UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA DE CAMPINAS

Este exemplar corresponde à redação final da tese defendida por Hermínio Simões Gomes e aprovada pela Comissão Julgadora em 18/09/1987.

Orientador

OTIMIZAÇÃO COM RESTRIÇÕES LINEARES E PRECONDICIONAMENTO PERIÓDICO

~~VOL. 1~~ : TEORIA E EXPERIMENTOS

HERMÍNIO SIMÕES GOMES

Orientador:
Prof. Dr. JOSÉ MÁRIO MARTÍNEZ

Tese apresentada à Faculdade de Engenharia de Campinas da Universidade Estadual de Campinas - UNICAMP, como parte dos requisitos exigidos para obtenção do título de DOUTOR EM ENGENHARIA ELÉTRICA.

Setembro/87

À minha família.

A G R A D E C I M E N T O S

Ao Prof. José Mário Martínez, pela sua paciência e extrema boa vontade, na orientação deste trabalho.

Aos professores da FEE e do IMECC, com quem pude discutir diversos aspectos deste trabalho.

Aos professores Secundino Soares Filho e Adriano Alber de França M. Carneiro, pela paciência na transposição de um modelo hidrelétrico para ser resolvido pelo sistema BUGRE.

A todos os colegas que, direta ou indiretamente, me ajudaram neste trabalho.

À Srta. Célia Dorázio, pela datilografia do primeiro volume.

À Srta. Elza Aoki, pela datilografia do segundo volume.

Ao Sr. Sidney Cunha pelos desenhos dos problemas-testes.

À Universidade Federal do Pará e à CAPES/PICD pelo apoio financeiro.

À minha esposa, pelo seu carinho, amor e compreensão durante este árduo trabalho.

R E S U M O

Propõe-se um algoritmo para otimização com restrições lineares e variáveis canalizadas que usa precondicionamento periódico para solução dos sistemas lineares. O algoritmo é do tipo gradientes conjugados com projeção e faz uso de fatorações ortogonais esparsas para o precondicionamento.

Uma coleção de testes é apresentada. É feita uma comparação, no caso de problemas lineares, com resultados obtidos pelo sistema MINOS.

I N D I C E

INTRODUÇÃO GERAL..... 1

CAPÍTULO I - OTIMIZAÇÃO COM VARIÁVEIS CANALIZADAS E O PROBLEMA GERAL DE OTIMIZAÇÃO COM RESTRIÇÕES LINEARES..... 4

1.1. Introdução..... 5

1.2. Algoritmo para Resolver um Problema de Programação Não-Linear com Variáveis Canalizadas e a Estratégia das Restrições Ativas..... 5

1.3. Problema de Programação Não-Linear com Restrições Lineares de Igualdade e Variáveis Canalizadas..... 10

1.4. Problema de Programação Não-Linear com Restrições Lineares e Variáveis Canalizadas..... 12

1.5. Estratégia do Gradiente Reduzido..... 14

CAPÍTULO II - A FATORAÇÃO ORTOGONAL E A SOLUÇÃO DO PROBLEMA GERAL DE OTIMIZAÇÃO COM RESTRIÇÕES LINEARES..... 20

2.1. Introdução..... 21

2.2. Obtenção dos Gradientes das Funções em y e das Direções de Busca..... 21

2.3. Decomposição Ortogonal..... 23

2.4. Precondicionamento de Sistemas Lineares..... 25

2.5. Solução dos Sistemas Lineares por Gradientes Conjugados..... 29

2.6. Escolha de uma Base Não-Singular..... 30

CAPÍTULO III - O SISTEMA BUGRE..... 32

3.1. Introdução..... 33

3.2. Programa para Otimização com Restrições Lineares..... 33

3.3. Precondicionamento Periódico..... 34

3.4. A Descrição do Sistema Bugre..... 35

3.5. Vantagens e Desvantagens do Sistema BUGRE..... 38

CAPÍTULO IV - TESTES COMPUTACIONAIS..... 40

4.1. Introdução..... 41

4.2. Comentários sobre Cada Teste..... 83

CONCLUSÃO GERAL..... 89

APÊNDICE 1..... 92

REFERÊNCIAS BIBLIOGRÁFICAS..... 95

INTRODUÇÃO GERAL

A proposta deste trabalho é introduzir um novo método para solução de problemas de otimização com restrições lineares. A idéia do trabalho surgiu da tentativa de generalização de um algoritmo de otimização, com função objetivo quadrática e com variáveis canalizadas, proposto por Polyak em [27], para trabalhar com função objetivo não-quadrática. Ver também O'Leary [26].

A transformação do problema geral com restrições lineares para uma sequência de problemas com variáveis canalizadas é dada por Bertsekas em [5] utilizando-se um método tipo Newton-Projetado.

Nossa proposta de trabalho é, basicamente, um algoritmo que baseia-se num método tipo gradientes conjugados [4, 10, 18, 19] com projeção, em vez do método tipo Newton-Projetado como proposto por Bertsekas.

As justificativas para o trabalho são diversas: os métodos mais comumente usados utilizam grande espaço de memória e em muitos problemas práticos chega-se no limite de espaço de memória disponível na máquina. O algoritmo e programa computacional proposto neste trabalho é superior no referente à memória aos tradicionalmente usados, como é o caso do Sistema MINOS [25]. Este último utiliza decomposição LU esparsa [3] para as inversões matriciais, podendo, dependendo da estrutura das restrições, apresentar muito "fill-in" (enchimento) e possíveis problemas de estabilidade.

Em nossa proposta, a "inversão" é feita utilizando-se processos iterativos (de pouca memória) com precondicionamento ortogonal [8, 11, 12, 28]. Propomos assim, melhorar o condicionamento da matriz através de uma fatoração ortogonal [14] incompleta e depois aplicar um processo iterativo na resolução dos sistemas lineares que surgem no método. Este processo, embora mais caro, em tempo, do que a fatoração LU, apresenta a vantagem de ser bastante estável, já que as fatorações ortogonais são mais estáveis do que as fatorações LU.

Outra vantagem da abordagem aqui apresentada é a de não utilização de variáveis de folga, constituindo-se uma forma mais elegante e econômica para trabalhar.

Outro fato digno de nota é que os sistemas que usam fatoração LU não podem, em geral, estabelecer, a priori, a quantidade de memória necessária para resolver um dado problema. Esta dificuldade foi superada, em nosso programa computacional, utilizando-se uma fatoração ortogonal incompleta por faixas, podendo-se fixar um dado número de faixas e consequentemente a memória a ser usada.

Uma outra vantagem, ainda, reside no fato de fazer-se o precondicionamento ortogonal somente em algumas bases e não em todas. A periodicidade do precondicionamento representa uma grande economia em tempo.

O algoritmo apresentado tem a vantagem de caminhar por dentro do convexo, quando isso é necessário, gerando, muitas vezes, um número bem menor de inversões de base, e consequentemente menos tempo de computação.

O texto é apresentado em dois volumes, o primeiro contém a teoria e os experimentos; o segundo é um manual técnico para uso do programa computacional, Sistema BUGRE, que desenvolvemos. O primeiro volume contém no capítulo final uma série de experimentos e comparações com os resultados obtidos pelo Sistema MINOS. O resultado de cada teste é amplamente discutido.

Por questões de simplificação, partimos da descrição do algoritmo do problema canalizado e depois passamos ao problema com restrições lineares gerais.

CAPÍTULO I

OTIMIZAÇÃO COM VARIÁVEIS CANALIZADAS E O PROBLEMA GERAL DE OTIMIZAÇÃO COM RESTRIÇÕES LINEARES

1.1. INTRODUÇÃO

Neste capítulo fornecemos um algoritmo para solução do problema de programação não linear com variáveis canalizadas. Mostramos o princípio da estratégia das restrições ativas e demonstramos um teorema em que se apóia a referida estratégia. O algoritmo mostrado é a base para um algoritmo mais geral, com restrições de igualdade, e este, por sua vez, é básico para outro, mais geral, com restrições de desigualdades.

Mostramos também, neste capítulo, como transformar um problema com restrições de igualdade para um problema do tipo "canalizado".

São apresentadas as condições de regularidade para o problema com restrições. Essas condições são muito importantes, pois estabelecem uma fronteira de validade para os algoritmos apresentados.

Preliminares do problema geral de minimização com restrições gerais lineares e variáveis canalizadas são apresentadas. Procurou-se, na medida do possível, uma simplificação do uso de índices e partições matriciais.

1.2. ALGORITMO PARA RESOLVER UM PROBLEMA DE PROGRAMAÇÃO NÃO-LINEAR COM VARIÁVEIS CANALIZADAS E A ESTRATÉGIA DAS RESTRIÇÕES ATIVAS

Considere o seguinte problema:

$$\begin{aligned} &\text{Minimizar} \quad f(x) \\ &\text{Sujeito a} \quad \ell \leq x \leq u \end{aligned} \tag{1.1}$$

onde f é uma função não-linear, diferenciável e estritamente convexa, x, ℓ, u são vetores do $\mathbb{R}^n$.

O problema (1.1) tem solução única, porém, não podemos, em geral, resolvê-lo em um número finito de passos. No caso de f quadrática positiva definida, (1.1) poderia ser resolvido em um número finito de passos por um processo tipo gradientes conjugados, veja [18, 26, 27]. O algoritmo para resolver (1.1) com f quadrática e positiva definida é dado no Apêndice 1.

Podemos resolver o problema (1.1) por uma sequência finita de problemas do tipo:

$$\begin{aligned} &\text{Minimizar} \quad f(x) \\ &\text{Sujeito a} \quad x \in V \end{aligned} \tag{1.2}$$

onde V é uma variedade linear. Isto pode ser conseguido através da estratégia das restrições ativas descritas a seguir.

Considere o problema (1.1). Seja I um sub-conjunto de $\{1,\ldots,n\}$ e F_I e F_I' os seguintes conjuntos:

$$F_I = \{\ell \leq x \leq u \mid x_i = \ell_i \text{ para } i \in I \text{ e } \ell_j < x_j < u_j \text{ para } j \notin I,\ i,j = 1,\ldots,n\}$$

$$F_I' = \{\ell \leq x \leq u \mid x_i = u_i \text{ para } i \in I \text{ e } \ell_j < x_j < u_j \text{ para } j \notin I,\ i,j = 1,\ldots,n\}$$

Dizemos que F_I (ou F_I') é uma "face" de um politopo com dimensão igual a dimensão da menor variedade linear que contêm F_I (ou F_I').

Pela definição, temos que:

$$F_I \cap F_J = \emptyset \quad \text{se} \quad I \neq J$$

$$F_I' \cap F_J' = \emptyset \quad \text{se} \quad I \neq J$$

$$S = \{\ell \leq x \leq u \mid x \in \mathbb{R}^n\} = \bigcup_{I \subset \{1,\ldots,n\}} F_I$$

Suponhamos que (x^k) é uma sequência gerada por um algoritmo para resolver (1.1), que todo ponto da sequência é factível e que (x^k) satisfaz os seguintes axiomas:

(i) Se x^k não é um ponto de Kuhn-Tucker [20], então x^{k+1} está definido e $f(x^{k+1}) < f(x^k)$.

(ii) Se x^{k+1} está definido e $x^k \in F_I$, então uma das seguintes possibilidades é verificada:

a) $x^{k+1} \in F_I$

b) $x^{k+1} \in F_J$ e $\dim(F_J) < \dim(F_I)$

c) x^k é o mínimo de f sobre F_I.

(iii) Para cada k, I, $x^k \in F_I$ existe um $\ell > k$ tal que $x^\ell \notin F_I$, ou x^ℓ é um ponto de Kuhn-Tucker.

Considere os axiomas acima e o teorema a seguir, também para F_I'. Utilizando os axiomas acima, podemos provar o seguinte teorema:

TEOREMA 1: Toda sequência gerada por um algoritmo que satisfaz os axiomas (i)-(iii) termina após um número finito de passos. (Ou seja, existe um $\bar{k}$ tal que $x^{\bar{k}}$ é um ponto de Kuhn-Tucker.)

PROVA: Provaremos que para cada sub-conjunto I do conjunto $\{1,\ldots,n\}$, o conjunto dos pontos x^k que estão em F_I é finito.

A prova é feita por ν, a dimensão de F_I. Para $\nu = 0$ $\dim F_I = \nu$ implica que F_I contém exatamente um ponto e assim a proposição é verdadeira.

Suponha que para cada F_I com $\dim F_I \leq \nu-1$, o conjunto dos x^k pertencentes a F_I é finito.

Por contradição, suponha, agora, que $\dim F_I = \nu$ e $x^k \in F_I$ para todo $k \in \mathbb{Z}_1$, onde $\mathbb{Z}_1$ é um conjunto infinito de inteiros. Seja k_1 o primeiro índice do conjunto $\mathbb{Z}_1$. Pela asserção de (iii) existe um $q_1 > 0$ tal que $x^{k_1+q_1} \notin F_I$. Se $x^{k_1+q_1-1}$ é o mínimo de f em F_I, então $f(x^\ell) < f(x^{k_1+q_1-1})$ para todo $\ell > k_1$ e também $x^\ell \notin F_I$ para todo $\ell > k_1$.

Agora se $x^{k_1+q_1-1}$ não é o mínimo de f em F_I, então por (ii) tem-se que $x^{k_1+q_1} \in F_{J_1}$, onde $\dim F_{J_1} \leq \nu-1$. Considere ainda $k_2 > k_1+q_1-1$ com $k_2 \in \mathbb{Z}_1$, usando o mesmo raciocínio acima, concluimos que $x^{k_2+q_2} \in F_{J_2}$, onde $\dim F_{J_2} \leq \nu-1$. Repetindo este argumento construimos um conjunto infinito $\{x^{k_i+q_i}\}$ contido na reunião $\cup \{F_I \,/\, \dim F_I \leq \nu-1\}$. Isto contradiz a hipótese indutiva.

Isto significa que se resolvermos (1.2), dentro de uma certa tolerân-

tir que (1.1) será resolvido em um número finito de passos.

É fácil ver que o teorema anterior é generalizável para o caso geral de restrições lineares de desigualdade. Por isso, não o repetiremos no capítulo correspondente.

A seguir mostramos um algoritmo tipo gradientes conjugados para solução de (1.1).

- ALGORITMO 1

PASSO 1 : Escolha um ponto x_1, $\ell \leq x_1 \leq u$, arbitrário e calcule $r_1 = -\nabla f(x_1)$.

PASSO 2 : Seja I o conjunto de índices i tais que

$$x_1^i = \ell^i \quad e \quad r_1^i \leq 0$$

ou

$$x_1^i = u^i \quad e \quad r_1^i \geq 0$$

Se $r_1^i = 0$ para todos os índices não pertencentes a I, então x_1 é o mínimo de f sobre $\ell \leq x \leq u$ e o processo termina.

PASSO 3 : (Projeção)

$$\text{Faça } d_1 = r_1 = \begin{cases} 0 & \text{se } i \in I \\ r_1^i & \text{se } i \notin I \end{cases}$$

PASSO 4 : (Solução do problema na variedade)
Comece com $k = 1$;

4.1. Calcule α_k de modo que

$$f(x_k+\alpha_k d_k) = \underset{\alpha}{\text{Mínimo}}\ f(x_k+\alpha d_k)$$

Faça

$$x_{k+1} = x_k+\alpha_k d_k$$

$r_{k+1} = -\nabla f(x_{k+1})$

Se x_{k+1} não é factível vá para o passo 5.

Se $r^i_{k+1} = 0$ para todo índice i não pertencente a I, faça

$x_1 = x_{k+1}$ (reinicialização)

$r_1 = -\nabla f(x_1)$

e vá para o passo 2, senão calcule uma nova direção

$$d_{k+1} = \begin{cases} 0 & \text{se } i \in I \\ \text{(direção obtida por algum processo do tipo gradientes conjugados) (ver [18])} & \text{se } i \notin I \end{cases}$$

faça $k = k+1$ e vá para o passo 4.1.

PASSO 5 : Seja J o conjunto dos índices j tais que

$x^j_{k+1} < \ell \quad$ ou $\quad x^j_{k+1} > u$

Faça

$\bar{\alpha}_k =$ Mínimo $\{\bar{\alpha}_\ell, \bar{\alpha}_u\}$

onde

$$\bar{\alpha}_\ell = \text{Mínimo} \left\{ \frac{\ell^j - x^j_k}{d^j_k}, \quad j \in J, \quad d^j_k > 0 \right\}$$

$$\bar{\alpha}_u = \text{Mínimo} \left\{ \frac{x^j_k - u^j_k}{d^j_k}, \quad j \in J, \quad d^j_k < 0 \right\}$$

Faça

$x_1 = x_k + \bar{\alpha}_k d_k$

$r_1 = -\nabla f(x_1)$

Redefina I como sendo o conjunto dos índices i tais que $x_1^i = \ell^i$ ou $x_1^i = u^i$. Se $r_1^i = 0$ para todos os índices não em I, então vá para o passo 2 onde I será redefinido novamente, senão vá para o passo 3.

Evidentemente que, dentro do passo 4, as alternativas podem ser as mais diversas. Além dos diversos métodos de gradientes conjugados, poderíamos escolher outras alternativas. Ver [5, 6].

No caso de a função f(x) ser não convexa podemos, em certos casos, obter um mínimo local do problema (1.2). Podemos, portanto, relaxar a exigência de f estritamente convexa em (1.1) em troca da obtenção de uma solução que seja apenas local.

Em última análise, a solução de (1.1), trata-se de uma política de obtenção e solução de diversos problemas tipo (1.2), até que um deles seja equivalente ao problema (1.1).

1.3. PROBLEMA DE PROGRAMAÇÃO NÃO-LINEAR COM RESTRIÇÕES LINEARES DE IGUALDADE E VARIÁVEIS CANALIZADAS

Considere o problema abaixo:

$$\begin{aligned} &\text{Minimizar} && f(x) \\ &\text{sujeito a} && Ax = b \\ & && \ell \leq x \leq u \end{aligned} \qquad (1.3)$$

com A(mxn), $m < n$, f continuamente diferenciável e estritamente convexa. Estamos assumindo que as canalizações podem ser generalizadas e que podemos ter $\ell = -\infty$ e/ou $u = +\infty$.

Seja x_k um ponto factível de (1.3) e uma aproximação da solução na iteração k, de modo que $Ax_k = b$ e $x_k^i = \ell^i$ ou u^i para $i \in I$.

Construindo a matriz $\bar{A}$ de restrições ativas em x_k, teremos:

$$\bar{A} = \begin{bmatrix} A \\ \hline E_I \end{bmatrix}_{[(n+ncan)\,x\,n]}$$

onde ncan é o número canalizações ativas em x_k =

= número de elementos em I

Se $m + ncan < n$ podemos "completar" a matriz retangular $\bar{A}$, de maneira que esta fique quadrada. Isto pode ser feito acrescentando-se, de modo adequado, linhas correspondentes às restrições de canalizações não-ativas.

A matriz $\tilde{A}$ fica da seguinte forma:

$$\tilde{A} = \begin{bmatrix} A \\ \hline E \end{bmatrix} = \begin{pmatrix} \text{a menos de} \\ \text{uma permutação} \\ \text{de linhas e} \\ \text{colunas} \end{pmatrix} \left[\begin{array}{c:c} B_{qxq} & N_{qxp} \\ \hline 0_{pxq} & \tilde{I}_{pxp} \end{array}\right]$$

onde E contém algumas linhas da matriz identidade I(nxn), que estão, em geral, desordenadas. A matriz $\tilde{I}_{pxp}$ será uma permutação da matriz identidade I_{pxp}. A partição de A em [B : N] é puramente conceitual, e significa dizer que podemos selecionar q colunas de A de modo que B_{qxq} (matriz das "variáveis básicas") seja inversível. As demais colunas de A formam a matriz das variáveis ditas "não-básicas" N_{qxp}.

Se conseguirmos esta partição (isto sempre é possível se x_k é regular [9]) poderemos inverter a matriz $\tilde{A}$. Computacionalmente, a inversão de $\tilde{A}$ é realizada necessitando-se somente da inversão de B.

Pelo exposto, podemos escrever:

$$\tilde{A}\, x_k = \begin{bmatrix} b \\ \hline lim^i \end{bmatrix} \qquad i \in I$$

$$\ell^j < x_k^j < u^j \qquad j \notin I ,$$

onde $lim^i = \ell^i$ ou u^i.

Consideremos agora a mudança de variáveis: $y = \tilde{A}x$ $(x = \tilde{A}^{-1}y)$ O problema original fica transformado em:

$$\text{Minimizar} \quad f[(\tilde{A})^{-1}y] = \phi(y)$$

$$\text{sujeito a} \quad \begin{bmatrix} y^b \\ \hline y^i \end{bmatrix} = \bar{b} = \begin{bmatrix} b \\ \hline \lim^i \end{bmatrix} \quad i \in I \qquad (1.4)$$

$$\ell^j \leq (\tilde{A})_j^{-1}\, y^j < u^j \,, \qquad j \notin I$$

Chegamos a um problema restrito, localmente mais simples que (1.3).

PROPOSIÇÃO: Se d_k é uma direção de descida para (1.6), então $\tilde{d}_k = \tilde{A}^{-1}d_k$ é uma direção de descida para o problema 1.4.

PROVA: Por hipótese, $\phi(y_{k+1}) < \phi(y_k)$, ou seja, $\phi(y_k+\alpha_k d_k) < \phi(y_k)$.

Colocando em termos de x temos:

$$f[(\tilde{A})^{-1}\,(y_k+\alpha_k d_k)] < f[(\tilde{A})^{-1}y_k]$$

Como $(\tilde{A})^{-1}y_k = x_k$ e, colocando $\tilde{d}_k = (\tilde{A})^{-1}d_k \neq \vec{0}$ vem que

$$f(x_k + \alpha_k \tilde{d}_k) < f(x_k) \,. \qquad \blacksquare$$

1.4. PROBLEMA DE PROGRAMAÇÃO NÃO-LINEAR COM RESTRIÇÕES LINEARES E VARIÁVEIS CANALIZADAS

Considere o seguinte problema

$$\begin{aligned} &\text{Minimizar} && f(x) \\ &\text{sujeito a} && Ax \left\{ \begin{matrix} \leq \\ \geq \\ = \end{matrix} \right\} b \qquad (1.5) \\ & && \ell \leq x \leq u \end{aligned}$$

Com $A(m\times n)$, $b \in \mathbb{R}^m$, $\ell, u, x \in \mathbb{R}^n$, f diferenciável e estritamente convexa. Estamos admitindo que ℓ e u podem ser canalizações generaliza-

das, ou seja, $\ell = -\infty$ e/ou $u = +\infty$. Considere (1.5) só contendo pontos regulares.

Considere, de agora em diante, que estamos em um ponto $x = x_k$ (numa iteração k) factível para o problema (1.5).

Sejam I e J os conjuntos dos índices das restrições gerais ativas em $x = x_k$ e das canalizações ativas em $x = x_k$ respectivamente. Ou seja

$$I = \{ i \mid A_i x^k = b^i \}$$

$$J = \{ j \mid x^j = \ell^j \text{ ou } x^j = u^j \}$$

A matriz das restrições ativas de (1.7) fica do seguinte modo:

$$\tilde{A} = \begin{bmatrix} A_I \\ \hline E_J \end{bmatrix}_{(na + ncan) \times n}$$

na = número de restrições próprias ativas
ncan = número de canalizações ativas

onde A_I é a partição de A cujos índices linhas estão em I e E_J é a matriz formada pelos vetores e_j^T, $j \in J$, onde e_j são as colunas j da matriz identidade.

Se o número de linhas de $\tilde{A}$ é igual ao número de colunas n, então $x = x_k$ é um vértice do convexo de (1.7) e $\tilde{A}$ é inversível. Porém, se $na + ncan < n$, $\tilde{A}$ é não-quadrada. Podemos, então, redefinir $\tilde{A}$ de modo que se possa conseguir uma base inversível.

Redefinindo $\tilde{A}$, temos:

$$\tilde{A} := \begin{bmatrix} \tilde{A} \\ \hline E_N \end{bmatrix} = \begin{bmatrix} A_I \\ \hline E_J \\ \hline E_N \end{bmatrix}$$

onde E_N são linhas da matriz identidade adequadamente escolhidas com índices não-pertencentes a I.

Com a hipótese de regularidade, podemos garantir que é sempre possível fazer uma escolha de E_N de modo que $\tilde{A}$ seja não-singular. Isto decorre do fato de que $\tilde{A}$ pode ser reescrita na forma

$$\tilde{A} \equiv \begin{bmatrix} B_{na \times na} & N_{na \times (n-na)} \\ 0_{(n-na) \times na} & \bar{I}_{(n-na) \times (n-na)} \end{bmatrix}$$

onde a partição $[B : N]$ é uma permutação das colunas de A_I, e como A_I tem posto *na*, podemos agrupar *na* colunas linearmente independentes para formar a matriz B. A matriz $\bar{I}$ é uma permutação de linhas e colunas de matriz identidade de ordem (n-*na*).

A escolha das colunas de A_I para formar B deve ser de modo que os índices das colunas de A_I, candidatas para formar B, devem estar fora do conjunto J das canalizações ativas.

A solução do problema (1.5) pode ser encarada como a solução de diversos problemas do tipo (1.4).

Para simplificar a exposição do algoritmo para solução de (1.5) explicamos, a seguir, uma estratégia.

1.5. ESTRATÉGIA DO GRADIENTE REDUZIDO

Considere o problema (1.5) disposto do seguinte modo:

$$\begin{aligned} &\text{Minimizar} \quad f(x) \\ &\text{sujeito a} \quad x \in P \end{aligned} \tag{1.6}$$

onde P pode ser visto como o seguinte politopo:

$$P = \{x \in \mathbb{R}^n \mid a_i^T x \geq b_i, \quad i = 1,\ldots,m \quad e \quad c_i^T x = d_i, \quad i = 1,\ldots,q\}$$

Seja x^k a k-ésima aproximação da solução. Suponha, sem perda de generalidade, que:

$$a_i^T x = b_i \qquad i = 1,\ldots,p$$

Suponha também que o conjunto de vetores dado por $\{a_1,\ldots,a_p,c_1,\ldots,c_q\}$ é linearmente independente, de modo que x^k é um ponto factível regular em P.

Fazendo a mudança de variáveis, transformamos, localmente, o problema original em um problema de minimização com variáveis canalizadas.

Definimos, portanto, novas variáveis y:

$$y_i = a_i^T x \quad , \qquad i = 1,\ldots,p$$

$$y_{p+j} = c_j^T x \, , \qquad j = 1,\ldots,q$$

Definiu-se assim a matriz de transformação A dada por:

$$A = \begin{pmatrix} a_1^T \\ \vdots \\ a_p^T \\ c_1^T \\ \vdots \\ c_q^T \end{pmatrix}$$

Suponha, sem perda de generalidade, que as primeiras $p+q$ colunas de A são linearmente independentes. Assim, $A = (B : N)$ onde B é uma matriz quadrada não singular. Completamos a definição das variáveis y_i restantes, do seguinte modo:

$$y_{p+q+1} = e_{p+q+1} \qquad i = 1,\ldots,n-p-q$$

Podemos, deste modo, escrever $y = Mx$ onde

$$M = \begin{pmatrix} B & N \\ 0 & I \end{pmatrix}$$

sendo M uma matriz não-singular.

Sendo assim o problema original pode ser escrito, na vizinhança de x^k como

$$\begin{aligned} &\text{Minimizar} \quad \Phi(y) \\ &\text{sujeito a} \quad y_i \geq b_i \qquad i = 1,\ldots,p \\ &\qquad\qquad y_i = d_{i-p} \qquad i = p+1,\ldots,p+q \\ &\qquad\qquad y_i \ \text{livre para} \quad i = p+q+1,\ldots,n \end{aligned} \tag{1.7}$$

onde $\Phi(y) = f(M^{-1}y)$.

Fazendo $y^k = M^{-1}x^k$, as condições de Kuhn-Tucker para (1.6) são

$$\frac{\partial \Phi}{\partial y_i}(y^k) \geq 0 \qquad \text{para} \quad i = 1,\ldots,p \tag{1.8}$$

e

$$\frac{\partial \Phi}{\partial y_i}(y^k) = 0 \qquad \text{para} \quad i = p+q+1,\ldots,n \tag{1.9}$$

Se (1.8) e (1.9) são verificadas, então x^k é um ponto de Kuhn-Tucker do problema (1.6).

Se (1.9) é verificado e (1.8) não o é, então y^k é um ponto estacionário do problema:

$$\begin{aligned} &\text{Minimizar} \quad \Phi(y) \\ &\text{sujeito a} \quad y_i = b_i \qquad i = 1,\ldots,p \\ &\qquad\qquad y_i = d_{i-p} \qquad i = p+1,\ldots,p+q \\ &\qquad\qquad y_i \ \text{livre para} \quad i = p+q+1,\ldots,n \end{aligned} \tag{1.10}$$

e $z = (z_1,\ldots,z_n)^T$ é uma direção de descida para (1.7) desde que

$$\begin{aligned} z_i &= -\frac{\partial \Phi}{\partial y_i}(y^k) \quad \text{se} \quad \frac{\partial \Phi}{\partial y_i}(y^k) < 0 \qquad i = 1,\ldots,p \\ &= 0 \quad \text{caso contrário} \end{aligned} \tag{1.11}$$

Se (1.9) não é verificada, então qualquer direção z satisfazendo:

$$z_i = 0, \quad i = 1,\ldots,p+q$$

e

$$\sum_{i=p+q+1}^{n} z_i \frac{\partial \Phi}{\partial y_i}(y^k) < 0 \tag{1.12}$$

é uma direção de descida para (1.7) e qualquer ponto $\bar{y} = y^k + \lambda z$ satisfaz as restrições de (1.10).

A discussão acima delineia os procedimentos que devemos utilizar para detectar um ponto de Kuhn-Tucker e como obter uma direção de descida a partir de um ponto não-estacionário. Uma vez que uma direção de descida tenha sido obtida, ela pode ser transformada para o espaço original através de:

$$\omega_k = M^{-1} z \tag{1.13}$$

e, consequentemente,

$$x^{k+1} = x^k + \lambda_k \omega \tag{1.14}$$

é um ponto factível se λ_k for suficientemente pequeno.

A seguir propomos um algoritmo para resolver (1.6).

- ALGORITMO 2

Suponha que x^0 é um ponto regular arbitrário, aproximação inicial em P. Uma vez calculado x^k, os passos utilizados para calcular x^{k+1} são enumerados a seguir:

PASSO 1: Seja I_k o conjunto dos índices i tais que $a_i^T x^k = b_i$, para $i \in I_k$, $a_i^T x^k > b_i$ para $i \notin I_k$. Usando a mudança de variáveis definida nesta seção, teste as condições de Kuhn-Tucker em x^k. Se forem satisfeitas, pare.

PASSO 2: Se as equações (1.9) são satisfeitas, calcule ω_k usando (1.11) e (1.13). Vá para o passo 5.

PASSO 3: Se $I_k \neq I_{k-1}$ ou $k = 0$, faça $z_i = \frac{-\partial\Phi}{\partial y_i}(y^k)$ em (1.12). Calcule ω_k usando (1.13). Vá para o passo 5.

PASSO 4: Calcule z satisfazendo (1.12) utilizando alguma fórmula do tipo gradientes conjugados no espaço das variáveis y e utilize (1.13) para calcular ω_k. De fato, pode-se calcular diretamente ω_k, incorrendo em mínimo esforço computacional.

PASSO 5: Defina $\lambda_k^{max} = \max \{\lambda \mid [x^k, x^k+\lambda\omega] \subset P\}$. Determine $\lambda_k \in (0,\lambda_k^{max})$ tal que $f(x^k + \lambda_k\omega_k) << f(x^k)$.

PASSO 6: Faça $x^{k+1} = x^k+\lambda_k\omega_k$.

OBSERVAÇÕES:

Em nossa implementação do algoritmo, utilizamos a fórmula de Fletcher-Reeves no passo 4, devido ser esta a fórmula de gradientes conjugados com menor necessidade de memória.

As hipóteses de regularidade sobre x^k podem ser relaxadas permitindo-se um passo de comprimento nulo $\lambda_k = 0$ no passo 5 do algoritmo. Contudo, uma análise cuidadosa sobre os arredondamentos é necessária, neste caso, para prevenir-se contra perda de factibilidade.

Para funções quadráticas utilizamos a minimização unidimensional dada por

$$f(x^k+\lambda_k\omega_k) = \min \{f(x^k+\lambda\omega_k), \quad \lambda \in (0, \lambda_k^{max}]\}$$

com o intuito de ganhar vantagens sobre as propriedades de terminação finita dos processos de Gradientes Conjugados.

Para funções mais gerais utilizamos a regra de Armijo:

$$f(x^k+\lambda_k\omega_k) \leq f(x^k) + 10^{-4}\lambda_k \langle \nabla f(x^k), \omega_k \rangle$$

O processo unidimensional utilizado foi o de interpolação cúbica.

No próximo capítulo veremos as técnicas utilizadas para solução dos sistemas lineares que aparecem no algoritmo quando calcula-se os gradientes.

CAPÍTULO II

A FATORAÇÃO ORTOGONAL E A SOLUÇÃO DO PROBLEMA GERAL DE OTIMIZAÇÃO COM RESTRIÇÕES LINEARES

2.1. INTRODUÇÃO

Neste capítulo, discutimos e detalhamos o algoritmo, para minimizar uma função não linear sujeita a restrições lineares e variáveis canalizadas. Descrevemos a fatoração ortogonal ou fatoração QR que é utilizada para precondicionamento (explicado neste capítulo) dos sistemas lineares que surgem no processo. Estes sistemas decorrem do cálculo dos algoritmos e das direções de busca.

Dedicamos uma parte neste capítulo à fatoração ortogonal por rotações planas. Mostramos como é realizado o processo do ponto de vista teórico e computacional.

Para a solução dos sistemas lineares faz-se necessária uma partição da matriz das restrições ativas em matriz base, correspondente às variáveis básicas, e matriz das variáveis não-básicas. Esta partição exige a escolha de uma base não-singular; problema este abordado neste capítulo.

Alguns esclarecimentos também são feitos sobre o uso do método de gradientes conjugados de Hestenes-Stiefel para solução dos sistemas lineares precondicionados pela fatoracão ortogonal.

2.2. OBTENÇÃO DOS GRADIENTES DAS FUNÇÕES EM y E DAS DIREÇÕES DE BUSCA

Consideremos o algoritmo 2. A menos de permutação de linhas e de colunas a matriz $\tilde{A}$ de (1.4) pode ser escrita do seguinte modo:

$$\tilde{A} = \begin{bmatrix} B & N \\ 0 & I \end{bmatrix}$$

O gradiente da função $\phi(y)$ tem a seguinte forma:

$$\nabla\phi(y) = \nabla f[(\tilde{A})^{-1}y] = (\tilde{A})^{-T}\nabla f[(\tilde{A})^{-1}y] = (\tilde{A})^{-T}\nabla f(x)$$

ou ainda

$$(\tilde{A})^{T}\,\nabla\phi(y) = \nabla f(x)$$

Ou seja:

$$\begin{bmatrix} B^T & 0 \\ N^T & I \end{bmatrix} \begin{bmatrix} g\phi_B \\ g\phi_N \end{bmatrix} = \begin{bmatrix} g_B \\ g_N \end{bmatrix} \qquad \nabla f(x) = [g_B \quad g_N]^T$$

$$\nabla \phi(y) = [g\phi_B \quad g\phi_N]^T$$

Portanto:

$$B^T g\phi_B = g_B \quad \text{e} \quad N^T g\phi_B + g\phi_N = g_N$$

Logo, resolvendo um sistema linear com a matriz B^T chegamos a $g\phi_B$. Com a obtenção de $g\phi_B$ chegamos a $g\phi_N$ pela segunda relação.

De modo semelhante podemos determinar as direções de busca na variável x.

Em x teríamos

$$x_{k+1} = x_k + \alpha \bar{d}_k$$

premultiplicando por $\bar{A}$ temos

$$\bar{A}x_{k+1} = \bar{A}x_k + \alpha \bar{A} \bar{d}_k$$

ou

$$y_{k+1} = y_k + \alpha d$$

sendo

$$\bar{A} \bar{d}_k = d$$

Para determinar $\bar{d}_k$ temos que resolver o sistema linear acima. Assim:

$$\begin{bmatrix} B & N \\ 0 & I \end{bmatrix} \begin{bmatrix} \bar{d}_B \\ \bar{d}_N \end{bmatrix} = \begin{bmatrix} d_B \\ d_N \end{bmatrix}$$

e

$$B \bar{d}_B = d_B \,, \qquad \bar{d}_N = d_N$$

bastando determinar $\bar{d}_B$.

Vemos, então, que precisamos resolver sistemas lineares com as matrizes B e B^T. Estes sistemas lineares foram resolvidos, no programa que fizemos, por um processo de gradientes conjugados com precondicionamento. Utilizou-se decomposição ortogonal para precondicionar os sistemas lineares.

2.3. DECOMPOSIÇÃO ORTOGONAL

Uma decomposição ortogonal é uma sequência de transformações lineares de modo a fatorar uma matriz A na seguinte forma:

$$A = Q\,R$$

ou

$$Q^{-1}A = R \qquad \text{onde} \qquad Q^{-1} = Q^T$$

onde Q é uma matriz ortogonal e R é triangular superior. Se A for quadrada e inversível podemos resolver sistemas lineares do tipo

$$Ax = b \qquad e \qquad A^Tx = b$$

com a mesma decomposição. Com efeito

$$QRx = b$$

$$Rx = Q^Tb = \tilde{b} \quad \therefore \quad x = R^{-1}\tilde{b}$$

e

$$(QR)^Tx = b$$

$$R^TQ^Tx = b \qquad \text{ou} \qquad R^Ty = b \qquad \text{onde} \qquad y = Q^Tx$$

assim

$$y = R^{-T}b \qquad e \qquad x = Qy$$

O fato de termos o fator ortogonal Q permite que uma decomposição resolva os dois tipos de sistemas lineares.

O processo que utilizamos para fatorar A em QR foi o das rotações de Givens, que passamos a descrever.

A transformação da matriz A em uma forma triangular superior requer o anulamento sistemático dos elementos que estão abaixo da diagonal principal. Esse anulamento é feito elemento por elemento até que não sobre nenhum elemento abaixo da diagonal de A.

A quantidade de rotações necessárias para fatorar A completamente, depende da estrutura da matriz.

Considere, por simplificação, a matriz (2x2) extraída de A da seguinte forma:

$$\bar{A} = \begin{bmatrix} a_{ij} & a_{ik} \\ a_{pj} & a_{pk} \end{bmatrix} = \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \end{bmatrix}$$

onde deseja-se zerar o elemento a_{pj}. Os elementos da linha i e da linha p serão alterados pela transformação; consideramos, por simplificação, somente os elementos em uma coluna genérica k.

Pré-multiplicamos a matriz $\bar{A}$ por uma matriz ortogonal $\bar{Q}$ (2x2) de modo que zere a_{pj} (ou o elemento y_1). Vejamos como:

$$\bar{Q} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

onde $c = \cos\theta$, $s = \operatorname{sen}\theta$, θ = ângulo de rotação.

Em termos dos elementos de $\bar{A}$

$$c = x_1 \mid d, \qquad s = y_1 \mid d, \qquad d = \sqrt{x_1^2 + y_1^2}$$

$$\bar{Q}\,\bar{A} = \bar{R}$$

onde

$$\bar{R} = \begin{bmatrix} d & cx_2 + sy_2 \\ 0 & cy_2 - sx_2 \end{bmatrix}$$

A matriz $\bar{R}$ é então sobreposta à matriz A obedecendo as posições de seus elementos. Esta transformação linear é aplicada a todas as colunas k que tenham elementos não-nulos na linha i e/ou na linha p.

Desafortunadamente este processo pode ser longo e demorado se a matriz for pouco esparsa, já que estas transformações tornam possíveis elementos que eram zeros, em elementos não-nulos nas linhas onde estão agindo; produzindo o fenômeno de "Fill-in" (enchimento). O aparecimento de "fill-in" faz com que se requeira muita memória de computador para seu armazenamento. Uma matriz esparsa, com uma estrutura ruim, pode produzir muito "fill-in" prejudicando o armazenamento e a rapidez da fatoração.

2.4. PRECONDICIONAMENTO DE SISTEMAS LINEARES

A solução de um sistema linear pode tornar-se difícil, se a matriz do sistema for mal-condicionada. Métodos iterativos podem não ser uma boa escolha neste caso, pois o número de iterações pode tornar-se proibitivo.

Se a opção por um método iterativo foi feita, pode-se atenuar o problema de mal-condicionamento fazendo-se um tratamento a priori da matriz do sistema, melhorando o seu condicionamento. A este tratamento chamamos de precondicionamento.

Considere, por exemplo, o seguinte sistema:

$$Ax = b$$

Podemos fazer um precondicionamento do sistema multiplicando-se ambos os membros por uma matriz B, adequada, de modo que o sistema resultante tenha um número de condição, veja [8], menor que o do sistema original. Ou seja

$$BAx = Bb$$

ou

$$Cx = \bar{b}$$

onde $cond(C) << cond(A)$.

Como não dispomos, em geral, da inversa de A, podemos decompor A em fatores ortogonais

$$A = QR$$

assim

$$QRx = b$$

fazendo $y = Rx$ vem que

$$Qy = b .$$

O sistema $Qy = b$ é bem condicionado, ou melhor, é ortogonal, bastando fazer $y = Q^{T}B$ e temos a sua solução. Por retro-substituição obtemos x.

Este precondicionamento conduz diretamente à solução, porém é muito caro computacionalmente pelos motivos expostos na seção anterior.

Um bom procedimento é fazer um precondicionamento mais barato, que não conduz diretamente à solução mas que obtém, em primeiro lugar, uma aproximação razoável da solução. De posse desta solução podemos ir a um método iterativo (de pouca memória). Se a aproximação da solução fornecida pelo precondicionamento for boa, um método iterativo será mais conveniente do que um método direto tipo eliminação de Gauss, já que os métodos iterativos conservam a esparsidade da matriz e são mais fáceis de programar do que os métodos de eliminação; estes últimos exigem, em geral, estratégias de pivotamento e tratamento de "fill-in" (enchimento) um tanto difíceis.

Em nosso trabalho utilizamos o precondicionamento tipo faixa com rotações de Givens. Descrevemos a seguir o procedimento usado:

Considere o exemplo da matriz abaixo, que desejamos decompor:

x			x				
x	x			x			
		x			x		
			x				
	x			x			
		x			x	x	
	x					x	
					x		x

Em cada posição que contêm um "x" há um elemento não-nulo. Nosso objetivo é zerar os elementos abaixo da diagonal considerando-se para efeito de trabalho, somente uma faixa da matriz. Consideraremos neste exemplo somente 4 faixas. Para facilitar a explicação adotamos o círculo para elemento zerado e a letra "s" para um "fill-in" (enchimento), ou simplesmente "sujo".

Tomando o elemento na posição (1,1) como elemento pivô, podemos zerar o elemento imediatamente abaixo e a nova configuração ficará do seguinte modo:

x	s		x	s			
ⓧ	x		s	x			
		x			x		
			x				
	x			x			
		x			x	x	
	x					x	
					x		x

Como estamos considerando somente 4 faixas, o "s" na posição (1,5) simplesmente não importará para nós, porém para efeito explicativo não abandonaremos, por enquanto, os elementos fora da faixa.

A primeira coluna da matriz já está pronta, tomemos pois o elemento da posição (2,2) para pivô e podemos zerar o elemento da posição (5,2) numa passada e o elemento da posição (7,2) na passada seguinte. A configuração ficaria da seguinte forma:

x	s		x	s			
ⓧ	x		s	x		s	
		x			x		
			x				
	ⓧ		s	x		s	
		x			x	x	
	ⓧ		s	s		x	
					x		x

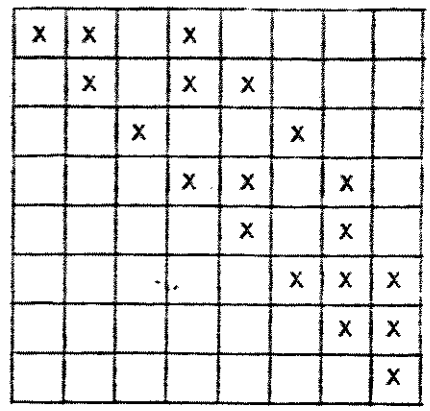
Tomando, agora, o elemento na posição (3,3) para pivô e fazendo a eliminação do elemento na posição (6,3) teremos:

x	s		x	s			
ⓧ	x		s	x			
		x			x	s	
			x				
	ⓧ		s	x		s	
		ⓧ			x	x	
	ⓧ		s	s		x	
					x		x

Continuando a repetição do procedimento até o final, teremos a seguinte configuração:

x	s		x		s		
ⓧ	x		s	x		s	
		x			x	s	
			x	s		s	
	ⓧ		ⓢ	x		s	
		ⓧ			x	x	s
	ⓧ		ⓢ	ⓢ		x	s
					ⓧ	ⓢ	x

Finalmente, considerando-se apenas as 4 primeiras faixas acima da diagonal, teremos uma matriz triangular superior da seguinte forma:



Os elementos acima da faixa foram desprezados. O fator obtido desta forma é uma aproximação da matriz R da fatoração ortogonal QA = R.

Esta fatoração melhora o condicionamento do sistema linear original [23].

Computacionalmente pode-se fazer a decomposição acima de uma maneira ligeiramente diferente e que também produz resultados bons. Pode-se, por exemplo, estabelecer dentro do processo o trabalho dentro de uma determinada faixa, acima e abaixo da diagonal, e elementos que caíssem fora da faixa seriam automaticamente eliminados. O processo geraria elementos "falsos", porém com a vantagem de um menor esforço computacional.

2.5. SOLUÇÃO DOS SISTEMAS LINEARES POR GRADIENTES CONJUGADOS

Podemos dizer que a solução de um sistema linear por gradientes conjugados é um método iterativo. Teoricamente, diversos algoritmos têm convergência finita [7], embora, computacionalmente não se possa conseguir a solução no número de passos previsto pela teoria.

Se estivermos trabalhando com um sistema linear grande e sem uso de precondicionamento, o método de gradientes conjugados apresenta uma solução razoável após um número em torno de 5n iterações [15]. É, portanto, impraticável a solução de sistemas lineares grandes, de uma maneira geral, por gradientes conjugados sem precondicionamento.

Considerando-se a matriz do sistema linear de ordem n e simétrica, o método de gradientes conjugados converge em no máximo n passos. Se a matriz apresenta autovalores repetidos então serão necessários apenas m passos, onde m é o número de autovalores distintos na matriz [18].

Se a matriz for ortogonal, o método atingirá a solução em um único passo. Podemos concluir que se a matriz é quase ortogonal, espera-se a convergência em bem menos que n passos.

A aplicação de um processo de precondicionamento tipo faixa e depois a aplicação de um método tipo gradientes conjugados, certamente conduzirá a um compromisso de esforços computacionais, desejando-se nem muitas rotações para fatorar e nem muitos passos de gradientes conjugados.

Para efeito de programa computacional, escolhemos o método de gradientes conjugados de Hestenes-Stiefel.

2.6. ESCOLHA DE UMA BASE NÃO-SINGULAR

Quando é feita a seleção das restrições ativas necessitamos particionar a matriz em dois grupos de colunas, sendo um grupo referente às variáveis básicas e o outro referente às variáveis não-básicas. No método Simplex essa partição surge automaticamente ao caminhar-se de uma base para outra, já que a mudança de base é feita seguindo-se um critério preciso de entrada e saída de variáveis da base. No referido método a base escolhida é sempre não-singular. A compreensão disto é facilmente vista geometricamente. O algoritmo proposto neste trabalho não é do tipo Simplex, já que pode "andar por dentro do convexo". O advento desta vantagem trás a perda da automaticidade na escolha das variáveis básicas e não-básicas. Isto significa que a cada novo ponto temos que escolher as variáveis básicas e não-básicas.

O problema citado por ser colocado de uma maneira precisa da seguinte maneira:

Considere a matriz $A = (A_{ij})$ de n colunas e m linhas, com $n >> m$. Temos que escolher m colunas das n existentes de modo que a matriz A possa ser colocada do seguinte modo:

$$A = [\ B_{m \times m} \qquad N_{m \times (n-m)}\ ]$$

A matriz B deverá ser não-singular. Qualquer processo que se possa imaginar é muito caro computacionalmente. Podemos fazer a escolha das colunas da matriz B de uma maneira *heurística*, barata, que na maioria dos casos serve. Esta escolha não garantiria a não-singularidade da matriz B.

A escolha que adotamos foi a seguinte: escolhe-se como primeira coluna aquela que apresenta o maior elemento, em módulo, na primeira linha, extrai-se esta coluna; escolhe-se como segunda coluna aquela que apresenta o maior elemento em módulo, na segunda linha, extrai-se esta coluna, a assim por diante.

Para exemplificar, considere a matriz abaixo:

$$A = \begin{bmatrix} 2 & 1 & -3 & 1 & 0 \\ 3 & 1 & 10 & 8 & 2 \\ -5 & 2 & 1 & 3 & -2 \end{bmatrix}$$

A matriz B escolhida de A seria

$$B = \begin{bmatrix} -3 & 1 & 2 \\ 10 & 8 & 3 \\ 1 & 3 & -5 \end{bmatrix}$$

Esta escolha dificilmente falha para matriz com elementos diversificados, podendo falhar para matrizes que trabalham com incidência e que é constituída por 1's e -1's. Neste caso a escolha heurística poderá ser mudada para uma escolha adequada ao problema particular.

No programa que elaboramos temos três opções para obtenção de uma base não-singular. Na primeira fazemos a escolha heurística como descrita; se falhar, fazemos uma nova escolha, como segunda opção, começando de trás para frente tornando-se as n primeiras colunas não-nulas da matriz reagrupada pelo método heurístico.

Geralmente se a primeira escolha falha, a segunda logra sucesso.

No segundo volume do texto apresentamos um exemplo devido a Maculan [21] em que a escolha heurística falha, e a segunda escolha logra sucesso.

No próximo capítulo veremos a estrutura do programa fonte para resolver o problema de programação não-linear, com restrições lineares e precondicionamento periódico.

CAPÍTULO III

O SISTEMA BUGRE

3.1. INTRODUÇÃO

Neste capítulo falamos do programa computacional BUGRE que elaboramos para solucionar o problema geral de otimização com restrições lineares e variáveis canalizadas. Detalhes de programação são comentados, entre eles, o do precondicionamento periódico, elemento chave para o seu bom desempenho.

Uma descrição resumida é feita das sub-rotinas do programa. Mostramos ainda algumas vantagens e desvantagens do sistema.

Este capítulo não pretende familiarizar o leitor sobre o funcionamento interno do programa, mas, apenas, dar uma noção geral do mesmo. Mais detalhes sobre o funcioanamento do programa e seus usos são dados no 2º volume do texto.

3.2. PROGRAMA PARA OTIMIZAÇÃO COM RESTRIÇÕES LINEARES

Elaborou-se um conjunto de sub-rotinas, o sistema BUGRE, em linguagem FORTRAN para o computador VAX-785 para resolver o problema abaixo pelo algoritmo proposto, algoritmo 2, com precondicionamento periódico.

$$\begin{aligned} &\text{Minimizar} \quad f(x) \quad \text{linear ou não linear} \\ &\qquad\qquad\qquad\quad \text{diferenciável e suave} \\ &\text{sujeito a} \quad Ax \left\{ \begin{array}{c} = \\ \leq \\ \geq \end{array} \right\} b \qquad\qquad (3.1) \\ &\qquad \ell \leq x \leq u, \quad \text{com } A \text{ esparsa} \end{aligned}$$

Utilizou-se, basicamente, o método de Gradientes Conjugados [10, 19] . As direções e gradientes são obtidos através de um método gradientes conjugados (Hestenes-Stiefel) com precondicionamento.

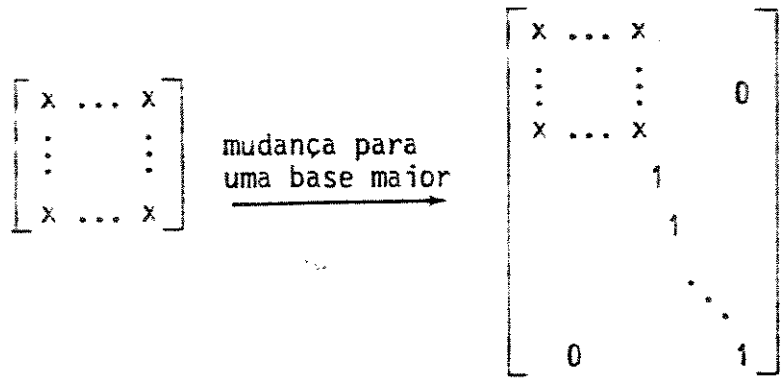
3.3. PRECONDICIONAMENTO PERIÓDICO

O programa BUGRE verifica, em uma dada base, se o número de iterações de gradientes conjugados para a solução do sistema linear já atingiu um certo número teto; se afirmativo, ele faz o precondicionamento, caso contrário ele usa o precondicionamento, a iteração anterior (base anterior) para esta nova iteração. Esta variável "gatilho" que dispara o precondicionamento, quando necessário, é fornecida pelo usuário, controlando-se, desta maneira, o compromisso entre o trabalho da fatoração ortogonal e o dispêndio de gradientes conjugados, para a solução dos sistemas lineares.

Conforme a estrutura da matriz do problema, pode-se "disparar o gatilho" mais rapidamente ou mais lentamente. Estruturas com muitos elementos iguais precisam menos iterações de gradientes conjugados, podendo-se colocar o gatilho para poucas iterações.

A passagem de uma base para outra implica, em geral, a troca de poucas colunas da matriz base. Se passamos de um vértice para outro adjacente, equivale a alterar-se somente uma coluna da matriz base, isto significa que o precondicionamento feito em uma base também servirá, parcialmente, para a base seguinte. Pode-se esperar, em casos práticos, que o precondicionamento sirva para várias bases. O precondicionamento deteriora-se com a mudança de bases, cedendo lugar a mais iterações de gradientes conjugados.

Pode acontecer de passar-se de uma base com uma dada dimensão para outra base com dimensão maior, ou seja, poderia passar-se de um ponto com, digamos, 5 restrições ativas, para um ponto com 8 restrições ativas. Neste caso, não podemos utilizar o mesmo precondicionamento anterior que era, no nosso exemplo, uma matriz 5x5 para a nova base, que necessita, obviamente, de uma matriz de precondicionamento de dimensão 8x8. Ainda, neste caso adverso, podemos conseguir uma saída bastante satisfatória, ou seja, completar a dimensão da matriz anterior com vetores da base canônica.



É plenamente justificável essa saída, pois, espera-se que algumas das restrições ativas da base anterior (a menor) continuem ativas na nova base (a maior).

3.4. A DESCRIÇÃO DO SISTEMA BUGRE

O sistema consta de 34 sub-rotinas, selecionadas a seguir:

BUGRE : É a rotina principal, ela obtém os índices das restrições ativas e canalizações ativas. Verifica se o ponto inicial, e os demais pontos obtidos, são factíveis. Controla as iterações de um modo geral.

CONSIS: Faz o teste de consistência para alguns dados de entrada. Faz também um teste de consistência sobre a função e gradiente, fornecidos pelo usuário através da sub-rotina FUN. Este teste é feito verificando se as componentes do gradiente estão próximas do gradiente discreto obtido pela função. No caso de discrepância, uma mensagem sai pelo terminal do computador.

TRANS : Gerencia a transformação das variáveis x (problema original) nas variáveis y (problema canalizado). Gerencia também o uso do precondicionamento.

MRGESP: Resolve o problema canalizado localmente nas variáveis y . Utiliza um gradientes conjugados canalizados.

REFOR : Faz uma perturbação nas restrições de trabalho, diante de uma dege-

PERTB : Faz uma perturbação no vetor termo independente na presença de uma degeneração técnica. Esta sub-sotina é chamada no máximo 4 vezes, após este número o programa pára.

SELIND: Faz a seleção de índices das restrições ativas.

FAZPRE: Decide se faz ou não o precondicionamento.

ESCAL : Faz. opcionalmente, uma normalização (escalamento) nas restrições próprias do problema.

GRADFI: Calcula o gradiente da função objetivo nas variáveis transformadas y.

DIREC : Calcula as direções de busca (direções transformadas) para o processo de busca unidimensional.

NORM : Normaliza as direções.

BUSCA : Faz uma busca unidimensional por interpolação cúbica.

PASMAX: Determina o passo máximo permitido para uma direção de busca.

ACHA : Determina um índice em um conjunto de índices. Faz busca sequencial e é utilizada para controle dos índices das restrições de trabalho. Não é usada para busca de índices "hashing" pelas rotinas GRADFI e DIREC.

AIX : Faz o produto de uma linha da matriz de tecnologia por um vetor.

SELEC : Faz a seleção das colunas da matriz de tecnologia e dos índices de trabalho, obtendo a matriz de trabalho para a fatoração ortogonal.

PRECO1: Faz a decomposição ortogonal por faixa para ser usado como precondicionamento.

CORR2 : Se a rotina PRECO1 produziu um elemento próximo de zero, na diagonal do fator R , da fatoração QR é porque R é quase singular

(ou singular). CORR2 faz a correção destes elementos tornando-os unitários.

RESEL : Faz a resseleção dos índices da base, na presença de uma degeneração técnica (escolha heurística desfavorável). Esta rotina só ficará disponível se a variável lógica RESE for .TRUE., dentro da sub rotina TRANS.

TBLOC : Transfere o bloco da matriz de tecnologia das restrições de trabalho, para a matriz base.

HSATA1: Resolve o sistema linear $A^Tx = b$, ou equivalentemente, $A^TAy = b$ com $x = Ay$ utilizando o precondicionamento da seguinte forma $R^{-T}A^TAR^{-1}z = R^{-T}b$ seguido de $y = R^{-1}z$, sendo AR^{-1} bem condicionada. Utiliza o gradientes conjugados de Hestenes-Stiefel.

HSATA2: Resolve o sistema linear $Ax = b$ ou equivalentemente $AR^{-1}y = b$ onde $x = R^{-1}y$. Sendo AR^{-1} bem condicionada.

ESTR : Faz a estrutura esparsa por linhas da matriz R fator da decomposição ortogonal.

ATXBSP: Calcula $B = A^Tx$, A fornecida com estrutura esparsa por linhas.

AXBESP: O mesmo que ATXBSP para $B = Ax$.

SOLTIE: Resolve o problema linear $R^Tx = b$, com R triangular inferior,com estrutura esparsa por linhas.

SOLTSE: O mesmo que SOLTIE para $Rx = b$.

AXY : Calcula $y = AR^{-1}x$.

ATXY : Calcula $y = R^{-1}A^Tx$.

G1 e G2: Fazem as rotações de Givens.

SAIDA : Faz relatório de saída

DATAE : Faz relatório de dados de entrada.

As sub-rotinas do BUGRE que trabalham com matrizes usam estrutura esparsa por linhas. Este tipo de estrutura é explicado no segundo volume do texto.

3.5. VANTAGENS E DESVANTAGENS DO SISTEMA BUGRE

VANTAGENS:

i) Trabalha com uma forma bastante estável para solução dos sistemas lineares.

ii) Sabe-se, a priori, a quantidade de memória necessária para resolver um dado problema.

iii) A cada iteração a base é "reinvertida" recalculando-se tudo. Dispensa-se o "updating", conseguindo-se deste modo a não propagação de erros de arredondamento/truncamento, ocorridos em processos normais do tipo Simplex com pivotamento.

iv) Espera-se menos iterações, por problema, do que nos processos que andam somente por vértices.

v) Embora o processo não seja do tipo gerador de coluna, como é o Simplex, o BUGRE gasta pouca memória se comparado com o sistema MINOS.

vi) Sua grande quantidade de variáveis de controle pode ser adequada a uma grande gama de problemas.

vii) BUGRE utiliza fatorações ortogonais e, por isso, é mais estável numericamente do que algoritmos que usam fatorações LU. Estas últimas apresentam a vantagem de serem mais rápidas.

DESVANTAGENS:

i) O uso de poucas faixas, para precondicionamento, pode produzir uma matriz R muito "alterada", fazendo com que o número de passos de gradientes conjugados, no sistema linear, aumente muito; tornando o processo lento, menos estável, se provocarmos uma saída prematura na rotina de gradientes conjugados.

ii) Pode ocorrer uma deterioração das direções no espaço das variáveis x , durante a caminhada, com a mesma base, em vários pontos de variedade. Esta deterioração pode gerar infactibilidade.

iii) A escolha de uma base pode falhar. Fazer uma escolha rigorosa seria, computacionalmente, impensável.

iv) Matrizes de tecnologia, ainda que muito esparsas, se apresentarem uma estrutura desfavorável à fatoração ortogonal podem provocar um "fill-in" grande, para um dado número de faixas de trabalho, provocando mais esforço computacional e um visível desgaste na rapidez do processo.

CAPÍTULO IV

TESTES COMPUTACIONAIS

4.1. INTRODUÇÃO

Muitos testes foram realizados com BUGRE. Apresentamos alguns e fazemos uma comparação com os resultados obtidos pelo sistema MINOS para o caso linear. Em todos eles foram utilizadas estruturas tipo escada em suas restrições. Um diagnóstico bastante detalhado é fornecido para cada teste. Os elementos das matrizes de tecnologia foram gerados aleatoriamente de forma que ficassem no intervalo [-20, 20]. Os vetores termos independentes, também gerados aleatoriamente no caso de desigualdades, foram gerados de modo a ter-se um ponto inicial factível, evitando-se a fase um.

Os resultados dos testes mostram uma visível superioridade do sistema BUGRE sobre o sistema MINOS no que concerne à utilização de memória e estabilidade numérica. Na maioria dos casos pode-se esperar menos iterações no BUGRE do que no MINOS, embora o tempo médio por iteração no MINOS seja significativamente menor.

O número de posições de memória usado é dado em palavras de precisão dupla. A tolerância fornecida é para o critério de otimalidade, que é realizado nos componentes do gradiente da função nas variáveis transformadas y. Na verdade esta tolerância serve de base para calcular as demais que independem da máquina utilizada. O número de fatorações ortogonais é igual ao número chamadas a sub-rotina PRECO1 que faz a decomposição ortogonal da matriz base. O número de faixas usadas na fatoração é um valor teto para o número de faixas usadas na fatoração ortogonal. O número de iterações gatilho é o número de iterações a partir do qual é ativado o precondicionamento.

O número de posições de memória gasto pelo MINOS é apresentado de modo aproximado. Procurou-se estabelecer cifras mais realistas possíveis, já que não é possível a priori saber-se, para um dado problema, a quantidade de memória necessária para resolvê-lo utilizando-se o MINOS.

Teve-se o cuidado de produzir exemplos que não produzissem muitas iterações, por uma limitação, óbvia, de tempo e trabalho. Contudo, os testes servirão de base, de estimativa, para rodar problemas futuros; podendo-se ter uma estimativa do tempo por iteração.

Podemos dizer que, se a base é grande, e não muito esparsa, o processo de obtenção de solução de um problema poderá tornar-se quase inviável com relação ao tempo computacional.

Deve ser frisado que o tempo por iteração do BUGRE é bastante grande, já que o programa não faz "updating"; ele calcula tudo a cada nova base. Po

rem, sua estabilidade numérica - no caso escolha correta de bases (não escolhendo bases singulares) - é muito boa.

Durante os testes podemos observar que o processo utilizado pelo MINOS é menos estável numericamente do que o processo usado pelo BUGRE.

Um total de 25 testes computacionais é mostrado, faz-se uma comparação, no caso linear, com resultados obtidos pelo sistema MINOS. Em todos eles utiliza-se uma estrutura particular, tipo blocos em escada, que, sem dúvida, favorece a decomposição ortogonal por faixas e consequentemente favorece o bom andamento do BUGRE.

Através de uma observação detalhada dos resultados, chega-se às seguintes conclusões:

i) BUGRE tem mais estabilidade numérica que MINOS para problemas do tipo mostrado;

ii) MINOS é muito mais rápido, por iteração, do que o BUGRE;

iii) MINOS gasta muito mais memória que o BUGRE;

iv) Em se tratando de blocos diferentes entre si, pode-se dizer que o MINOS exige muito mais memória, especialmente se a base for grande.

Temos, a seguir, uma coleção de 25 testes, divididos da seguinte maneira:

i) Oito problemas com função objetivo linear;

ii) Oito problemas com função objetiva quadrática;

iii) Oito problemas com função objetivo tipo entropia;

iv) Um problema especial.

Cada um dos oito foi dividido em quatro problemas com blocos iguais entre si, e quatro problemas com blocos diferentes entre si. E cada um dos quatro aborda dois problemas com restrições tipo $Ax = b$, dois problemas com restrições tipo $Ax \leq b$, abordando-se para um caso, dimensão de aproximadamente 100 e, para outro caso, uma dimensão de aproximadamente 1000.

O último teste é um problema linear, com 6002 variáveis e 4000 restrições com 100 restrições de igualdade. Não conseguimos rodar este problema com o MINOS devido à insufuciência de espaço de memória, normalmente disponível para um usuário.

RELAÇÃO DOS PROBLEMAS TESTES E

RESULTADOS OBTIDOS

PROBLEMA Nº 1

FUNÇÃO OBJETIVO: linear, $f = -\Sigma x_i$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 100
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 84
TIPO DE RESTRIÇÕES: $Ax \leq b$, $b > 0$
TAMANHO DOS BLOCOS DE A: 5×8, blocos iguais entre si
NÚMERO DE BLOCOS: 20
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 800
DENSIDADE DA MATRIZ A: 9,52%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 40
CANALIZAÇÕES: $0 \leq x \leq 2$
PONTO INICIAL: $x^0 = \underline{0}$

RESULTADOS OBTIDOS PELO MINOS

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 130
TEMPO DE CPU (segundos): 10.42
MEMÓRIA USADA EM PALAVRAS REAL*8: 4000

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 24
NÚMERO DE FATORAÇÕES ORTOGONAIS: 1
NÚMERO DE BASES USADAS: 25
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 25
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 2 (3)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 13
NÚMERO DE ITERAÇÕES GATILHO: 5
TEMPO DE CPU (segundos): 3.02
MEMÓRIA USADA EM PALAVRAS REAL*8: 3575
TOLERÂNCIA USADA: 10^{-4}

PROBLEMA Nº 1-D

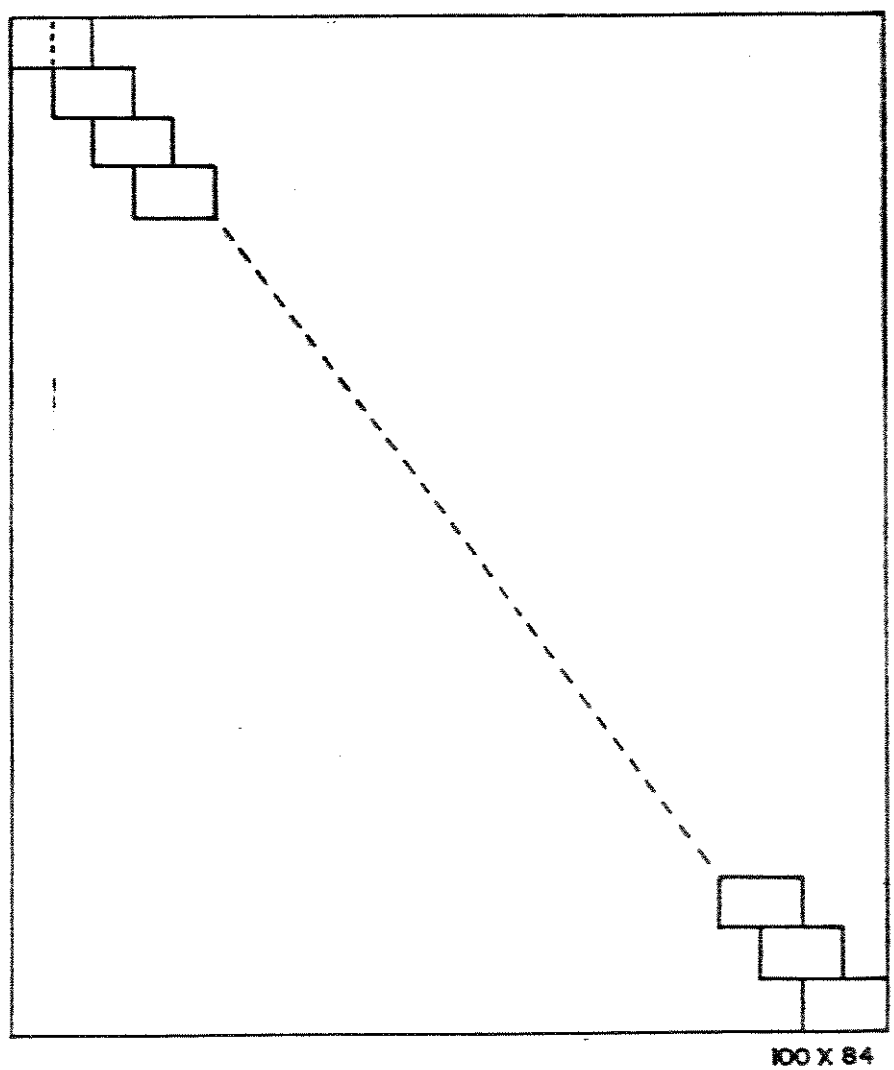
FUNÇÃO OBJETIVO: linear, $f = -\Sigma x_i$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 100
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 84
TIPO DE RESTRIÇÕES: $Ax \leq b$, $b < 0$
TAMANHO DOS BLOCOS DE A: 5×8, blocos diferentes entre si
NÚMERO DE BLOCOS: 20
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 800
DENSIDADE DA MATRIZ A: 9.52%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 800
CANALIZAÇÕES: $0 \leq x \leq 2$
PONTO INICIAL: $x^0 = \underline{0}$

RESULTADOS OBTIDOS PELO MINOS

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 158
TEMPO DE CPU (segundos): 11.5
MEMÓRIA USADA EM PALAVRAS REAL*8: 5000

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 84
NÚMERO DE FATORAÇÕES ORTOGONAIS: 22
NÚMERO DE BASES USADAS: 85
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 85
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 3 (5)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 13
NÚMERO DE ITERAÇÕES GATILHO: 5
TEMPO DE CPU (segundos): 15.7
MEMÓRIA USADA EM PALAVRAS REAL*8: 3775
TOLERÂNCIA USADA: 10^{-4}



PROBLEMA Nº 1

PROBLEMA Nº 2

FUNÇÃO OBJETIVO: linear, $f = -\Sigma x^i$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 1200
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 1202
TIPO DE RESTRIÇÕES: $Ax \leq b$, $b > 0$
TAMANHO DOS BLOCOS DE A: 3×5, blocos iguais entre si
NÚMERO DE BLOCOS: 400
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 6000
DENSIDADE DA MATRIZ A: 0.42%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 15
CANALIZAÇÕES: $0 \leq x \leq 50$
PONTO INICIAL: $x^0 = \underline{0}$

RESULTADOS OBTIDOS PELO MINOS

DIAGNÓSTICO: Parada por mal-condicionamento na rotina CHUZR numa iteração entre 1050 e 1100
NÚMERO DE ITERAÇÕES:
TEMPO DE CPU (segundos):
MEMÓRIA USADA EM PALAVRAS REAL*8: 150.000

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 450
NÚMERO DE FATORAÇÕES ORTOGONAIS: 59
NÚMERO DE BASES USADAS: 451
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 451
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 155 (173)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 8
NÚMERO DE ITERAÇÕES GATILHO: 4
TEMPO DE CPU (segundos): 3148.89
MEMÓRIA USADA EM PALAVRAS REAL*8: 38802
TOLERÂNCIA USADA: 10^{-4}

PROBLEMA Nº 2-D

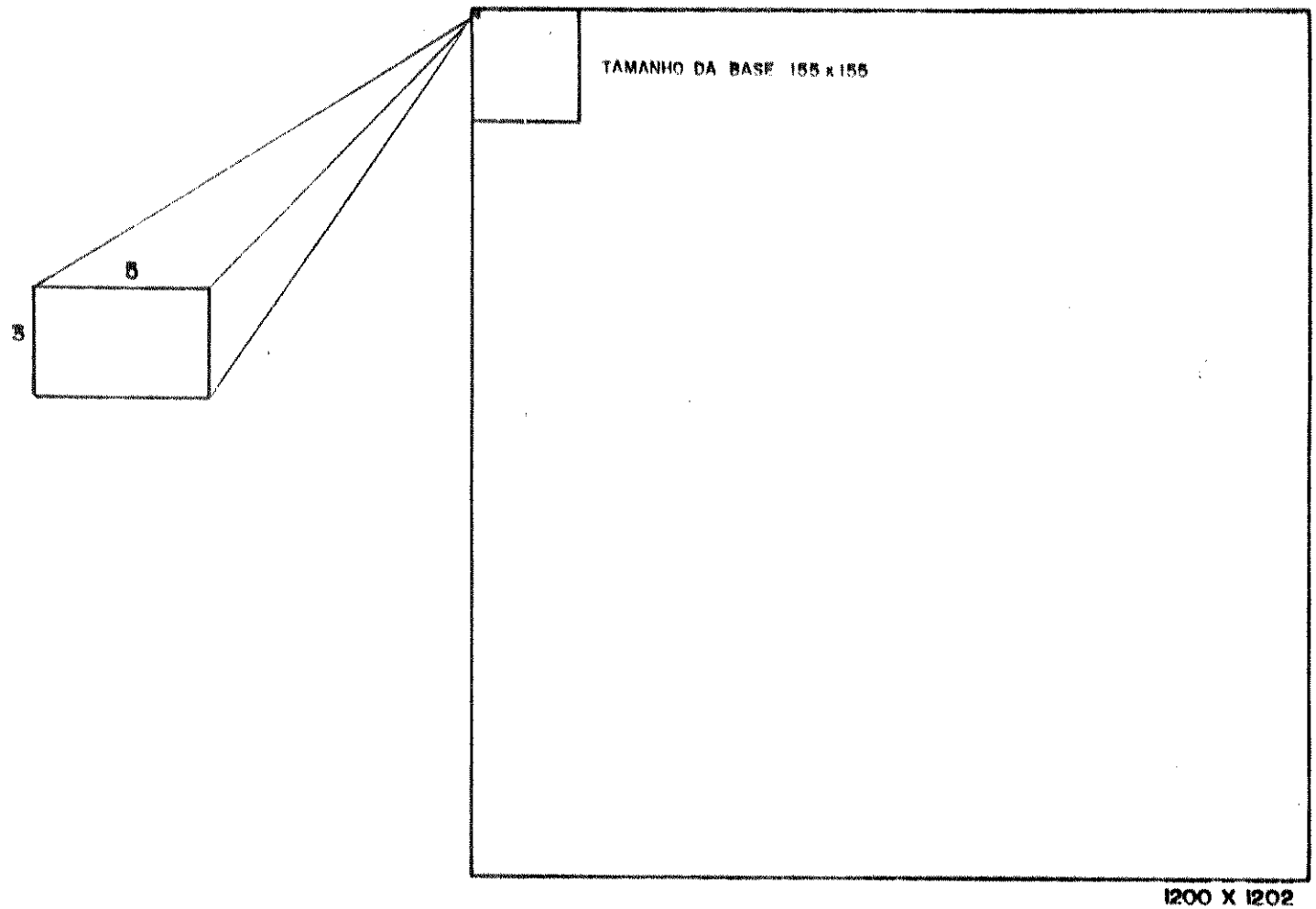
FUNÇÃO OBJETIVO: linear $f = -\Sigma x_i$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 1200
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 1202
TIPO DE RESTRIÇÕES: $Ax \leq b$, $b > 0$
TAMANHO DOS BLOCOS DE A: 3 x 5, blocos diferentes entre si
NÚMERO DE BLOCOS: 400
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 6000
DENSIDADE DA MATRIZ A: 0.42%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 6000
CANALIZAÇÕES: $0 \leq x \leq 50$
PONTO INICIAL: $x^0 = \underline{0}$

RESULTADOS OBTIDOS PELO MINOS

DIAGNÓSTICO: Parou por mal-condicionalmento na rotina CHUZR numa iteração entre 1 e 50
NÚMERO DE ITERAÇÕES:
TEMPO DE CPU (segundos): --
MEMÓRIA USADA EM PALAVRAS REAL*8: 100000

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 282
NÚMERO DE FATORAÇÕES ORTOGONAIS: 94
NÚMERO DE BASES USADAS: 283
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 283
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 152 (175)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 8
NÚMERO DE ITERAÇÕES GATILHO: 4
TEMPO DE CPU (segundos): 2573.49
MEMÓRIA USADA EM PALAVRAS REAL*8: 38802
TOLERÂNCIA USADA: 10^{-4}



PROBLEMA Nº 2

PROBLEMA Nº 3

FUNÇÃO OBJETIVO: linear $f = -\Sigma x_i$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 100
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 100
NÚMERO DE VARIÁVEIS PRÓPRIAS: 142
TIPO DE RESTRIÇÕES: $Ax = b$, $b = 0$
TAMANHO DOS BLOCOS DE A: 5×9, blocos iguais entre si
NÚMERO DE BLOCOS: 20
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 900
DENSIDADE DA MATRIZ A: 9%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 45
CANALIZAÇÕES: $0 \leq x \leq 5$
PONTO INICIAL: $x^0 = \underline{0}$

RESULTADOS OBTIDOS PELO MINOS

DIAGNÓSTICO: Saída normal
NÚMERO DE ITERAÇÕES: 46
TEMPO DE CPU (segundos): 8.37
MEMÓRIA USADA EM PALAVRAS REAL*8: 6400

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 15
NÚMERO DE FATORAÇÕES ORTOGONAIS: 4
NÚMERO DE BASES USADAS: 16
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 16
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 100 (100)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 14
NÚMERO DE ITERAÇÕES GATILHO: 3
TEMPO DE CPU (segundos): 33.64
MEMÓRIA USADA EM PALAVRAS REAL*8: 6365
TOLERÂNCIA USADA: 10^{-4}

PROBLEMA Nº 3-D

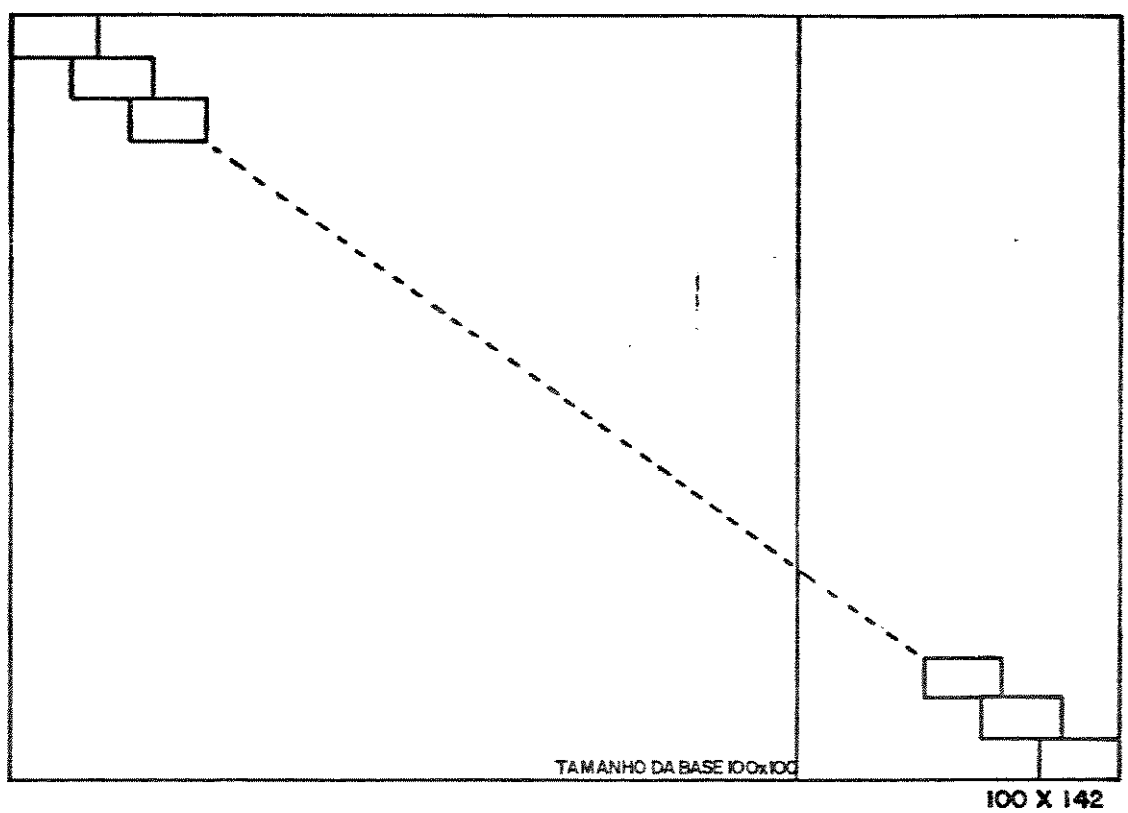
FUNÇÃO OBJETIVO: linear $f = -\Sigma x_i$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 100
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 100
NÚMERO DE VARIÁVEIS PRÓPRIAS: 142
TIPO DE RESTRIÇÕES: $Ax = b$, $b = 0$
TAMANHO DOS BLOCOS DE A: 5 x 9, blocos diferentes entre si
NÚMERO DE BLOCOS: 20
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 900
DENSIDADE DA MATRIZ A: 9%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 900
CANALIZAÇÕES: $0 \leq x \leq 5$
PONTO INICIAL: $x^0 = \underline{0}$

RESULTADOS OBTIDOS PELO MINOS

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 108
TEMPO DE CPU (segundos): 12.04
MEMÓRIA USADA EM PALAVRAS REAL*8: 45000

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 75
NÚMERO DE FATORAÇÕES ORTOGONAIS: 20
NÚMERO DE BASES USADAS: 76
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 76
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 100 (100)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 14
NÚMERO DE ITERAÇÕES GATILHO: 5
TEMPO DE CPU (segundos): 189.73
MEMÓRIA USADA EM PALAVRAS REAL*8: 6365
TOLERÂNCIA USADA: 10^{-4}



PROBLEMA Nº 3

PROBLEMA Nº 4

FUNÇÃO OBJETIVO: linear, $f = -x_1$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 1000
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 1000
NÚMERO DE VARIÁVEIS PRÓPRIAS: 1801
TIPO DE RESTRIÇÕES: $Ax = b$, $b = 0$
TAMANHO DOS BLOCOS DE A: 5×10, blocos iguais entre si
NÚMERO DE BLOCOS: 200
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 10000
DENSIDADE DA MATRIZ A: 0,55%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 50
CANALIZAÇÕES: $-100 \leq x \leq 100$
PONTO INICIAL: $x^0 = \underline{0}$

RESULTADOS OBTIDOS PELO MINOS

DIAGNÓSTICO: Parada por mal-condicionamento na rotina CHUZR
NÚMERO DE ITERAÇÕES: numa iteração entre 195 e 240
TEMPO DE CPU (segundos):
MEMÓRIA USADA EM PALAVRAS REAL*8: 100000

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 55
NÚMERO DE FATORAÇÕES ORTOGONAIS: 2
NÚMERO DE BASES USADAS: 56
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 56
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 1000 (1000)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 10
TEMPO DE CPU (segundos): 3314.86
MEMÓRIA USADA EM PALAVRAS REAL*8: 74348
TOLERÂNCIA USADA: 10^{-5}

PROBLEMA Nº 4-D

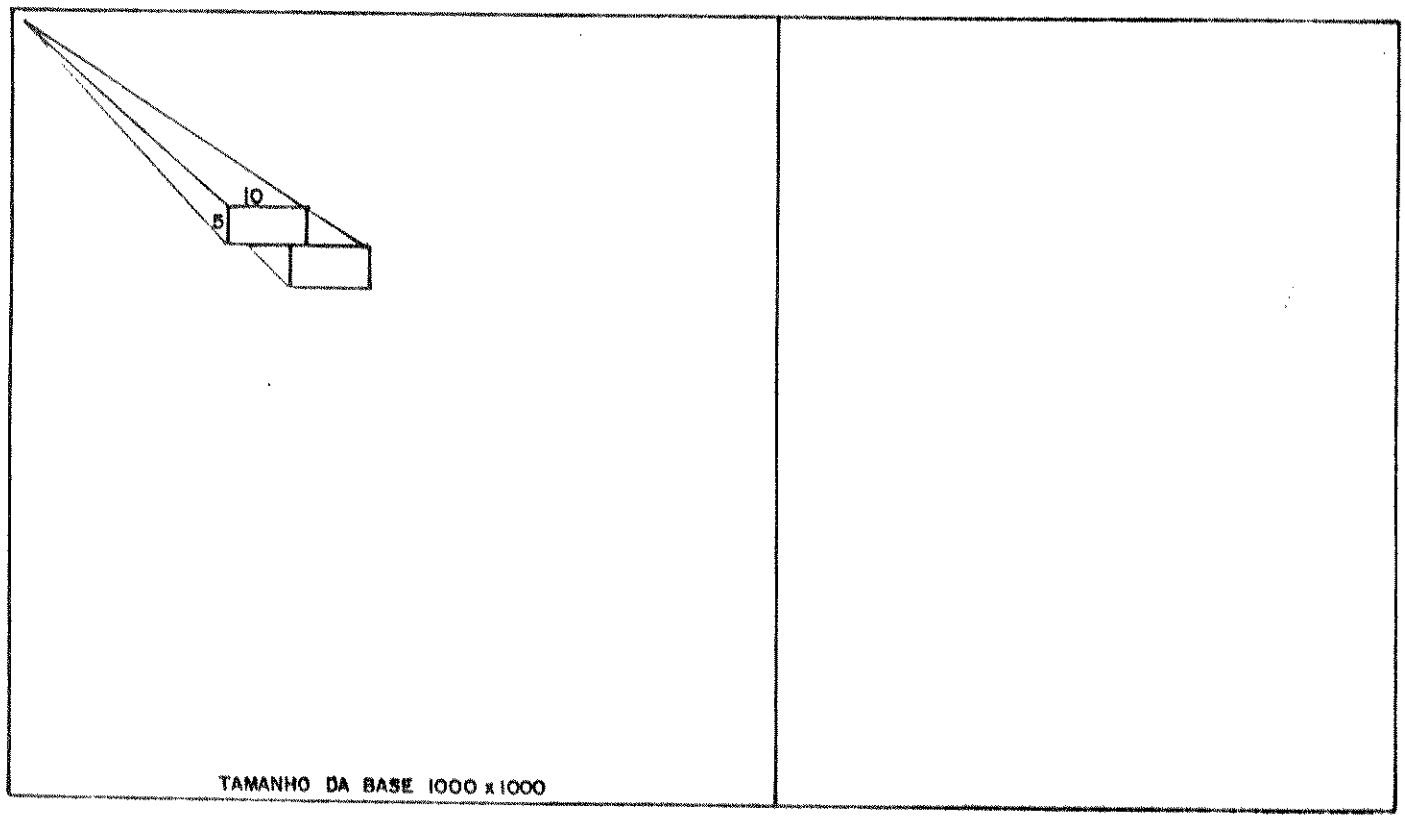
FUNÇÃO OBJETIVO: linear, $f = -x_1$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 1000
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 1000
NÚMERO DE VARIÁVEIS PRÓPRIAS: 1801
TIPO DE RESTRIÇÕES: $Ax = b$, $b = 0$
TAMANHO DOS BLOCOS DE A: 5×10, blocos diferentes entre si
NÚMERO DE BLOCOS: 200
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 10000
DENSIDADE DA MATRIZ A: 0,55%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 10000
CANALIZAÇÕES: $-100 \leq x \leq 100$
PONTO INICIAL: $x^0 = \underline{0}$

RESULTADOS OBTIDOS PELO MINOS

DIAGNÓSTICO: Parou numa iteração entre 575 e 625 por falta de memória
NÚMERO DE ITERAÇÕES:
TEMPO DE CPU (segundos):
MEMÓRIA USADA EM PALAVRAS REAL*8: Atribuiu-se 250000

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 7
NÚMERO DE FATORAÇÕES ORTOGONAIS: 1
NÚMERO DE BASES USADAS: 8
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 8
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 1000, 1000
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 10
TEMPO DE CPU (segundos): 404.95
MEMÓRIA USADA EM PALAVRAS REAL*8: 74348
TOLERÂNCIA USADA: 10^{-5}



PROBLEMA Nº 4

PROBLEMA Nº 5

FUNÇÃO OBJETIVO: não linear, $f = 1/2 \Sigma x_i^2$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 150
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 242
TIPO DE RESTRIÇÕES: $Ax \leq b$
TAMANHO DOS BLOCOS DE A: 5 x 10, blocos iguais entre si
NÚMERO DE BLOCOS: 30
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 1500
DENSIDADE DA MATRIZ A: 4,1 3%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 1500
CANALIZAÇÕES: $1 \leq x \leq 10$
PONTO INICIAL: $x^0 = 5$

RESULTADOS OBTIDOS PELO BUGRE

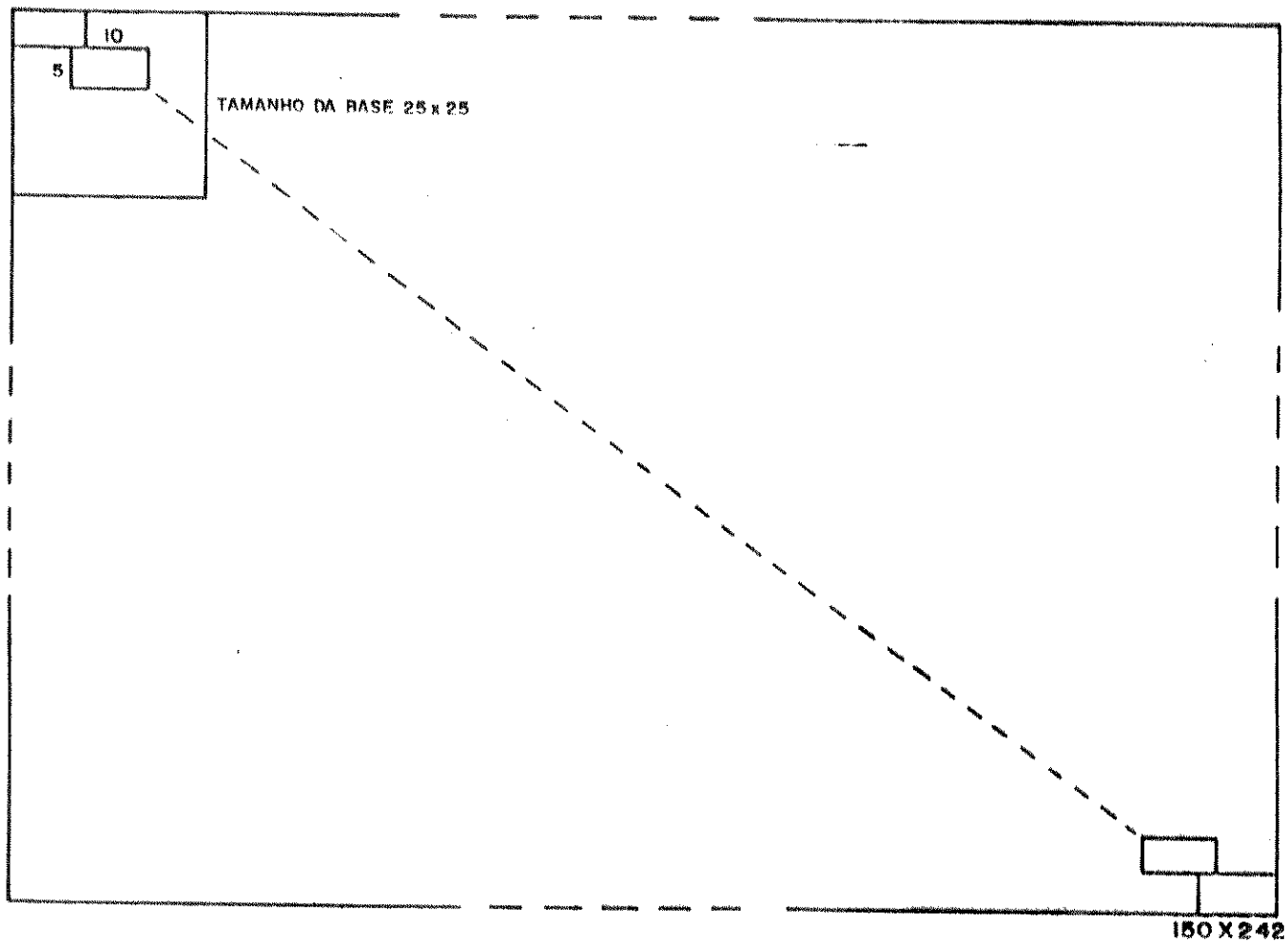
DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 61
NÚMERO DE FATORAÇÕES ORTOGONAIS: 3
NÚMERO DE BASES USADAS: 42
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 82
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 24 (31)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 10
TEMPO DE CPU (segundos): 29.43
MEMÓRIA USADA EM PALAVRAS REAL*8: 8446
TOLERÂNCIA USADA: 10^{-4}

PROBLEMA Nº 5-D

FUNÇÃO OBJETIVO: não linear, $f = 1/2 \Sigma x_i^2$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 150
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 242
TIPO DE RESTRIÇÕES: $Ax \leq b$
TAMANHO DOS BLOCOS DE A: 5 x 10, blocos diferentes entre si
NÚMERO DE BLOCOS: 30
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 1500
DENSIDADE DA MATRIZ A: 4,13%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 1500
CANALIZAÇÕES: $1 \leq x \leq 10$
PONTO INICIAL: $x^0 = 5$

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 144
NÚMERO DE FATORAÇÕES ORTOGONAIS: 9
NÚMERO DE BASES USADAS: 121
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 168
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 25 (31)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 10
TEMPO DE CPU (segundos): 65.02
MEMÓRIA USADA EM PALAVRAS REAL*8: 8446
TOLERÂNCIA USADA: 10^{-4}



PROBLEMA Nº 5

PROBLEMA Nº 6

FUNÇÃO OBJETIVO: não linear, $f = 1/2 \sum x_i^2$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 1050
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 1052
TIPO DE RESTRIÇÕES: $Ax \leq b$
TAMANHO DOS BLOCOS DE A: 3 x 5, blocos diferentes entre si
NÚMERO DE BLOCOS: 350
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 5250
DENSIDADE DA MATRIZ A: 0,47%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 5250
CANALIZAÇÕES: $1 \leq x \leq 10$
PONTO INICIAL: $x^0 = \underline{5}$

RESULTADOS OBTIDOS PELO BUGRE

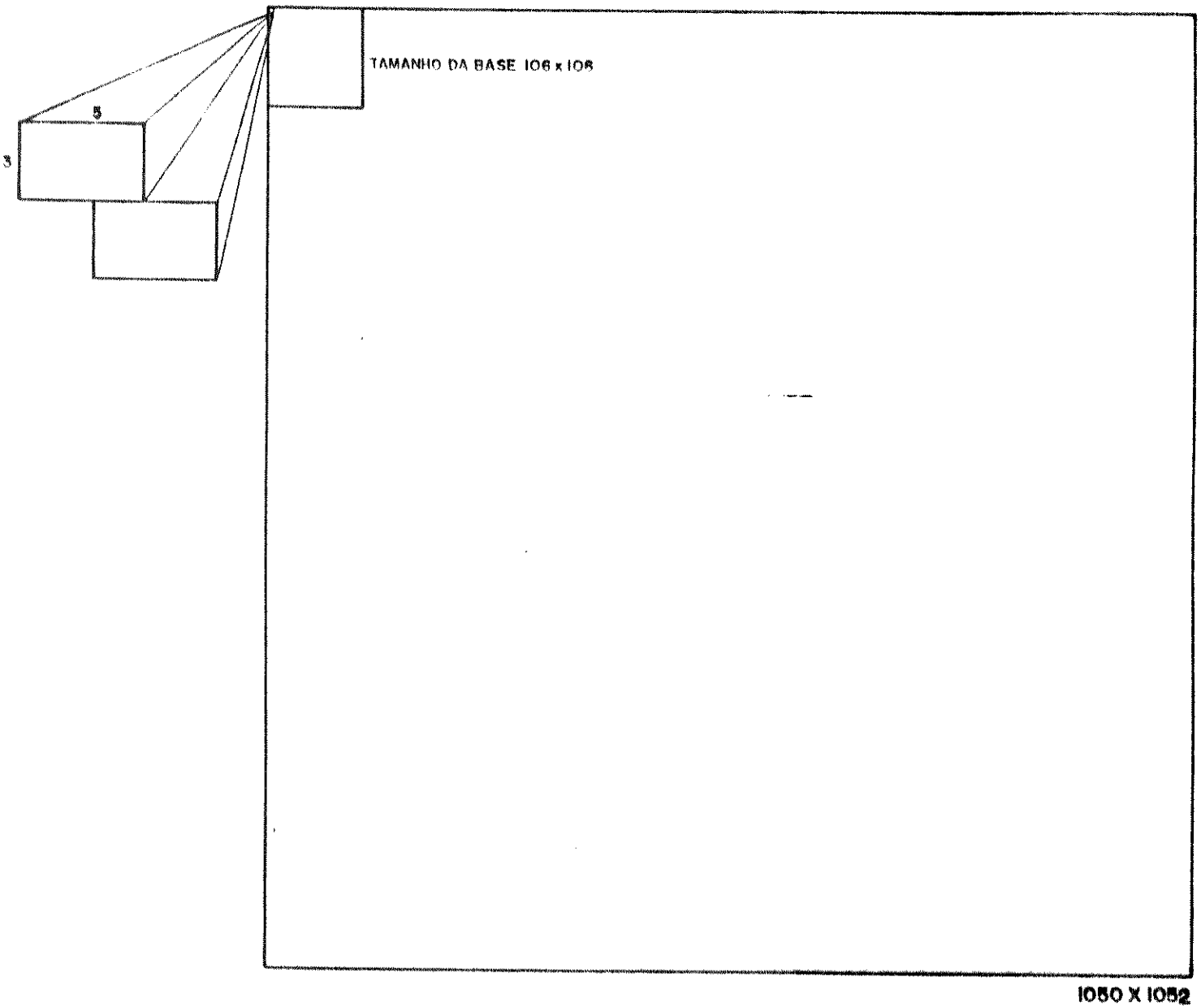
DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 290
NÚMERO DE FATORAÇÕES ORTOGONAIS: 45
NÚMERO DE BASES USADAS: 266
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 332
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 106 (125)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 10
NÚMERO DE ITERAÇÕES GATILHO: 5
TEMPO DE CPU (segundos): 1391.93
MEMÓRIA USADA EM PALAVRAS REAL*8: 33893
TOLERÂNCIA USADA: 10^{-4}

PROBLEMA Nº 6-D

FUNÇÃO OBJETIVO: não linear, $f = 1/2 \Sigma x_i^2$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 1050
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 1052
TIPO DE RESTRIÇÕES: $Ax \leq b$
TAMANHO DOS BLOCOS DE A: 3 x 5, blocos iguais entre si
NÚMERO DE BLOCOS: 350
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 5250
DENSIDADE DA MATRIZ A: 0,47%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 15
CANALIZAÇÕES: $1 \leq x \leq 10$
PONTO INICIAL: $x^0 = \underline{5}$

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 78
NÚMERO DE FATORAÇÕES ORTOGONAIS: 6
NÚMERO DE BASES USADAS: 65
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 95
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 103 (112)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 10
NÚMERO DE ITERAÇÕES GATILHO: 5
TEMPO DE CPU (segundos): 305.13
MEMÓRIA USADA EM PALAVRAS REAL*8: 33893
TOLERÂNCIA USADA: 10^{-4}



PROBLEMA Nº 6

PROBLEMA Nº 7

FUNÇÃO OBJETIVO: não linear, $f = 1/2 \Sigma x_i^2$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 100
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 100
NÚMERO DE VARIÁVEIS PRÓPRIAS: 162
TIPO DE RESTRIÇÕES: $Ax = b$
TAMANHO DOS BLOCOS DE A: 5 x 10, blocos iguais entre si
NÚMERO DE BLOCOS: 20
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 1000
DENSIDADE DA MATRIZ A: 6,17%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 1000
CANALIZAÇÕES: $-100 \leq x \leq 100$
PONTO INICIAL: $x^0 = \underline{5}$

RESULTADOS OBTIDOS PELO BUGRE

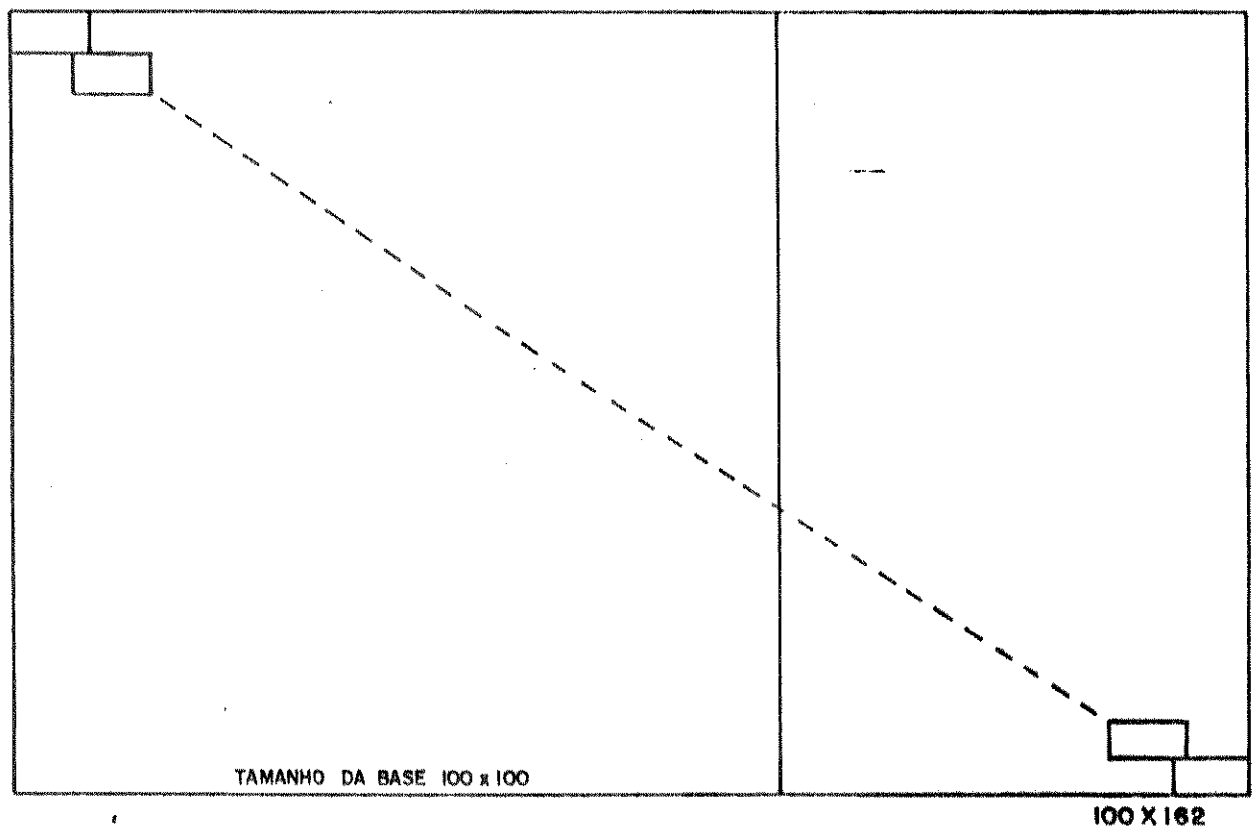
DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 125
NÚMERO DE FATORAÇÕES ORTOGONAIS: 2
NÚMERO DE BASES USADAS: 15
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 156
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 100 (100)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 10
TEMPO DE CPU (segundos): 60.46
MEMÓRIA USADA EM PALAVRAS REAL*8: 7030
TOLERÂNCIA USADA: 10^{-4}

PROBLEMA Nº 7-D

FUNÇÃO OBJETIVO: não linear, $f = 1/2 \sum x_i^2$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 100
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 100
NÚMERO DE VARIÁVEIS PRÓPRIAS: 162
TIPO DE RESTRIÇÕES: $Ax = b$
TAMANHO DOS BLOCOS DE A: 5×10, blocos diferentes entre si
NÚMERO DE BLOCOS: 20
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 1000
DENSIDADE DA MATRIZ A: 6,17%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 1000
CANALIZAÇÕES: $-100 \leq x \leq 100$
PONTO INICIAL: $x^0 = \underline{5}$

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 58
NÚMERO DE FATORAÇÕES ORTOGONAIS: 1
NÚMERO DE BASES USADAS: 1
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 134
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 100, 100
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 10
TEMPO DE CPU (segundos): 26.41
MEMÓRIA USADA EM PALAVRAS REAL*8: 7030
TOLERÂNCIA USADA: 10^{-4}



PROBLEMA Nº 7

PROBLEMA Nº 8

FUNÇÃO OBJETIVO: não linear, $f = 1/2 \Sigma x_i^2$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 800
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 800
NÚMERO DE VARIÁVEIS PRÓPRIAS: 1204
TIPO DE RESTRIÇÕES: $Ax = b$
TAMANHO DOS BLOCOS DE A: 4 x 10, blocos iguais entre si
NÚMERO DE BLOCOS: 200
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 8000
DENSIDADE DA MATRIZ A: 0,83%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 40
CANALIZAÇÕES: $3 \leq x \leq 5$
PONTO INICIAL: $x^0 = \underline{4}$

RESULTADOS OBTIDOS PELO BUGRE

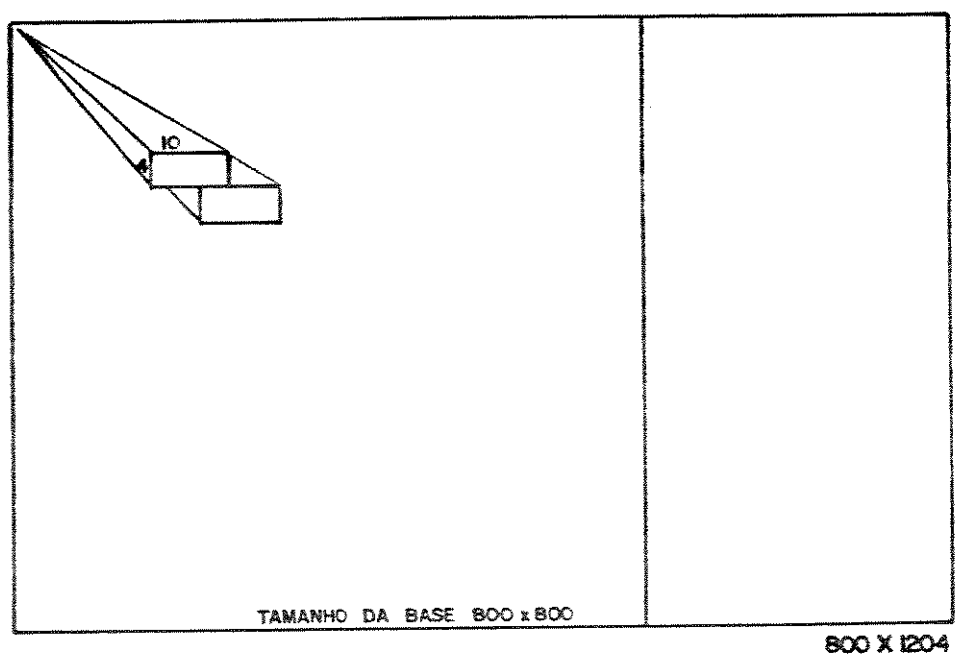
DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 74
NÚMERO DE FATORAÇÕES ORTOGONAIS: 15
NÚMERO DE BASES USADAS: 28
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 125
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 800 (800)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 10
NÚMERO DE ITERAÇÕES GATILHO: 4
TEMPO DE CPU (segundos): 2030.34
MEMÓRIA USADA EM PALAVRAS REAL*8: 51613
TOLERÂNCIA USADA: 10^{-4}

PROBLEMA Nº 8-D

FUNÇÃO OBJETIVO: não linear, $f = 1/2 \sum x_i^2$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 800
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 800
NÚMERO DE VARIÁVEIS PRÓPRIAS: 1204
TIPO DE RESTRIÇÕES: $Ax = b$
TAMANHO DOS BLOCOS DE A: 4×10, blocos diferentes entre si
NÚMERO DE BLOCOS: 200
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 8000
DENSIDADE DA MATRIZ A: 0,83%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 8000
CANALIZAÇÕES: $3 \leq x \leq 5$
PONTO INICIAL: $x^0 = \underline{4}$

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 378
NÚMERO DE FATORAÇÕES ORTOGONAIS: 64
NÚMERO DE BASES USADAS: 52
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 713
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 800 (800)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 10
NÚMERO DE ITERAÇÕES GATILHO: 4
TEMPO DE CPU (segundos): 11715.0
MEMÓRIA USADA EM PALAVRAS REAL*8: 51613
TOLERÂNCIA USADA: 10^{-4}



PROBLEMA Nº 8

PROBLEMA Nº 9

FUNÇÃO OBJETIVO: não linear, $f = \Sigma\, x^i \log(x^i)$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 100
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 162
TIPO DE RESTRIÇÕES: $Ax \leq b$
TAMANHO DOS BLOCOS DE A: 5 x 10, blocos iguais entre si
NÚMERO DE BLOCOS: 20
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 1000
DENSIDADE DA MATRIZ A: 6,17%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 50
CANALIZAÇÕES: $2 \leq x \leq 10$
PONTO INICIAL: $x^0 = \underline{7}$

RESULTADOS OBTIDOS PELO BUGRE

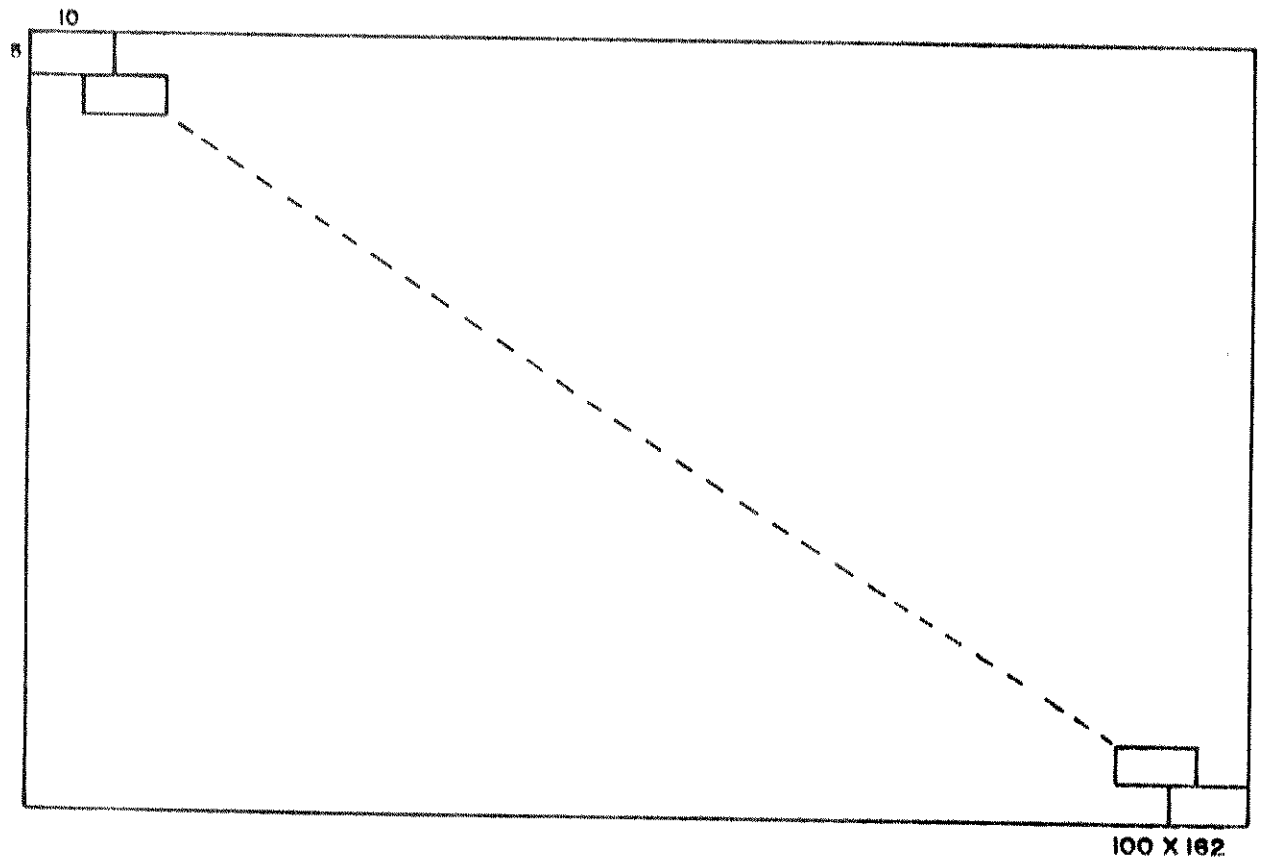
DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 112
NÚMERO DE FATORAÇÕES ORTOGONAIS: 1
NÚMERO DE BASES USADAS: 106
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 123
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 7 (9)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 4
TEMPO DE CPU (segundos): 21.10
MEMÓRIA USADA EM PALAVRAS REAL*8: 5413
TOLERÂNCIA USADA: 10^{-4}

PROBLEMA Nº 9-D

FUNÇÃO OBJETIVO: não linear, $f = \Sigma\, x^i \log(x^i)$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 100
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 162
TIPO DE RESTRIÇÕES: $Ax \leq b$
TAMANHO DOS BLOCOS DE A: 5 x 10, blocos diferentes entre si
NÚMERO DE BLOCOS: 20
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 1000
DENSIDADE DA MATRIZ A: 6,17%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 50
CANALIZAÇÕES: $2 \leq x \leq 10$
PONTO INICIAL: $x^0 = \underline{7}$

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 84
NÚMERO DE FATORAÇÕES ORTOGONAIS: 26
NÚMERO DE BASES USADAS: 65
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 99
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 10 (12)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 4
TEMPO DE CPU (segundos): 35.23
MEMÓRIA USADA EM PALAVRAS REAL*8: 5413
TOLERÂNCIA USADA: 10^{-4}



PROBLEMA Nº 9

PROBLEMA Nº 10

FUNÇÃO OBJETIVO: não linear, $f = \Sigma\, x^i \log x^i$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 1000
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 1602
TIPO DE RESTRIÇÕES: $Ax \leq b$
TAMANHO DOS BLOCOS DE A: 5 x 10, blocos iguais entre si
NÚMERO DE BLOCOS: 200
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 10000
DENSIDADE DA MATRIZ A: 0,62%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 50
CANALIZAÇÕES: $2 \leq x \leq 6$
PONTO INICIAL: $x^0 = \underline{6}$

RESULTADOS OBTIDOS PELO BUGRE

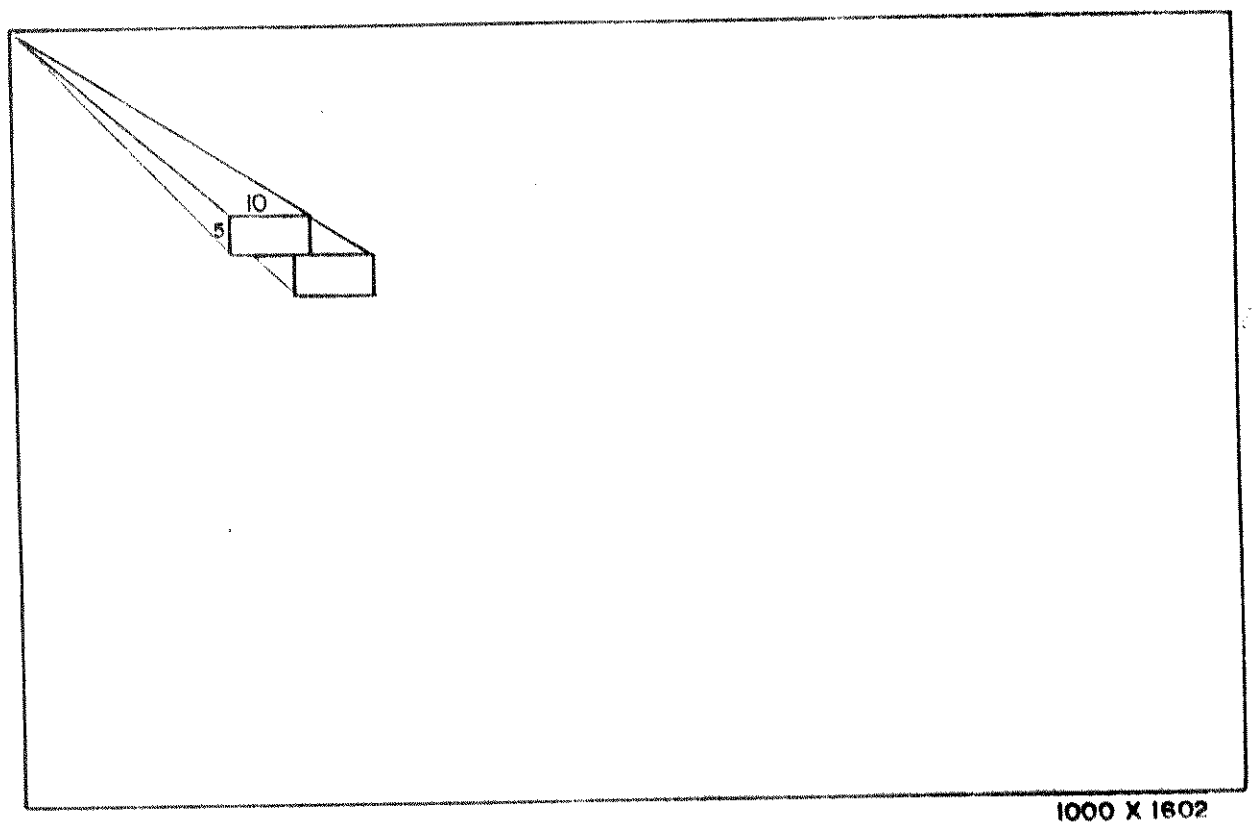
DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 425
NÚMERO DE FATORAÇÕES ORTOGONAIS: 23
NÚMERO DE BASES USADAS: 426
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 427
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 33 (40)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 4
TEMPO DE CPU (segundos): 1285.36
MEMÓRIA USADA EM PALAVRAS REAL*8: 53785
TOLERÂNCIA USADA: 10^{-4}

PROBLEMA Nº 10-D

FUNÇÃO OBJETIVO: não linear, $f = \Sigma\, x^i \log x^i$
MODO DE GERAÇÃO: aleatório para os elementos de A e de b
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 1000
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 0
NÚMERO DE VARIÁVEIS PRÓPRIAS: 1602
TIPO DE RESTRIÇÕES: $Ax \leq b$
TAMANHO DOS BLOCOS DE A: 5 x 10, blocos diferentes entre si
NÚMERO DE BLOCOS: 200
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 10000
DENSIDADE DA MATRIZ A: 0,62%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 50
CANALIZAÇÕES: $2 \leq x \leq 6$
PONTO INICIAL: $x^0 = \underline{6}$

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 963
NÚMERO DE FATORAÇÕES ORTOGONAIS: 92
NÚMERO DE BASES USADAS: 964
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 965
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 42 (47)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 4
TEMPO DE CPU (segundos): 3217.45
MEMÓRIA USADA EM PALAVRAS REAL*8: 53785
TOLERÂNCIA USADA: 10^{-4}



TAMANHO DA BASE 33 x 33

PROBLEMA Nº 10

PROBLEMA Nº 11

FUNÇÃO OBJETIVO: não linear, $f(x) = \Sigma\, x^i \log(x^i)$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 100
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 100
NÚMERO DE VARIÁVEIS PRÓPRIAS: 124
TIPO DE RESTRIÇÕES: Ax = b
TAMANHO DOS BLOCOS DE A: 5 x 10, blocos iguais entre si
NÚMERO DE BLOCOS: 20
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 1000
DENSIDADE DA MATRIZ A: 8,06%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 50
CANALIZAÇÕES: $5 \leq x \leq 10$
PONTO INICIAL: $x^0 = \underline{8}$

RESULTADOS OBTIDOS PELO BUGRE

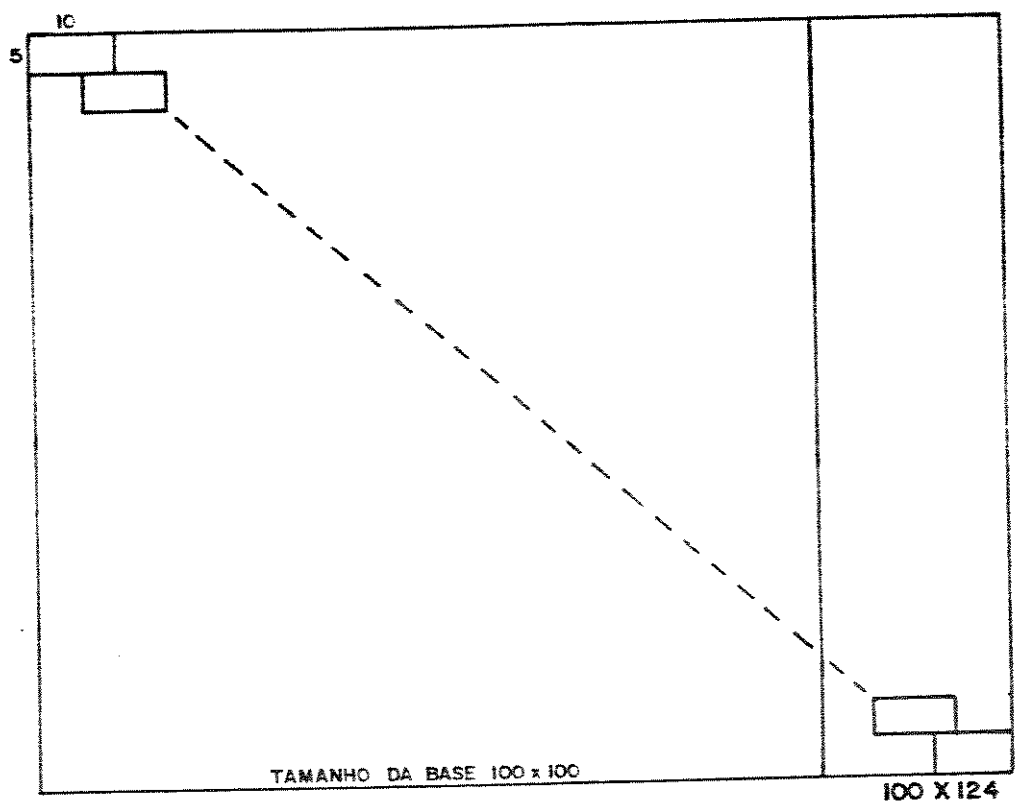
DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 67
NÚMERO DE FATORAÇÕES ORTOGONAIS: 3
NÚMERO DE BASES USADAS: 41
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 115
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 100 (100)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 5
TEMPO DE CPU (segundos): 51.21
MEMÓRIA USADA EM PALAVRAS REAL*8: 6403
TOLERÂNCIA USADA: 10^{-4}

PROBLEMA Nº 11-D

FUNÇÃO OBJETIVO: não linear, $f = \Sigma\, x^i \log x^i$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 100
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 100
NÚMERO DE VARIÁVEIS PRÓPRIAS: 124
TIPO DE RESTRIÇÕES: $Ax = b$
TAMANHO DOS BLOCOS DE A: 5×10, blocos diferentes entre si
NÚMERO DE BLOCOS: 20
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 1000
DENSIDADE DA MATRIZ A: 8,06%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 1000
CANALIZAÇÕES: $5 \leq x \leq 10$
PONTO INICIAL: $x^0 = \underline{8}$

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 39
NÚMERO DE FATORAÇÕES ORTOGONAIS: 7
NÚMERO DE BASES USADAS: 28
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 73
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 100 (100)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 5
TEMPO DE CPU (segundos): 35.93
MEMÓRIA USADA EM PALAVRAS REAL*8: 6403
TOLERÂNCIA USADA: 10^{-4}



PROBLEMA Nº 11

PROBLEMA Nº 12

FUNÇÃO OBJETIVO: não linear, $f = \Sigma\, x_i \log x_i$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 900
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 900
NÚMERO DE VARIÁVEIS PRÓPRIAS: 905
TIPO DE RESTRIÇÕES: $Ax = b$
TAMANHO DOS BLOCOS DE A: 5×10, blocos iguais entre si
NÚMERO DE BLOCOS: 180
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 9000
DENSIDADE DA MATRIZ A: 1,1%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 50
CANALIZAÇÕES: $5 \leq x \leq 10$
PONTO INICIAL: $x^0 = \underline{8}$

RESULTADOS OBTIDOS PELO BUGRE

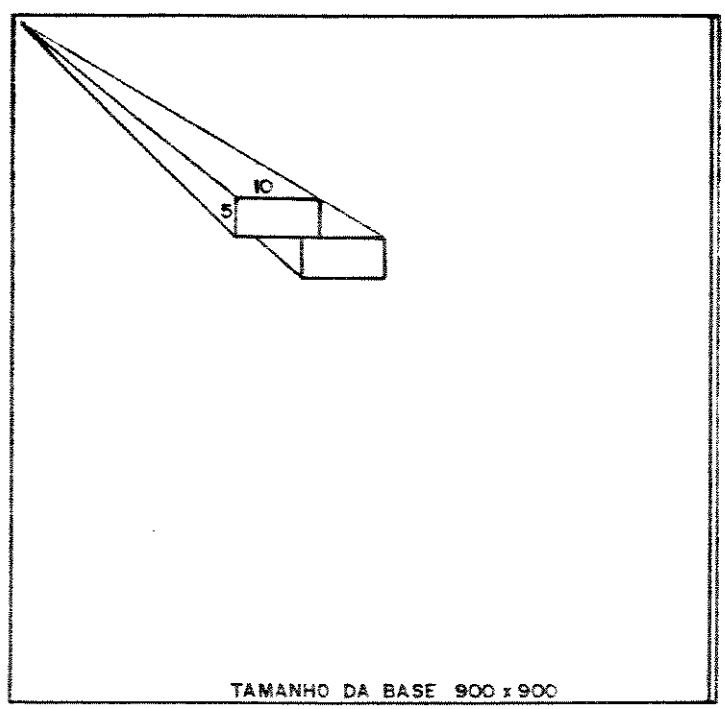
DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 3
NÚMERO DE FATORAÇÕES ORTOGONAIS: 1
NÚMERO DE BASES USADAS: 3
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 5
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 900 (900)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 5
TEMPO DE CPU (segundos): 1475.9
MEMÓRIA USADA EM PALAVRAS REAL*8: 55089
TOLERÂNCIA USADA: 10^{-4}

PROBLEMA Nº 12-D

FUNÇÃO OBJETIVO: não linear, $f = \Sigma\, x^i \log x^i$
MODO DE GERAÇÃO: aleatório para os elementos de A
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 900
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 900
NÚMERO DE VARIÁVEIS PRÓPRIAS: 905
TIPO DE RESTRIÇÕES: $Ax = b$
TAMANHO DOS BLOCOS DE A: 5×10, blocos diferentes entre si
NÚMERO DE BLOCOS: 180
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 9000
DENSIDADE DA MATRIZ A: 1,1%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 9000
CANALIZAÇÕES: $5 \leq x \leq 10$
PONTO INICIAL: $x^0 = \underline{8}$

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 67
NÚMERO DE FATORAÇÕES ORTOGONAIS: 5
NÚMERO DE BASES USADAS: 13
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 85
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 900 (900)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 15
NÚMERO DE ITERAÇÕES GATILHO: 5
TEMPO DE CPU (segundos): 32.710.5
MEMÓRIA USADA EM PALAVRAS REAL*8: 55089
TOLERÂNCIA USADA: 10^{-4}



PROBLEMA Nº 12

PROBLEMA Nº 13

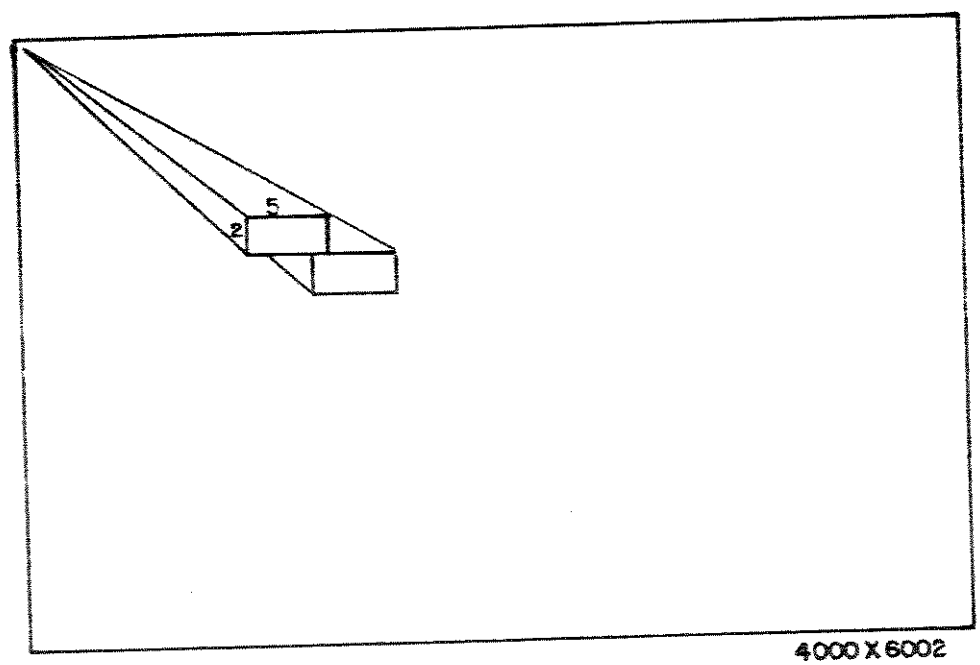
FUNÇÃO OBJETIVO: linear, $f = -x_1$
MODO DE GERAÇÃO: aleatório
NÚMERO DE RESTRIÇÕES PRÓPRIAS: 4000
NÚMERO DE RESTRIÇÕES DE IGUALDADE: 100
NÚMERO DE VARIÁVEIS PRÓPRIAS: 6002
TIPO DE RESTRIÇÕES: $Ax \leq b$
TAMANHO DOS BLOCOS DE A: 2 x 5
NÚMERO DE BLOCOS: 2000, blocos diferentes entre si
NÚMERO DE ELEMENTOS DIFERENTES DE ZERO EM A: 20000
DENSIDADE DA MATRIZ A: 0,0083%
NÚMERO DE ELEMENTOS DIFERENTES ENTRE SI: 20000
CANALIZAÇÕES: $0 \leq x \leq 10$
PONTO INICIAL: $x^0 = \underline{0}$

RESULTADOS OBTIDOS PELO MINOS

DIAGNÓSTICO: Não roda por falta de memória disponível ao usuário
NÚMERO DE ITERAÇÕES:
TEMPO DE CPU (segundos):
MEMÓRIA USADA EM PALAVRAS REAL*8:

RESULTADOS OBTIDOS PELO BUGRE

DIAGNÓSTICO: saída normal
NÚMERO DE ITERAÇÕES: 8
NÚMERO DE FATORAÇÕES ORTOGONAIS: 3
NÚMERO DE BASES USADAS: 9
NÚMERO DE AVALIAÇÕES DE FUNÇÃO E GRADIENTE: 9
TAMANHO MÉDIO (MÁXIMO) DAS BASES: 100 (100)
NÚMERO DE FAIXAS USADAS NA FATORAÇÃO: 6
NÚMERO DE ITERAÇÕES GATILHO: 5
TEMPO DE CPU (segundos): 239.2
MEMÓRIA USADA EM PALAVRAS REAL*8: 156.914
TOLERÂNCIA USADA: 10^{-4}



PROBLEMA Nº 13

4.2. COMENTÁRIOS SOBRE CADA TESTE

Rotulamos os problemas testes numerando-os 1, 1-D; 2, 2-D,... e assim por diante. Os testes que foram rotulados com a letra "D" foram produzidos de modo que seus blocos de restrições eram diferentes em si. Os demais apresentam blocos iguais entre si.

Para uma perfeita noção do tamanho de cada problema, elaboramos desenhos da matriz de restrições de cada teste, mostrando a sua dimensão e o tamanho de cada bloco. Em alguns casos mostramos uma vista explodida do bloco para fornecer uma idéia mais precisa do tamanho do problema.

A seguir enumeramos os referidos testes e um comentário sobre cada um, detendo-se na comparação dos resultados fornecidos pelo sistema MINOS (para o caso de problemas lineares).

PROBLEMA Nº 1

Este pequeno teste mostra, de imediato, uma superioridade do programa BUGRE sobre o MINOS. Observa-se, de imediato, um número de iteracões muito menor e também um tempo de CPU menor.

Neste exemplo tivemos, aproximadamente, os mesmos requisitos de memória para ambos.

Este exemplo mostra que a caminhada por dentro do convexo pode ser, em alguns casos, uma economia de tempo computacional.

Ainda, no referido teste, deve ser observado que foi realizado somente uma fatoração ortogonal pelo BUGRE; fato este que, sem dúvida, diminui bastante o tempo computacional.

PROBLEMA Nº 1-D

Este problema, de pequenas dimensões, mostra uma superioridade do sistema MINOS sobre o BUGRE, no que concerne a tempo gasto por iteração. O MINOS realizou 158 iterações em 11,5 segundos e o BUGRE realizou somente 84 em 15,7 segundos. Porém o BUGRE realizou a tarefa com menos memória que o MINOS.

Em problemas maiores a problemática de memória torna-se mais evidente.

PROBLEMA Nº [illegible]

[illegible] já [illegible] que o problema do Nº [illegible]. Apesar de que o MINOS [illegible] condicionado, de modo que o referido sistema não consegue resolver e apresenta problemas de estouro de aritmética de ponto flutuante ("overflow").

[illegible] o MINOS chegou a realizar mais de 1000 iterações sem atingir a solução e parando por mal-condicionamento. Note-se ainda que o BUGRE requereu menos que 40.000 palavras de memória, enquanto o MINOS, para este mesmo problema, requereu 150.000 palavras.

Outra observação interessante é que o número de fatorações ortogonais usadas pelo BUGRE foi somente 59 no total de 450 iterações. Mostra-se com isso, que o precondicionamento periódico é uma excelente técnica para ganhar-se tempo.

Com um número tão grande de iterações, tivemos como era de se esperar, um tempo razoavelmente grande de CPU.

PROBLEMA Nº 2-D

O sistema MINOS mostrou, neste teste, uma maior vulnerabilidade ao mal-condicionamento, e parou, muito cedo, numa iteração entre 1 e 50. Isto, possivelmente, é devido à existência de blocos diferentes nas restrições, causando um enchimento ("fill-in") grande e fatal para o MINOS.

Em problemas com estruturas particularmente boas, o enchimento é mais ou menos limitado, e, um sistema como o MINOS, pode fazer muitas inversões matriciais com segurança. Porém se o enchimento torna-se grande, espera-se um mal comportamento das inversões, pois o número de elementos é um fator de desgaste do condicionamento do problema, principalmente quando esses elementos são diferentes entre si.

Também, neste teste, a quantidade de memória requerida pelo BUGRE foi substancialmente menor do que a requerida pelo MINOS.

PROBLEMA Nº 3

No presente teste, o MINOS resolve o problema em um tempo bem menor do que o BUGRE. As necessidades de memória são aproximadamente as mesmas para os dois sistemas. Este teste mostra que quando os blocos são iguais entre si, pode-se esperar um bom desempenho do sistema MINOS, já que espera-se menos enchimento nas fatorações. Podemos dizer que problemas que apresentam muitos elementos iguais e/ou muitos elementos 1's e -1's favorecem enormemente o bom desempenho do MINOS.

PROBLEMA Nº 3-D

Enfatizando o comentário sobre o problema 3, este teste mostra que o aparecimento de elementos diferentes (blocos diferentes) nos problemas pode comprometer a disponibilidade de memória para se resolver um problema grande pelo MINOS.

O par de problemas 3 e 3-D mostra como a natureza dos blocos interfere no andamento e necessidades do sistema MINOS.

O BUGRE resolveu o problema em um tempo muito grande em comparação ao MINOS, possivelmente devido a um número muito grande de fatorações ortogonais em relação ao número de iterações.

PROBLEMA Nº 4

Este problema apresenta uma base muito grande 1000 x 1000 (todas as restrições são de igualdade) e têm blocos iguais entre si. O sistema MINOS usa variáveis de folga (e/ou artificiais para bases não evidentes) tornando os problemas um tanto maior e também mais complicados. No presente caso o MINOS parou por mal-condicionamento numa iteração entre 195 e 240, mostrando que nem sempre problemas com blocos iguais de estrutura particularmente boa serão resolvidos por este sistema.

Observando o desenho dos blocos no problema 4, nota-se a quase ortogonalidade dos blocos, fator este que favorece o desempenho do BUGRE. Este último realizou somente duas fatorações ortogonais num total de 55 iterações. O BUGRE resolveu o problema com menos necessidade de memória que o MINOS e em um tempo razoável já que o tamanho da base é grande e possui 10.000 elementos diferentes de zero.

...este teste apresenta alguns resultados bastante [illegible] comparados ao problema Nº 6. Neste problema, apesar de ter sido resolvido com blocos diferentes, o BUGRE só necessitou de uma fatoração ortogonal, realizando um total de 3 iterações. Ao MINOS, neste teste, atribuiu-se bem mais memória que o anterior, e ainda assim, não resolveu o problema. Chegou, este último, a realizar mais de 500 iterações e parou por falta de memória.

Mais uma vez fica evidente a utilização do BUGRE em certos tipos de problema em que o MINOS mostra-se inútil.

PROBLEMAS Nº 5 e Nº 5-D

Estes problemas que apresentam funções objetivas não-lineares também foram resolvidos com êxito pelo BUGRE. Os resultados podem servir para uma possível previsão do tempo computacional e comportamento do BUGRE para problemas com funcões objetivos quadráticas, como é o caso destes testes.

Nestes testes observa-se um aumento no tempo de CPU em relação ao problema linear já que o custo computacional da funcão objetivo se faz valer.

Observando o problema Nº 5 nota-se que foram realizadas 82 avaliações de função e gradiente num total de 61 iterações e 42 bases. Isto significa que tivemos vários passos de gradientes conjugados numa dada face, ou seja, ocorreram passos do algoritmo do tipo minimização de função sobre uma variedade.

Nos dois problemas o número de fatorações ortogonais foi bastante pequeno, conseguindo assim um bom desempenho do algoritmo.

PROBLEMAS Nº 6 e Nº 6-D

Estes testes já apresentam um tamanho considerável e gastam um tempo também considerável para sua soluçãc. Neste caso, menos comum, o problema com blocos diferentes foi resolvido mais eficientemente do que o com blocos iguais. A relacão entre o número de iteracões e o número de fatorações ortogonais foi melhor no problema com blocos diferentes.

Comparando-se os problemas de número 6 com os problemas de números 5 podemos dizer que o crescimento do tamanho do problema acarreta um crescimento

PROBLEMAS Nº 7 e Nº 7-D

Como nos problemas Nº 6 e Nº 6-D, estes também têm função objetivo quadrática. Estes problemas que apresentam somente restrições de igualdade possuem, relativamente, poucos graus de liberdade (162 variáveis e 100 restrições de igualdade). Espera-se em problemas deste tipo, idas a poucas bases e muitas avaliações de função e gradiente, ou seja, há muitos sub-problemas de minimização em uma variedade (que em última análise é um problema de minimização sem restrições), com poucos problemas de mudança de base.

No problema Nº 7 tivemos 15 bases usadas com 2 fatorações ortogonais contra 156 avaliações de função e gradiente e no problema Nº 7-D foram usadas uma única base contra 134 avaliações de função e gradiente.

PROBLEMAS Nº 8 e Nº 8-D

Estes problemas apresentam uma base grande 800 x 800 e, como anteriormente, têm funções objetivas quadráticas. Deve ser observado que o número de iterações no problema 8-D é bastante alto, e também o é o número de avaliações de função e gradiente acarretando um grande aumento do tempo de CPU.

PROBLEMAS Nº 9 a Nº 12

Do mesmo modo que realizamos testes com função objetivo quadrática, também realizamos testes com função objetivo do tipo entropia; essas função são do tipo $\Sigma\, x_i \log x_i$. O cálculo da função logaritmica presente na função torna muito maior o tempo de CPU se compararmos com os testes quadráticos.

Esses testes foram realizados para mostrar a eficiência do BUGRE mesmo em problemas bastante não-lineares. Além disto esta coleção de testes poderá servir para se fazer estimativas sobre o tempo de CPU para um dado problema com uma dada função objetivo, já que, com algum esforço pode-se fazer uma correlação entre tempo de CPU, número de iterações e tamanho da base.

PROBLEMA Nº 13

Sabe-se que o MINOS guarda várias cópias dos elementos da matriz para atualizações, número de cópias estimado em aproximadamente 5 vezes o número de elementos diferentes de zero da matriz. Isto faz com que a necessidade de memória total para o MINOS seja bastante grande, já que este trabalha com variáveis de folga. No presente teste, não conseguimos a nível de usuário a quantidade de memória necessária para rodar o problema no VAX-785. Tentamos até 350.000 palavras de precisão dupla e ainda assim o sistema MINOS pedia mais memória; ao atribuir-se valor mais elevado para a dimensão do vetor de trabalho do MINOS atingia-se o teto de memória disponível ao usuário.

O BUGRE resolveu este problema utilizando apenas 157.000 palavras em precisão dupla e com 3 fatorações ortogonais e 8 iterações. Este exemplo constitui-se numa forte evidência de que o BUGRE é uma alternativa para solução de problemas em que o MINOS falha.

CONCLUSÃO GERAL

CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

Dentre diversas opções possíveis para continuidade do trabalho podemos enumerar algumas mais imediatas:

i) Melhorar o precondicionamento.

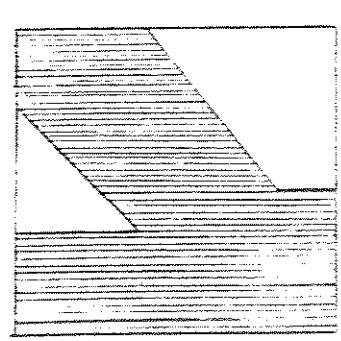
É possível fazer uma melhora significativa no precondicionamento dos sistemas lineares utilizando-se rotações rápidas de Givens (FGR) que não calculam raizes quadradas. É necessário um estudo mais aprofundado sobre o tratamento de fatorações incompletas; poder-se-ia conseguir um precondicionamento mais seguro e melhor.

ii) Incorporar uma fase I Inteligente.

No programa que elaboramos não há fase I. O usuário terá que fazer duas chamadas à rotina principal se quiser resolver um problema em que não dispõe de uma solução inicial factível. Esta desvantagem faz o usuário gastar mais tempo para resolver seu problema.
Outro aspecto importante é que uma fase I inteligente poderia ser usada para tornar o programa mais robusto, já que a ida a um ponto infactivel poderia ser tratada de modo automático pelo programa.

iii) Testar diferentes estruturas e adaptar o precondicionamento para essas estruturas.

Esta opção além de ser bem viável é de grande importância já que certas classes de problemas de grande porte apresentam uma estrutura particular que pode (e deve) ser explorada. A adaptacão do precondicionamento seria de grande valor teórico e prático. Para exemplificar suponha que a matriz do sistema tenha a seguinte forma:



Neste caso, uma fatoração LQ poderia ser usada para precondicionamento.

iv) Pré-processamento do problema para que fique mais adequado para o precondicionamento.

Certos problemas podem apresentar uma estrutura ruim para o precondicionamento, fazendo-se uma reestruturacão do problema poder-se-ia melhorar a sua estrutura. Isto poderia ser feito internamente no programa para um certo tipo de precondicionamento, por exemplo através de permutação de linhas e de colunas poderia conseguir-se uma forma matricial tipo faixa ou quase isso. Ou ainda, o mais triangular superior possível.

v) Adaptar para trabalhar com restrições não-lineares.

Muitos programas que trabalham com restrições lineares podem ser adaptados para trabalhar com restrições não-lineares. A tarefa de uma adaptação, embora difícil, pode ser de grande valia já que um programa robusto que trabalha com restrições lineares poderá ter um comportamento excelente quando adaptado para resolver problemas com restrições não-lineares.

APÊNDICE 1

Algoritmo para minimizar uma função quadrática e positiva definida sujeita a restrições canalizadas.

Considere o problema abaixo:

$$\text{Minimizar} \quad f(x) = 1/2\, x^T Ax - h^T x$$

$$\text{sujeito a} \quad \ell \leq x \leq u$$

com A (nxn) positiva definida.

Um algoritmo para resolver o problema acima pode ter a seguinte forma:

PASSO 1 : Escolha um ponto inicial x_1, $\ell \leq x_1 \leq u$ e calcule $r_1 = h - Ax_1$.

PASSO 2 . Seja I o conjunto de índices i tais que

$x_1^i = \ell^i$ e $r_1^i \leq 0$

ou

$x_1^i = u^i$ e $r_1^i \geq 0$

Se $r_1^i = 0$ para todos os índices não pertencentes a I, então x_1 é o mínimo de f sobre as canalizações e o processo termina.

PASSO 3 : Faça $p_1 = \bar{r}_1$ onde

$$\bar{r}_1^i = \begin{cases} 0, & \text{para todo } i \in I \\ r_1^i, & \text{caso contrário} \end{cases}$$

PASSO 4 : (Gradientes Conjugados). Começando com $k = 1$, calcule

Passo 4.1: $s_k = Ap_k$, $c_k = p_k^T r_k$, $d_k = p_k^T s_k$, $a_k = \dfrac{c_k}{d_k}$

$x_{k+1} = x_k + a_k p_k$, $r_{k+1} = r_k - a_k s_k$

Se x_{k+1} está fora dos limites das canalizações, vá para o passo 5. Caso contrário, se $r_{k+1}^i = 0$ para todo índice i não pertencente a I, recoloque $x_1 = x_{k+1}$, $r_1 = r_{k+1} = h - Ax_1$ e vá para o passo 2. Senão calcule $\bar{r}_{k+1}$ e p_{k+1} como segue:

$$\bar{r}_{k+1} = \begin{cases} 0 & , \quad \text{se } i \in I \\ r^i_{k+1} & , \quad \text{caso contrário} \end{cases}$$

$$p_{k+1} = \bar{r}_{k+1} + b_k p_k, \quad b_k = -\frac{s_k^T \bar{r}_{k+1}}{d_k} = \frac{|\bar{r}_{k+1}|^2}{c_k}$$

atribua $k := k+1$ e vá para o passo 4.1.

PASSO 5 : Sejam $i_1,\ldots,i_s$ os índices i tais que $x^i_{k+1} < \ell^i$ ou $x^i_{k+1} > u^i$. Seja $\bar{\alpha}_k$ dado por

$\bar{\alpha}_k$ = Mínimo $\{\alpha_\ell, \alpha_u\}$

onde

$$\alpha_\ell = \text{Mínimo} \left\{ \frac{\ell^i_k - x^i_k}{p^i_k}, \quad p^i_k > 0, \quad i = i_1,\ldots,i_s \right\}$$

$$\alpha_u = \text{Mínimo} \left\{ \frac{u^i_k - x^i_k}{p^i_k}, \quad p^i_k < 0, \quad i = i_1,\ldots,i_s \right\}$$

Atribua $x_1 := x_k + \bar{\alpha}_k p_k$, $r_1 := r_k - \bar{\alpha}_k s_k = h - Ax_1$.

Redefina I como sendo o conjunto dos índices i, tais que $x^i_1 = \ell^i$ ou $x^i_1 = u^i$. Se $r^i_1 = 0$ para todos os índices i não pertencentes a I vá para o passo 2. Senão vá para o passo 3.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] O. AXELSSON, G. LINDSKOG, *On the Eigenvalue Distribution of a Class of Preconditioning Methods*, Numer. Math. 48 (1983), 479-498.

[2] O. AXELSSON, G. LINDSKOG, *On the Rate of Convergence of the Preconditioned Conjugate Gradient Method*, Numer. Math. 48 (1986), 479-498.

[3] R.H. BARTELS, G.H. GOLUB, M.A. SAUNDERS, *Numerical Techniques in Mathematical Programming* in Nonlinear Programming, J.B. Rosen, O.L. Mangasarian and K. Ritter editors, Academic Press, (1970), 123-176.

[4] M. BERTOCCHI, E. SPEDICATO, *Computational Experience with Conjugate Gradient Algorithms*, Calcolo, vol. 15, 3 (1979), 255-269.

[5] D.P. BERTSEKAS, *On the Goldstein - Levitin - Polyak Gradient Projection Method*, IEEE Trans. Aut. Control, AC-21 (1976), 174-184.

[6] D.P. BERTSEKAS, *Project Newton Methods for Optimization Problems with Simple Constraints*, SIAM J. Control and Optimization, vol. 20, 2 (1982), 221-246.

[7] Å. BJORCK, *Methods for Sparse Linear Least Squares Problems* in Sparse Matrix Computations, J.R. Bunch and D.J. Rose editors, Academic Press Inc. (1976), 177-199.

[8] J.J. DONGARRA, G.K. LEAF, M. MINKOFF, *A Preconditioned Conjugate Gradient Method for Solving a Class of Non-Symmetric Linear Systems*, ANL-81-71, Argonne National Laboratory, Argonne, Ill, 1981.

[9] R. FLETCHER, Practical Methods of Optimization, John-Wiley & Sons, Vol. 2, 1981.

[10] R. FLETCHER, C.M. REEVES, *Function Minimization by Conjugate Gradients*, Comput. J. 2 (1964), 149-153.

[11] W.M. GENTLEMAN, *Least Squares Computations by Givens Transformations without Square Roots*, J. Inst. Maths. Applic. 12 (1973), 329-336.

[12] A. GEORGE, M.T. HEATH. *Solution of Sparse Linear Least Squares Using Givens Rotations*, Linear Algebra and its Applics.,34 (1980), 69-83.

[13] P.E. GILL, W. MURRAY, *Numerically Stable Methods for Quadratic Programming*, Mathematical Programming 14 (1978), 349-372.

[14] P.E. GILL, W. MURRAY, *The Orthogonal Factorization of a Large Sparse Matrix* in Sparse Matrix Computations, J.R. Bunch and D.J. Rose editors, Academic Press Inc. (1976), 201-212.

[15] P.E. GILL, W. MURRAY, M. WRIGHT, Practical Optimization, Academic Press, 1981.

[16] J. GOODMAN, *Newton's Method for Constrained Optimization*, Mathematical Programming 14 (1978), 349-372.

[17] I.M. GOULD, *On Practical Conditions for the Existence and Uniqueness of Solutions to the General Equality Quadratic Programming Problem*, Mathematical Programming 32 (1985), 90-99.

[18] M.R. HESTENES, Conjugate Direction Methods in Optimization, Springer-Verlag, 1980.

[19] M.R. HESTENES, E. STIEFEL, *Methods of Conjugate Gradients for Solving Linear Systems*, J. Res. Nat. Bur. Standards 49 (1952), 409-436.

[20] D.G. LUENBERGER, Linear and Nonlinear Programming, (2nd edition), Addison Wesley, 1984.

[21] N. MACULAN, M.V. PEREIRA, Programação Linear, Editora Atlas, 1980.

[22] J.M. MARTÍNEZ, *Notas sobre Transformações de Householder*, notas de aula, IMECC, UNICAMP, 1981.

[23] J.M. MARTÍNEZ, *A New Algorithm for Sparse Nonlinear Least Squares Problems*, trabalho apresentado no International Congress of Industrial and Applied Mathematics (ICIAM 87), Paris, julho 1987.

[24] B.A. MURTAGH, M.A. SAUNDERS, *Large-Scale Linearly Constrained Optimization*, Mathematical Programming 14 (1978), 41-72.

[25] B.A. MURTAGH, M.A. SAUNDERS, MINOS User's Guide, Technical Report, Department of Oper. Res., Stanford, California, 1977.

[26] D.P. O'LEARY, *A Generalized Conjugate Gradient Algorithm for Solving a Class of Quadratic Programming Problems*, Linear Algebra and its Applics. 34 (1980), 371-399.

[27] B.T. POLYAK, *The Conjugate Gradient Method in Extremal Problems*, USSR Comp. Math. and Math. Physics 9 (1969), 94-112.

[28] G.W. STEWART, *The Economical Storage of Plane Rotations*, Numer. Math. 25 (1978), 137-138.