
//LKED,SYSLMOD DD DSN=CLOVIS(DIRVS),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM, LKED='NCAL,LET'
//FORT,SYSLIN DD *
  SUBROUTINE VINEX
    IMPLICIT REAL*8(A-H,O-Z)
    COMMON GG(20,100),GH(20,100),VX(100),GX(100),GD(20),
    1HI(20),FX,VF(100),VY(100),DR(100),VE(100),VZ(100),
    2HM(100),DG(100),DV(100),FY,ZR,ZE,XN,XM,ZN,ZF,TE,FI,
    3FL,FH,FX,EP,E1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,V6,V7,
    4VA(100),VB(100),VC(100),VD(100),PX(100),PY(100),
    5IX(20),IN,IO,NI,ID,IK,NC,NB,NA,NG,NR,IL,NT,ND,NH,
    6NS,N1,N2,N3,N4,IC,IA,IB,KL,NF,NK,NP,NU,NV,NX,IT,JN,
    7MX,J1,J2,J3,J4,J5,J6,J7
C**** PREPARA PARAMETROS PARA O SUBPROBLEMA DE DIRECOES VIAVEIS
  CALL FUNGD
  DO 40 K=1,NG
    GD(K)=GD(K)+EP
  40 CONTINUE
  ND=4
  CALL INDIC
  ID=IC
  60 CONTINUE
  NR=2*NC+IC+1
  70 CONTINUE
  N3=0
  N4=0
  NA=NR
  NH=NA
  NT=NC+NH+1
  NS=NA+1
  IC=NS+1
  ND=IC+1
  IL=NC+2
  N1=1
  N2=0
  NV=NB
  NB=-1
  IK=0

```

```

CALL GRADF
CALL GRADV
CALL VETEX
CALL CAREX
NB=NV
RETURN
END

```

```
/*
```

```

//LKED,SYSLMOD DD DSN=CLOVIS(VINEX),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM,LKED='NCAL,LET'
//FORT,SYSLMOD DD *

```

```

SUBROUTINE VETEX

```

```

IMPLICIT REAL*8(A-H,O-Z)

```

```

COMMON GG(20,100),GH(20,100),VV(100),GX(100),GD(20),
1AC(221),VS(300),YP(300),FY,ZD,ZE,XN,XM,ZN,ZF,
2TE,FI,FL,FH,FX,EP,S1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,
3V6,V7,VB(300),VH(300),IX(20),IN,IO,NI,ID,IJ,NC,NB,
4MT,NG,M1,IG,NT,M5,MP,M4,MN,NP,M2,M3,NL,IA,IB,KL,NF,
5NK,NE,NU,NV,NX,IT,JN,MX,J1,J2,J3,J4,J5,J6,J7

```

```

C**** TRANSFERE PARAMETROS DE DIRVS PARA SINEX

```

```

IC=ID

```

```

ID=1

```

```

NE=0

```

```

MM=2*NC+IC+3

```

```

NZ=2*NC

```

```

NY=NZ+1

```

```

NN=NC+2

```

```

NM=MM-1

```

```

DO 160 K=1,NN

```

```

I=0

```

```

L=K-2

```

```

DO 100 J=1,MM

```

```

IF(K,NE,1) GO TO 40

```

```

TA=0.

```

```

IF(J,LE,NZ) TA=1.

```

```

GO TO 80

```

```

40 CONTINUE

```

```

IF(K,NE,2) GO TO 60

```

```

TA=0.

```

```

IF(J,GT,NZ) TA=-1.

```

```

GO TO 80

```

```

60 CONTINUE

```

```

TA=0.

```

```

IF(J,LE,NZ) GO TO 80

```

```

TA=GX(L)

```

```

IF(J,EQ,NY) GO TO 80

```

```

TA=0.

```

```

IF(IC,EQ,0) GO TO 80

```

```

70 CONTINUE

```

```

I=I+1

```

```

      IF(IX(I).EQ.0) GO TO 70
      TA=GG(I,L)
80    CONTINUE
      AC(J)=TA
100   CONTINUE
      IF(K.EQ.1) GO TO 140
      IF(K.NE.2) GO TO 120
      AC(NM)=1.
      GO TO 140
120   CONTINUE
      M=L+NC
      AC(L)=1.
      AC(M)=-1.
140   CONTINUE
      AC(MM)=1.
      M3=M3+1
      NE=NE+1
      CALL SINEX
160   CONTINUE
      RETURN
      END

/*
//LKED.SYSLMOD DD DSN=CLOVIS(VETEX),DISP=OLD
//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *
      SUBROUTINE SINEX
      IMPLICIT REAL*8(A-H,O-Z)
      COMMON GG(20,100),GH(20,100),VV(100),GX(100),GD(20),
      1AC(221),VS(300),YP(300),FY,ZD,ZE,XN,XM,ZN,ZF,
      2TE,FI,FL,FH,FX,EP,S1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,
      3V6,V7,VB(300),VH(300),IX(20),IN,IO,NI,ID,IJ,NC,NB,
      4MT,NG,M1,IG,NT,M5,MP,M4,MN,NP,M2,M3,NL,IA,IB,KL,NF,
      5NK,NE,NU,NV,NX,IT,JN,MX,J1,J2,J3,J4,J5,J6,J7
C**** PREPARA OS DADOS PARA A SOLUCAO DO PL
      J=0
      IF(MX.EQ.3) J=NE
      IF(MX.EQ.3) GO TO 20
10    CONTINUE
      J=J+1
      READ(IN,11) (AC(I),I=1,NL)
11    FORMAT(4D20.13)
      WRITE(IO,11) (AC(I),I=1,NL)
20    CONTINUE
      IF(MN.EQ.1) AC(M4)=-AC(M4)
      S1=0.
      DO 30 K=1,MT
30    S1=S1+AC(K)
      AC(M5)=AC(NL)
      AC(NL)=AC(M4)
      AC(M4)=S1

```

```

      IF(J.NE.1) GO TO 70
      DO 60 N=1,NL
60    VS(N)=AC(N)
      GO TO 110
70    IJ=IJ+1
      IF(AC(M5).EQ.0.) GO TO 100
      AC(M5)=IJ
      WRITE(20'ID) (AC(N),N=1,M5)
      DO 90 K=1,M5
90    AC(K)=-AC(K)
      NT=NT+1
      GO TO 110
100   AC(M5)=IJ
110   WRITE(20'ID) (AC(N),N=1,M5)
130   CONTINUE
      IF(NB.EQ.-1.AND.M3.LT.IG) RETURN
      IF(J.LT.IG) GO TO 10
C****
C**** GERACAO DE VARIAVEIS DE FOLGA
C****
      DO 170 K=1,MP
      IJ=IJ+1
      IH=1
      DO 150 I=1,M5
150   AC(I)=0.
      IF(K.GT.M1) IH=-IH
      AC(K)=IH
      AC(M4)=IH
      AC(M5)=IJ
      WRITE(20'ID) (AC(N),N=1,M5)
170   CONTINUE
      IA=(NT+1)*M5+1
      KL=IA+(NL-2)*NL
      IB=IJ
      IC=0
C****
C**** GERACAO DA MATRIZ IDENTIDADE
C****
      DO 210 K=1,NL
      DO 190 I=1,NL
190   AC(I)=0.
      AC(K)=1.
      IJ=IJ+1
      VB(K)=IJ
      WRITE(20'ID) (AC(N),N=1,NL)
210   CONTINUE
      RETURN
      END
/*
//LKED,SYSLMOD DD DSN=CLOVIS(SINEX),DISP=OLD

```

```

//CLOVIS EXEC FORTGCL,PARM=LKED='NCAL,LET'
//FORT.SYSIN DD *
  SUBROUTINE CAREX
    IMPLICIT REAL*8(A-H,O-Z)
    COMMON GG(20,100),GH(20,100),VV(100),GX(100),GD(20),
    1AC(221),VS(300),YP(300),FY,ZD,ZE,XN,XM,ZN,ZF,
    2TE,FI,FL,FH,FX,EP,S1,E2,P1,P2,Q1,Q2,V1,V2,V3,V4,V5,
    3V6,V7,VB(300),VH(300),IX(20),IN,IO,NI,ID,IJ,NC,NB,
    4MT,NG,M1,IG,NT,M5,MP,M4,MN,NP,M2,M3,NL,IA,IB,KL,NF,
    5NK,NE,NU,NV,NX,IT,JN,IX,J1,J2,J3,J4,J5,J6,J7
C**** METODO DO SIMPLEX REVISADO
C****
    IF(NB.EQ.-1) GO TO 20
    MN=NP
    M1=M2
    M2=MP
    M3=NG
    MT=M1+M2+M3
    MP=M1+M2
    NT=NC+MP
    M4=MT+1
    NL=M4+1
    M5=NL+1
    IG=NC+1
    CALL SINEX
20 CONTINUE
    KF=1
    ID=1
    ZR=0.01
    IC=0
210 CONTINUE
C****
C**** ELIMINACAO DE VARIVAVEL ARTIFICIAL POSITIVA DA BASE NA 2A. FASE
C****
    IF(KF.EQ.1) GO TO 250
    DO 230 K=1,MT
    IF(VB(K).GT.IB) IC=VB(K)
230 CONTINUE
    IV=1
250 S1=0.
C****
C**** CALCULO DOS Z(J) - C(J)
C****
    ID=KL
    READ(20,ID) (YP(N),N=1,NL)
    ID=M5+1
    DO 310 K=1,NT
    READ(20,ID) (AC(N),N=1,M5)
    FIND(20,ID)
    DO 270 J=1,NL

```

```

      IF(AC(M5).EQ.VB(J)) GO TO 310
270 CONTINUE
      VX=0.
      DO 280 J=1,NL
280  VX=VX+YP(J)*AC(J)
      IF(VX.LE.ZR) GO TO 310
      IF(S1.GE.VX) GO TO 310
      DO 290 I=1,NL
290  VH(I)=AC(I)
      S1=VX
      IH=AC(M5)
      IV=0
310 CONTINUE
      S1=1.D60
      IF(IV.NE.0) GO TO 500
C****
C**** CALCULO DOS VETORES Y(H)
C****
      ID=IA
      DO 370 K=1,NL
      READ(20,ID) (AC(N),N=1,NL)
      FIND(20,ID)
      VX=0.
      DO 330 J=1,NL
330  VX=VX+AC(J)*VH(J)
      YP(K)=VX
      IF(K.GT.MT) GO TO 370
      IF(VX.GT.ZR) GO TO 350
C****
C**** DETERMINACAO DO VETOR QUE SAIRA' DA BASE
C****
      IV=IV+1
      GO TO 370
350  VX=VS(K)/VX
      IF(S1.LT.VX) GO TO 370
      S1=VX
      IG=K
370 CONTINUE
      IF(IV.GE.MT) GO TO 730
      IF(IC.NE.0) IG=IC
C****
C**** ALTERACAO DA BASE
C****
      VB(IG)=IH
C****
C**** CALCULO DA MATRIZ BASICA INVERSA
C****
      DO 450 I=1,NL
      ID=IA+I-1
      IV=ID

```

```

      DO 390 J=1,NL
      READ(20,ID) AC(J)
390  ID=ID+M4
      VX=AC(IG)
      DO 410 K=1,NL
410  AC(K)=AC(K)-YP(K)/YP(IG)*VX
      AC(IG)=VX/YP(IG)
      DO 430 J=1,NL
      WRITE(20,IV) AC(J)
430  IV=IV+NL
450  CONTINUE
C****
C****  CALCULO DAS SOLUCOES BASICAS VIAVEIS
C****
      ID=1
      READ(20,ID) (VH(N),N=1,NL)
      ID=IA
      DO 490 K=1,NL
      READ(20,ID) (AC(N),N=1,NL)
      FIND(20,ID)
      S1=0.
      DO 470 J=1,NL
470  S1=S1+AC(J)*VH(J)
      VS(K)=S1
490  CONTINUE
      IF(JN.EQ.2) GO TO 210
      WRITE(5,11)
11  FORMAT(/50X,'SOLUCOES BASICAS')
      WRITE(5,12) (VS(N),N=1,NL)
12  FORMAT(/30X,8F10.4)
      WRITE(5,13)
13  FORMAT(/50X,'VETORES BASICOS')
      WRITE(5,14) (VB(N),N=1,NL)
14  FORMAT(/40X,8F5.0)
      GO TO 210
500  IF(KF.NE.1) GO TO 530
      IF(DABS(VS(M4)).GT.ZR) GO TO 510
C****
C****  FIM DA PRIMEIRA FASE
C****
      KL=KL+NL
      KF=2
      IF(JN.EQ.2) GO TO 210
      WRITE(5,31)
31  FORMAT(/50X,'FIM DA PRIMEIRA FASE'//)
      GO TO 210
C****
C****  SOLUCAO INVIAVEL PARA C PROBLEMA
C****
510  NL=1

```

```

      IF(JN.EQ.2) RETURN
      WRITE(IO,3)
3     FORMAT(//50X,'O PROBLEMA NAO TEM SOLUCAO VIAVEL'//)
      RETURN
C****
C**** SOLUCAO OTIMA
C****
530  IF(MN.EQ.0) VS(NL)=-VS(NL)
      ID=2*M5
      IH=(NT+1)*M5
      IG=M5+NL
540  READ(20*ID) VX
      IF(VX.NE.0.) GO TO 550
      ID=ID+NL
      GO TO 670
550  DO 560 J=1,MT
      IF(VB(J).EQ.VX) GO TO 590
560  CONTINUE
      DO 570 K=1,MT
      IF(VB(K).EQ.-VX) GO TO 580
570  CONTINUE
      GO TO 650
580  VS(K)=-VS(K)
      VB(K)=-VB(K)
      GO TO 650
590  I=J
      DO 610 J=1,MT
      IF(VB(J).EQ.-VX) GO TO 630
610  CONTINUE
      GO TO 650
630  VS(I)=VS(I)-VS(J)
650  ID=ID+IG
670  IF(ID.LE.IH) GO TO 540
C****
C**** ORDENACAO DOS VALORES OTIMOS DE SAIDA
C****
      DO 680 K=1,MT
      AC(K)=VS(K)
      IX(K)=VB(K)
680  CONTINUE
      DO 720 K=1,MT
      I=K
      IA=IX(K)
      DO 700 J=K,MT
      IF(IA.LE.IX(J)) GO TO 700
      I=J
      IA=IX(J)
700  CONTINUE
      IX(I)=IX(K)
      VB(K)=IA

```



```
VS(K)=AC(I)
AC(I)=AC(K)
720 CONTINUE
IF(JN.EQ.2) RETURN
WRITE(5,4)
4 FORMAT(//50X,'SOLUCAO OTIMA'//30X,'VETORES BASICOS',20X,
1'SOLUCOES BASICAS'//)
DD 690 K=1,MT
IF(VB(K).GT.0.) WRITE(5,5) VB(K),VS(K)
690 CONTINUE
5 FORMAT(//38X,F4.0,15X,F10.4)
WRITE(5,7) VS(NL)
7 FORMAT(//30X,'VALOR DA FUNCAO OBJETIVO:',F10.4)
790 CONTINUE
RETURN
C****
C**** SOLUCAO ILIMITADA PARA O PROBLEMA
C****
730 NL=1
IF(JN.EQ.2) RETURN
WRITE(10,6)
6 FORMAT(//50X,'O PROBLEMA TEM SOLUCAO ILIMITADA')
RETURN
END
/*
//LKED,SYSLMOD DD DSN=CLOVIS(CAREX),DISP=OLD
//
```

BIBLIOGRAFIA

1. HIMMELBLAU, DAVID M., Applied Nonlinear Programming, Mcgraw-Hill Book Company, (1972).
2. POLAK, E., Computations Methods in Optimization, Academic Press, (1971).
3. TABAK, Daniel, IEEE Transactions on Systems, Man and Cybernetics, jan, (1973).
4. HUANG, H.Y. and CHAMBLISS, J.P. Journal of Optimization Theory and Applications, vol.11, n° 2, (1973).
5. GOLDSTEIN, A.A., Constructive Real Analysis, Harper, (1967).
6. ARMIJO, L., Minimization of functions heaving continuous partial derivatives, Pacific J.Math. 16,1-3, (1966).
7. DAVIES, SWANN and CAMPEY, ICI Note 64/3, (1964).
8. POWELL, M.J.D., Computer Journal 7:155, (1964).
9. CAUCHY, A.L., Méthode générale pour la résolution des systèmes d'équations simultanées, Compt.Rend. 25,536-38, (1847).
10. Hestenes, M.R. and Stiefel, E.L., J. Res.Natl.Bur.Std., B49:409, (1952). BECKMAN, F.S., The Solution of Linear Equations by the Conjugate Gradient Method, Mathematical Methods for Digital Computers, vol.1, John Wiley & Sons, Inc., (1960).
11. DIXON, L.C.W., Nonlinear Optimization, The English Universities Press Ltd., (1972).

12. DAVIDON, W.C., USAEC Doc. ANL - 5990 (rev.), Nov., (1959).
13. FLETCHER, R. and POWELL, M.J.D., Computer Journal, 6:163, (1963).
14. ADACHI, N., On Variable-Metric Algorithms, J.O.T.A., vol. 7, n^o 6, (1971).
15. PEARSON, J.D., Computer Journal, 13:171, (1969).
16. POWELL, M.J.D., On the Convergence of the Variable Metric Algorithm, Mathematics Branch, A.E.R.E., Harwell, Berkshire, England, out., (1969).
17. BROYDEN, C.G., Math. Computation, 21:368, (1967).
18. AOKI, M., Introduction to Optimization Techniques, The Macmillan Company, (1971).
19. POWELL, M.J.D., Computer Journal, 7:155, (1964) e 7:303, (1965).
20. ZANGWILL, W.I., Computer Journal, 10:293, (1967).
21. NELDER, J.A., and MEAD, R., Computer Journal, 7:308, (1964).
22. BOX, M.J., Computer Journal, 9:67, (1966).
23. ZANGWILL, W.I., Nonlinear Programming, Prentice-Hall, (1969).
24. ZOUTENDIJK, G., Methods of Feasible Directions, Elsevier, Amsterdam, (1960).
25. PAVIANI, D.A. and HIMMELBLAU, D.M., Constrained Nonlinear Optimization by Heuristic Programming, Operations Research, vol. 17, (1969).