MÉTODOS ITERATIVOS PARALELOS PARA PROBLEMAS DE NORMA MÍNIMA

JOSÉ MARCOS LOPES

Tese apresentada à Faculdade de Engenharia de Ilha Solteira, da Universidade Estadual Paulista "Julio de Mesquita Filho", como parte dos requisitos para a obtenção do título de Livre-docente na disciplina "Cálculo Numérico".

Ilha Solteira (SP), maio de 2001.

MÉTODOS ITERATIVOS PARALELOS PARA PROBLEMAS DE NORMA MÍNIMA

JOSÉ MARCOS LOPES

Tese apresentada à Faculdade de Engenharia de Ilha Solteira, da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos para a obtenção do título de Livre-docente na disciplina "Cálculo Numérico".

Ilha Solteira(SP), maio de 2001.

AGRADECIMENTOS

A todas as pessoas que direta ou indiretamente colaboraram com a realização desse trabalho, em especial aos colegas do Departamento de Matemática da UNESP/Ilha Solteira, pelo apoio e companheirismo; aos meus alunos, com quem muito aprendi e a Elaine e a Lúcia que digitaram parte desse texto.

À FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) pelo apoio financeiro concedido para o desenvolvimento desse projeto de pesquisa.(Processo FAPESP 95/1008-5)

À minha esposa Beth e às minhas filhas Melina, Milena e Emília, pelo amor e carinho a min dedicados.

Aos meus pais biológicos:
Vicente e Joana.
Aos meus pais adotivos:
tio Nico(in memorian) e tia Geni(in memorian);
dedico.

RESUMO

LOPES, J. M. **Métodos Iterativos Paralelos Para Problemas de Norma Mínima.** Ilha Solteira, 2001, Tese (livre-docência) – Faculdade de Engenharia de Ilha Solteira, Universidade Estadual Paulista "Júlio de Mesquita Filho". 72p.

Apresentamos neste trabalho, dois novos métodos iterativos paralelos para problemas de norma mínima. O primeiro método refere-se ao problema de estimação na norma ℓ_∞. O algoritmo é a versão paralela de um proposto por Dax, é do tipo relaxação por linha e conveniente quando o sistema de equações lineares a ser resolvido é inconsistente de grande porte, esparso e não possui uma estrutura detectável. O segundo é um método iterativo paralelo para a solução do sistema linear inconsistente $\boldsymbol{Ax} = \boldsymbol{b}$, utilizando a norma ℓ_p, para $1 < p < \infty$. O algoritmo proposto é do tipo relaxação por coluna; ou seja; em cada sub-iteração (passo) apenas uma coluna da matriz $\boldsymbol{A}$ é utilizada e como o método é paralelo, todas as colunas (variáveis) podem ser processadas simultaneamente. Métodos de relaxação por coluna (linha) são convenientes quando $\boldsymbol{A}$ é de grande porte, esparsa e não possui uma estrutura detectável.

Palavras-chave: sistemas lineares, matriz esparsa, norma mínima, métodos iterativos paralelos, métodos tipo relaxação coluna(linha).

ABSTRACT

LOPES, J. M. **Iterative Parallel Methods for Minimum Norm Problems.** Ilha Solteira, 2001, Tese (livre-docência) – Faculdade de Engenharia de Ilha Solteira, Universidade Estadual Paulista "Júlio de Mesquita Filho'. 72p.

This work presents two new iterative parallel methods for minimum norm problems. The first method solves the problem in an estimation ℓ_∞ norm. This algorithm, a parallel version of Dax's algorithm, is a row relaxation type. It is convenient when the system to be solved is inconsistent, large, sparse and unstructured. The second one is an iterative parallel method applied to solve inconsistent system, $\boldsymbol{Ax} = \boldsymbol{b}$, using the ℓ_p norm, for $1 < p < \infty$. The proposal algorithm is a column relaxation type, i.e., in each step only one column of matrix $\boldsymbol{A}$ is used and, because it is a parallel method, all columns can be simultaneously processed. Column-relaxation methods are convenient if $\boldsymbol{A}$ is large, sparse, and without any detectable structure.

Key words: linear systems, sparse matrix, minimum norm, parallel iterative methods, column(row)-relaxation methods.

SUMÁRIO

PARTE 1 – Um método iterativo paralelo para problemas minimax.

PARTE 2 – Um método iterativo paralelo para problemas de ajuste l_p.

PARTE 1 – Um método iterativo paralelo para problemas minimax.

1. INTRODUÇÃO

Estamos interessados aqui em obter a solução do seguinte problema minimax

$$\text{minimize} \quad \| A x - b \|_\infty \tag{1.1}$$

onde A é uma matriz real $m \times n$, $b = (b_1, \ldots, b_m)^t \in \Re^m$ e $x = (x_1, \ldots, x_n)^t \in \Re^n$ denota o vetor de incógnitas ; "t" representa transposto.

O problema (1.1) é equivalente a um Problema de Programação Linear. Quando a matriz A é de grande porte, métodos que utilizam modificações de A, como é por exemplo o caso do método Simplex, não são convenientes ou mesmo inaplicáveis para a resolução de (1.1).

Propomos neste trabalho, um método iterativo paralelo do tipo ação por linha, onde as linhas (variáveis) podem ser processadas em paralelo. Os métodos do tipo ação por linha são convenientes quando A é de grande porte, esparsa e sem uma estrutura detectável. Tais métodos tem se mostrado eficientes por exemplo, para resolver grandes sistemas lineares que ocorrem no campo da reconstrução de imagens por projeção, Censor (1981), Censor e Zenios (1993). Neste caso não é necessário o armazenamento da matriz A, as entradas não nulas da i-ésima linha são geradas de dados experimentais em cada iteração.

Consideremos a seguinte regularização para o problema (1.1)

$$\text{minimize} \quad \frac{1}{2} \varepsilon \left(\| x \|_2^2 + \| A x - b \|_\infty^2 \right) + \| A x - b \|_\infty \tag{1.2}$$

onde ε é um número real positivo. O interesse no estudo do problema (1.2) é que através da solução deste problema, podemos obter uma aproximação da solução do problema (1.1).

Segundo Dax (1991), o dual de (1.2) é definido por

$$\text{minimize} \quad \frac{1}{2} \| A^t y \|_2^2 - \varepsilon b^t y + \frac{1}{2} \left(\max\{ 0, \| y \|_1 - 1 \} \right)^2 \tag{1.3}$$

e se $y = (y_1, y_2, \ldots, y_m)^t \in \Re^m$ resolve (1.3) então $x = (A^t y)/\varepsilon$ resolve (1.2).

A idéia da regularização foi introduzida por Tikhnov e outros, como uma forma de melhorar a estabilidade de problemas mal condicionados, (Tikhnov e Arsenin (1977)). No caso aqui considerado, o objetivo da regularização é obter um problema dual mais simples.

Uma regularização natural para o problema (1.1) seria

$$\text{minimize } \frac{1}{2}\,\varepsilon\,\|\boldsymbol{x}\|_2^2 + \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty \tag{1.4}$$

entretanto, (Dax (1991)) o dual de (1.4) tem a forma

$$\begin{aligned} &\text{minimize } \frac{1}{2}\,\|\boldsymbol{A}^t\boldsymbol{y}\|_2^2 - \varepsilon\,\boldsymbol{b}^t\boldsymbol{y} \\ &\text{sujeito a } \|\boldsymbol{y}\|_1 \le 1 \end{aligned} \tag{1.5}$$

as restrições de (1.5) introduzem uma certa dificuldade na implementação de métodos do tipo ação por linha.

Em Lopes e De Pierro (1992), apresentamos um algoritmo paralelo, para estimação de mínimo valor absoluto, onde a regularização utilizada é como em (1.4). Mangassariam (1981), foi o primeiro a utilizar este tipo de regularização, para propor um algoritmo iterativo em Programação Linear.

Da notação utilizada ,

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty = \max_{1\le i\le m} \left| \boldsymbol{a}_i^t\boldsymbol{x} - b_i \right|$$

onde $\boldsymbol{a}_i^t$ denota a i-ésima linha de $\boldsymbol{A}$, e

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \quad \text{para} \quad 1 \le p < \infty \quad \text{e qualquer} \quad \boldsymbol{x} \in \Re^n.$$

Para quaisquer $\boldsymbol{x}, \boldsymbol{y} \in \Re^n$, o produto escalar euclidiano será denotado por $\boldsymbol{x}^t\boldsymbol{y}$ ou $\langle \boldsymbol{x}, \boldsymbol{y} \rangle$.

O artigo está organizado da seguinte forma. Na seção 2 apresentamos uma conveniente regularização para o problema minimax; na seção 3 o novo algoritmo é definido

e provas de convergência são fornecidas. Na seção 4 apresentamos os resultados de algumas experiências computacionais e a seção 5 refere-se a conclusões finais.

2. UMA REGULARIZAÇÃO PARA O PROBLEMA MINIMAX

Consideremos o problema minimax regularizado

$$\text{minimize } \frac{1}{2}\varepsilon\left(\|x\|_2^2 + \|Ax - b\|_2^2\right) + \|Ax - b\|_\infty \tag{2.1}$$

apresentamos a seguir o dual do problema (2.1) e relacionamos as soluções primal-dual. Todos os resultados desta seção são devidos a Dax (1991).

Mangassarian (1981) estudou o Problema de Programação Linear (PPL) regularizado

$$\begin{aligned} &\text{minimize } \frac{1}{2}\varepsilon\|x\|_2^2 + c^t x \\ &\text{sujeito a } \quad Ax \geq b \end{aligned} \tag{2.2}$$

como uma forma de obter uma solução aproximada para o PPL,

$$\begin{aligned} &\text{minimize } c^t x \\ &\text{sujeito a } \quad Ax \geq b \end{aligned} \tag{2.3}$$

onde ε, A, b e x são como definidos em (1.2) e $c \in \Re^n$ é o vetor da função objetivo. Mostrou-se neste caso que se (2.3) tem solução e $\hat{y} \in \Re^m$ resolve o problema

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|A^t y - c\|_2^2 - \varepsilon b^t y \\ &\text{sujeito a } \quad y \geq 0 \end{aligned} \tag{2.4}$$

então o vetor $\hat{x} = \left(A^t\hat{y} - c\right)/\varepsilon$ é a solução única de (2.2).

Agora, o problema (1.1) é equivalente ao seguinte PPL (Sposito (1975))

$$\text{minimize } (1,0,\ \ldots\ ,0)\begin{pmatrix}\tau\\ x\end{pmatrix} \tag{2.5}$$

$$\text{sujeito a} \quad \begin{bmatrix} e & A \\ e & -A \end{bmatrix} \begin{bmatrix} \tau \\ x \end{bmatrix} \geq \begin{bmatrix} b \\ -b \end{bmatrix}$$

onde $e = (1, 1, \ldots, 1)^t \in \Re^m$. Assim, utilizando as idéias de Mangassariam (1981), Dax (1991) mostrou que o dual do problema (2.1) tem a forma

$$\text{minimize } F(y) = \frac{1}{2}\| A^t y \|_2^2 - \varepsilon b^t y + \frac{1}{2}\left(\max\left\{0, \sum_{i=1}^{m} |y_i| - 1\right\}\right)^2 \qquad (2.6)$$

A equivalência entre os problemas (2.1) e (2.6) é estabelecida pelo teorema abaixo.

Teorema 2.1. O problema (2.6) sempre tem uma solução. Se $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_m)^t \in \Re^m$ resolve este problema então o vetor

$$\hat{x} = A^t \hat{y}/\varepsilon \qquad (2.7)$$

é a solução única de (2.1) e

$$\| A\hat{x} - b \|_\infty = \max\left\{ 0, \sum_{i=1}^{m} |\hat{y}_i| - 1 \right\} \Big/ \varepsilon \quad . \qquad (2.8)$$

Observações:

2.1. A existência de solução para o problema (2.6) segue diretamente da convexidade da função objetivo $F(y)$.

2.2. Se o sistema linear $Ax = b$ é inconsistente então de (2.8) qualquer solução $\hat{y}$ de (2.6) satisfaz

$$\| \hat{y} \|_1 > 1 \, .$$

Portanto qualquer ponto de mínimo da função

$$G(y) = \frac{1}{2} \| A^t y \|_2^2 - \varepsilon b^t y + \frac{1}{2}\left(\| y \|_1 - 1 \right)^2$$

é também um ponto de mínimo de $F(y)$ e vice-versa.

A parte final desta seção é destinada a apresentação de algumas propriedades da função $F(y)$.

Dado $\hat{y} = (\hat{y}_1, \ldots, \hat{y}_m)^t \in \Re^m$, e considerando F(y) como definido em (2.6) defina a função de uma variável

$$\hat{f}_i(\theta) = F(\hat{y} + \theta e_i), \qquad i = 1,2, \ldots, m. \tag{2.9}$$

onde e_i denota a i-ésima coluna da matriz identidade de ordem m x m.

A função $\hat{f}_i(\theta)$ é estritamente convexa e contínua. O teorema a seguir estabelece a relação entre o problema (2.6) e o ponto de mínimo de $\hat{f}_i(\theta)$.

Teorema 2.2. O vetor $\hat{y} \in \Re^m$ resolve o problema (2.6) se e somente se para cada i=1,2, ... ,m, o ponto de mínimo de $\hat{f}_i(\theta)$ ocorre para $\theta = 0$.

O lema abaixo estabelece um limitante inferior para a função F(y).

Lema 2.1. Seja F(y) como definido em (2.6), então

$$\| y \|_1 \leq \frac{F(y)}{\varepsilon} + \gamma + \frac{1}{2} \varepsilon \gamma^2$$

onde

$$\gamma = 1 + \max_{i = 1, m} | b_i | . \tag{2.10}$$

3. DEFINIÇÃO DO ALGORITMO

Dax (1991) propôs o seguinte método iterativo seqüencial para calcular o ponto de mínimo da função

$$F(y) = \frac{1}{2} \| A^t y \|_2^2 - \varepsilon b^t y + \frac{1}{2} \left(\max \left\{ 0 , \sum_{i=1}^{m} | y_i | - 1 \right\} \right)^2 . \tag{3.1}$$

Cada iteração do algoritmo é composta de m passos, onde o i-ésimo passo, i = 1, 2, ..., m, considera a i-ésima linha de A, a qual é denotada por a_i^t. Seja $y = (y_1, \ldots, y_m)^t$ a estimativa atual da solução para o início do i-ésimo passo e defina

$$r = A^t y \tag{3.2}$$

$$\text{e} \qquad \rho = \sum_{i=1}^{m} | y_i | - 1 . \tag{3.3}$$

Para cada passo i, o valor y_i é substituído por $y_i + \theta^*$ onde θ^* minimiza a função de uma variável

$$f_i(\theta) = F(\boldsymbol{y} + \theta\, \boldsymbol{e}_i) = \frac{1}{2}\| \theta\, \boldsymbol{a}_i + \boldsymbol{r} \|_2^2 - \theta\varepsilon b_i - \varepsilon\, \boldsymbol{b}^t \boldsymbol{y} + \frac{1}{2}\left(\max\left\{ 0\,,\ |\, y_i + \theta\,| + \rho - |\, y_i| \right\}\right)^2 \qquad (3.4)$$

O i-ésimo passo do algoritmo é implementado por:

a) Faça $\theta = -\left(\boldsymbol{a}_i^t \boldsymbol{r} - \varepsilon\, b_i\right) / \boldsymbol{a}_i^t \boldsymbol{a}_i$, $\eta = y_i + \theta$ e $\rho := \rho - |\, y_i\,|$.

b) Se $\rho + |\, \eta\,| \leq 0$ vá para (d).

c) Se $\eta > 0$ faça $\theta := \theta + \max\left\{ -\eta\,,\ -(\eta + \rho) / \left(\boldsymbol{a}_i^t \boldsymbol{a}_i + 1\right) \right\}$.

Se $\eta < 0$ faça $\theta := \theta + \min\left\{ -\eta\,,\ -(\eta - \rho) / \left(\boldsymbol{a}_i^t \boldsymbol{a}_i + 1\right) \right\}$.

d) Faça $y_i := y_i + \theta$, $\boldsymbol{r} := \boldsymbol{r} + \theta\, \mathbf{a}_i$ e $\rho := \rho + |\, y_i\,|$.

O símbolo := denota atribuição aritmética.

Apresentamos agora a definição do novo algoritmo, o qual consiste em ser a versão paralela do algoritmo descrito anteriormente e será denominado algoritmo MINIMAXPAR.

Inicialização: Sejam $\mathbf{y}^0 \in \Re^m$ a aproximação inicial, ε um número real positivo e $\lambda_1, \lambda_2, \ldots, \lambda_m$ números reais positivos tais que

$$\sum_{i=1}^{m} \lambda_i = 1. \qquad (3.5)$$

Faça $$\boldsymbol{r}^0 = \boldsymbol{A}^t \boldsymbol{y}^0 \qquad (3.6)$$

e $$\rho^0 = \sum_{i=1}^{m} \left| y_i^0 \right| - 1\ . \qquad (3.7)$$

Iteração Principal: Para k = 0, 1, 2, ... faça

$$\boldsymbol{y}^{k-1} = \boldsymbol{y}^k + \Lambda \delta^k \qquad (3.8)$$

$$\boldsymbol{r}^{k-1} = \boldsymbol{r}^k + \sum_{i=1}^{m} \lambda_i\, \delta_i^k\, \boldsymbol{a}_i \qquad (3.9)$$

onde Λ é uma matriz diagonal com elementos diagonais λ_i's e δ^k é um m-vetor com componentes δ_i^k definidas por : para i = 1, 2, ..., m sejam

$$\theta_i^k = -\left(\langle\, \boldsymbol{a}_i\,,\, \boldsymbol{r}^k \rangle - \varepsilon\, b_i\right) / \langle\, \boldsymbol{a}_i\,,\, \boldsymbol{a}_i\, \rangle \qquad (3.10)$$

$$\eta_i = y_i^k + \theta_i^k \qquad (3.11)$$

$$\rho_i = \rho^k - \left| y_i^k \right| \qquad (3.12)$$

Se $\rho_i + \left| \eta_i \right| \leq 0$ faça $\delta_i^k = \theta_i^k$.

Se $\eta_i > 0$ faça

$$\delta_i^k = \max\left\{\frac{\theta_i^k - \eta_i}{\lambda_i},\ \theta_i^k - (\eta_i + \rho_i)\Big/(\langle a_i , a_i \rangle + 1)\right\}. \qquad (3.13)$$

Se $\eta_i < 0$ faça

$$\delta_i^k = \min\left\{\frac{\theta_i^k - \eta_i}{\lambda_i},\ \theta_i^k - (\eta_i - \rho_i)\Big/(\langle a_i , a_i \rangle + 1)\right\} \qquad (3.14)$$

Para cada iteração k, no cálculo de δ_i^k, usamos apenas a i-ésima linha da matriz A, assim todos os δ_i^k, i = 1, 2, ..., m podem ser calculados simultaneamente caso se utilize um computador com processamento paralelo.

No que se segue $\{y^k\}$ denota uma seqüência gerada pelo algoritmo MINIMAXPAR.

Lema 3.1. Para todo k, k = 0, 1, 2, ...

$$r^{k+1} = A^t y^{k+1}.$$

Prova: Por indução finita sobre k. Para k = 0, o resultado segue de (3.6). Vamos supor que o resultado seja válido para k e provar que o mesmo é verdadeiro para k + 1. De (3.9) vem,

$$\begin{aligned} r^{k+1} &= r^k + \sum_{i=1}^{m} \lambda_i \delta_i^k a_i \\ &= r^k + \sum_{i=1}^{m} \left(y_i^{k+1} - y_i^k\right) a_i \\ &= A^t y^k + A^t y^{k+1} - A^t y^k = A^t y^{k+1} \end{aligned}$$

onde a segunda igualdade ocorre de (3.8) e a terceira da hipótese de indução.

□

Observações:

3.1. O valor θ_i^k como definido em (3.10) é o ponto de mínimo da função de uma variável

$$\begin{aligned} h(\theta) &= \frac{1}{2}\left\| A^t\left(y^k + \theta e_i\right)\right\|_2^2 - \varepsilon b^t\left(y^k + \theta e_i\right) \\ &= \frac{1}{2}\left\| \theta a_i + r^k\right\|_2^2 - \theta \varepsilon b_i - \varepsilon b^t y^k \end{aligned}$$

onde a segunda igualdade segue do lema (3.1) e e_i denota a i-ésima coluna da matriz identidade de ordem m x m.

Agora as expressões (3.13) e (3.14), fazem a correção quando necessário, para que este ponto seja o mínimo da função $f_i(\theta)$ definida por:

$$f_i(\theta) = \frac{1}{2}\left\|\theta a_i + r^k\right\|_2^2 - \theta \varepsilon b_i - \varepsilon b^t y^k + \frac{1}{2}\left(\max\left\{0\ ,\ \left|\, y_i^k + \theta \,\right| + \rho_i - \left|\, y_i^k \,\right|\right\}\right)^2 \qquad (3.15)$$

3.2. O valor θ_i^k como definido em (3.10) pode também ser visto como sendo a variação de y_i no i-ésimo passo da iteração $k + 1$ do método de Jacobi para o sistema de equações lineares

$$AA^t y = \varepsilon b. \tag{3.16}$$

De fato, o esquema iterativo de Jacobi para (3.16) é definido para cada i por:

$$y_i^{k+1} = \frac{1}{\langle a_i , a_i \rangle}\left[\varepsilon b_i - \sum_{\substack{j=1 \\ j\neq i}}^{m} \langle a_i , a_j \rangle y_j^k\right]$$

$$= \frac{1}{\langle a_i , a_i \rangle}\left[\varepsilon b_i - \langle a_i , A^t y^k \rangle + \langle a_i , a_i \rangle y_i^k\right]. \tag{3.17}$$

Assim, do lema (3.1), de (3.10) e (3.17) temos que

$$y_i^{k+1} - y_i^k = \theta_i^k.$$

Logo o algoritmo MINIMAXPAR pode ser visto como sendo um do tipo Jacobi.

3.3. De (3.5) e (3.8) temos que para qualquer k = 0, 1, 2, ... , o iterado y^{k+1} pode ser escrito como:

$$y^{k+1} = \left(y_1^k + \lambda_1 \delta_1^k , y_2^k + \lambda_2 \delta_2^k , \ldots , y_m^k + \lambda_m \delta_m^k\right)$$

$$= \sum_{i=1}^{m} \lambda_i y^k + \left(\lambda_1 \delta_1^k, \lambda_2 \delta_2^k, \ldots , \lambda_m \delta_m^k\right)$$

$$= \lambda_1\left(y^k + \delta_1^k e_1\right) + \lambda_2\left(y^k + \delta_2^k e_2\right) + \ldots + \lambda_m\left(y^k + \delta_m^k e_m\right). \tag{3.18}$$

Assim, y^{k+1} é definido como sendo a combinação convexa dos vetores $y^k + \delta_i^k e_i$, i = 1, 2, ... m.

Agora da maneira como o algoritmo MINIMAXPAR foi definido, para qualquer i = 1, 2, ... ,m temos que

$$F\left(y^k + \delta_i^k e_i\right) \leq F\left(y^k\right). \tag{3.19}$$

O lema a seguir, mostra que a função objetivo F(y) definida em (3.1) é decrescente sobre a seqüência de iterados $\left\{y^k\right\}$ e converge.

Lema 3.2.

(i) $F\left(y^{k+1}\right) < F\left(y^k\right)$ para todo k , k = 0, 1, 2, ...

(ii) $\lim_{k\to\infty}\left(F\left(y^{k+1}\right) - F\left(y^k\right)\right) = 0.$

Prova:

(i) De (3.18) temos que

$$F\left(y^{k+1}\right) = F\left(\sum_{i=1}^{m}\lambda_i\left(y^k + \delta_i^k e_i\right)\right) < \sum_{i=1}^{m}\lambda_i F\left(y^k + \delta_i^k e_i\right) \leq \sum_{i=1}^{m}\lambda_i F\left(y^k\right) = F\left(y^k\right)$$ onde a

primeira desigualdade ocorre da convexidade estrita de $F(\cdot)$, a segunda de (3.19) e a última igualdade de (3.5).

(ii) Do item (i), $\left\{ F\left(y^k\right) \right\}$ é monotónica decrescente, do teorema (2.1) $F(y)$ possui um ponto de mínimo, ou seja, é limitada inferiormente, logo a seqüência $\left\{ F\left(y^k\right) \right\}$ converge e temos o resultado desejado.

□

Defina a função de uma variável

$$\psi(t) = F\left(y^k + t\left(y^{k+1} - y^k\right)\right) \tag{3.20}$$

onde y^k e y^{k+1} são iterados consecutivos gerados pelo algoritmo MINIMAXPAR e $F(.)$ é como definido em (3.1).

Sejam $t_1 = 0$ e $t_2 = 1$, então de (3.20) temos que $\psi(t_1) = F\left(y^k\right)$ e $\psi(t_2) = F\left(y^{k+1}\right)$. Agora do item (i) do lema (3.2) segue que

$$\psi(t_1) > \psi(t_2). \tag{3.21}$$

A função $\psi(t)$ é duas vezes continuamente diferenciável e estritamente convexa (de fato, um "spline" quadrático), então como $\psi(t_1) > \psi(t_2)$ segue que

$$\psi'(t_2) \leq 0. \tag{3.22}$$

Seja $H(y)$ a matriz Hessiana de $F(y)$ para o ponto y. Como $F(y)$ é duas vezes diferenciável, estritamente convexa é quadrática então $H(y)$ é definida positiva.

O lema abaixo estabelece a limitação da seqüência $\left\{ y^k \right\}$.

Lema 3.3. A seqüência $\left\{ y^k \right\}$ é limitada.

Prova: Do item (i) do lema (3.2) temos que para qualquer k, k = 1, 2, 3, ...

$$F\left(y^k\right) < F\left(y^{k-1}\right) < \ldots < F\left(y^1\right) < F\left(y^0\right). \tag{3.23}$$

Agora do lema (2.4) temos que

$$\left\| y^k \right\|_1 \leq \frac{F\left(y^k\right)}{\varepsilon} + \gamma + \frac{1}{2}\varepsilon\gamma \tag{3.24}$$

onde γ é dado por (2.10).

Assim de (3.23) e (3.24) segue que

$$\| y^k \|_1 \leq \frac{F(y^0)}{\varepsilon} + \gamma + \frac{1}{2}\varepsilon\gamma \qquad (3.25)$$

e temos o resultado desejado.

O lema a seguir é devido a Dax (1985) e será utilizado para a demonstração da convergência do algoritmo MINIMAXPAR.

Lema 3.4. Seja $\psi(t)$ uma função duas vezes continuamente diferenciável e sejam dois pontos $t_1 < t_2$ satisfazendo as seguintes condições:

1) $\psi(t_1) > \psi(t_2)$;

2) $\psi'(t_2) \leq 0$ e

3) $\psi''(t) \geq a$ para todo $t_1 \leq t \leq t_2$ onde a é uma constante positiva. Então

$$t_2 - t_1 \leq 2\left(\frac{\psi(t_1) - \psi(t_2)}{a}\right)^{1/2} \qquad (3.26)$$

Teorema 3.1. Seja $L = \left\{ y \,\middle|\, F(y) \leq F(y^0), \text{ onde } \| y \|_2 \leq c, \text{ c: constante} \right\}$, um conjunto de nivel. Cada ponto de acumulação da seqüência $\{y^k\}$ é um ponto de mínimo de $F(y)$ sobre L.

Prova: Fazendo $c = \frac{F(y^0)}{\varepsilon} + \gamma + \frac{1}{2}\varepsilon\gamma$ como em (3.25), então de (3.23) e do lema (3.3) temos que para qualquer k, k = 1,2, ..., o ponto y^k gerado pelo algoritmo MINIMAX é tal que $y^k \in L$.

Desde que L é um conjunto compacto e $H(y)$ é definida positiva, então existe uma constante $a > 0$ tal que $H(y) \geq aI$ para todo $y \in L$ (I : matriz identidade).

Seja $u = y^{k+1} - y^k$ e de (3.20), $\psi(t) = F(y^k + tu)$. Assim assumindo-se $u \neq 0$, temos que

$$\psi''(t) \geq a \| u \|_2^2 \quad . \qquad (3.27)$$

Logo considerando $t_1 = 0$, $t_2 = 1$ e de (3.21), (3.22), (3.27) e do lema (3.3) segue que

$$1 \leq 2\left(\frac{F(y^k) - F(y^{k-1})}{a\,\|u\|_2^2}\right)^{1/2}$$

ou

$$\|y^{k+1} - y^k\|_2 \leq 2\left(\frac{F(y^k) - F(y^{k+1})}{a}\right)^{1/2}. \qquad (3.28)$$

Passando o limite na expressão (3.28) quando $k \to \infty$ e do item (ii) do lema (3.2) obtemos que

$$\lim_{k\to\infty} \|y^{k+1} - y^k\|_2 = 0. \qquad (3.29)$$

Do lema (3.3), a seqüência $\{y^k\}$ é limitada, isto assegura a existência de pelo menos um ponto de acumulação. Seja $\hat{y}$ um ponto de acumulação de $\{y^k\}$, então existe uma subseqüência $\{y^{i_k}\}$ de $\{y^k\}$ tal que

$$\lim_{k\to\infty} y^{i_k} = \hat{y}. \qquad (3.30)$$

Para concluirmos a prova, devemos mostrar que para cada i, i = 1,2, ... m, o ponto de mínimo da função de uma variável

$$\hat{f}_i(\theta) = F(\hat{y} + \theta e_i) \qquad (3.31)$$

ocorre para $\theta = 0$.

De (3.30), (3.31) e da continuidade da função F vem,

$$\begin{aligned}\lim_{k\to\infty} F(y^{i_k} + \theta e_i) &= F\left(\lim_{k\to\infty} y^{i_k} + \theta e_i\right)\\ &= F(\hat{y} + \theta e_i)\\ &= \hat{f}_i(\theta).\end{aligned}$$

Ou seja, quando y^{i_k} aproxima-se de $\hat{y}$, o ponto de mínimo da função de uma variável

$$f_i(\theta) = F(y^{i_k} + \theta e_i)$$

aproxima-se daquele de $\hat{f}_i(\theta)$. Portanto se para algum índice i, zero não é um ponto de mínimo de $\hat{f}_i(\theta)$ então o limite definido em (3.29) seria violado.

Teorema 3.2. A seqüência $\{A^t y^k/\varepsilon\}$ converge para um ponto $\hat{x}$ tal que $\hat{x}$ é a solução única do problema (2.1).

Prova: A seqüência $\{A^t y^k/\varepsilon\}$ é limitada desde que $\{y^k\}$ é limitada. Seja $\hat{x}$ um ponto de acumulação de $\{A^t y^k/\varepsilon\}$ e seja $\{y^{i_k}\}$ uma subseqüência de $\{y^k\}$ tal que

$$\lim_{k \to \infty} A^t y^{j_k} / \varepsilon = \hat{x}. \tag{3.32}$$

Agora, $\{ y^{j_k} \}$ possui um ponto de acumulação $\hat{y}$, pois a mesma é limitada. Assim existe uma subseqüência $\{ y^{j_k} \}$ de $\{ y^{j_k} \}$ tal que

$$\lim_{k \to \infty} y^{j_k} = \hat{y} \tag{3.33}$$

e de (3.32) vem

$$\lim_{k \to \infty} A^t y^{j_k} / \varepsilon = \hat{x}. \tag{3.34}$$

Assim de (3.33) e (3.34) temos que

$$\hat{x} = A^t \hat{y} / \varepsilon \tag{3.35}$$

onde $\hat{y}$ é um ponto de acumulação de $\{ y^k \}$.

Portanto, pelo teorema (3.1) $\hat{y}$ é um ponto de mínimo de F(y) e pelo teorema (2.1) temos que $\hat{x}$ é a solução única do problema (2.1).

4. RESULTADOS NUMÉRICOS

Apresentamos a seguir, algumas experiências computacionais para o algoritmo paralelo MINIMAXPAR e o algoritmo seqüencial de Dax. Os algoritmos foram implementados em FORTRAN visual Workbench 1.0 da Microsoft (1993) e os testes foram realizados em um computador Pentium-S com CPU a 133 Mhz e 64 megabytes de memória RAM.

Para os dois algoritmos e em todos os exemplos, utilizamos a solução inicial $y^0 = (0, 0, \cdots, 0) \in \Re^m$ e para o algoritmo MINIMAXPAR usamos em todos os casos $\lambda_i = 1/m$ para $i = 1, 2, \cdots, m$, onde m denota o número de linhas da matriz A. Uma escolha mais refinada para os pesos λ_i poderá produzir melhores resultados de convergência para o algoritmo MINIMAXPAR.

O critério de parada considerado foi

$$\| y^{k+1} - y^k \|_\infty \leq \text{TOL}$$

ou o número de iterações excede 2.000 (TOL é um número real positivo e pequeno pré-fixado) O símbolo "*" indica que o algoritmo não convergiu após 2.000 iterações.

O valor ótimo é indicado por VO e calculado como

$$\text{VO} = \| Ax^* - b \|_\infty$$

onde $x^* = A^t y^* / \varepsilon$ e y^* é o valor fornecido pelo algoritmo.

Para os testes utilizamos diferentes valores de ε e a comparação entre os algoritmos foi feita através do número de iterações gastas, o qual está indicado por ITER, pelo valor da função objetivo VO e pelo tempo de CPU, o qual é indicado por TEMPO e representado como

min: sec. 10^{-2} sec.

Para problemas onde a dimensão da matriz A é pequena, não foi possível medir o tempo de CPU, pois a menor fração de tempo medida é de centésimos de segundo. Os casos mais interessantes são aqueles onde A é de médio para grande porte, e assim o tempo de CPU pode ser medido sem qualquer problema.

Consideramos dois tipos de problemas para os testes numéricos. No primeiro caso, A é uma matriz de pequeno a médio porte e a solução do sistema linear $Ax = b$ é conhecida e esta indicada por x^* (Exemplos 1, 2, 3 e 4). Utilizamos estes exemplos, como uma forma de verificar a exatidão da solução fornecida pelos algoritmos. No segundo caso, consideramos problemas gerados aleatoriamente. Para o exemplo 6, a seguir, a matriz A é esparsa, sem uma estrutura detectável e com $m >> n$, ou seja, o sistema linear $Ax = b$ é inconsistente. Assim, este é o principal exemplo desta seção, tendo em vista que os algoritmos apresentados são do tipo ação por linha e convenientes para este tipo de sistema linear.

Exemplo 1: Cheney (1982) pag. 44

$$A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 2 \\ 2 & 4 \\ 2 & 1 \\ 3 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 3 \\ 1 \\ 7 \\ 11.1 \\ 6.9 \\ 7.2 \end{bmatrix} \qquad \text{onde} \quad x^* = (2,\ 2)^t.$$

Exemplo 2:

$$A = \begin{bmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 4 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad \text{onde} \quad x^* = \left(\frac{36}{99}, \frac{45}{99}, \frac{45}{99}, \frac{36}{99}\right)^t.$$

Exemplo 3: Murty (1988) pag. 19.

A é uma matriz n x n, onde

$$a_{ij} = \begin{cases} 1 & \text{se } i = j \\ 0 & \text{se } i < j \\ 2 & \text{se } i > j \end{cases} \quad \text{e} \quad b_i = \sum_{j=1}^{n} a_{ij} \quad \text{para} \quad i = 1, 2, \cdots, m.$$

Assim A é uma matriz semidefinida positiva, com $\det A = 1$, e o sistema linear $Ax = b$ possui a solução única $x^* = (1, 1, \cdots, 1)^t$.

***Exemplo 4*:** Ruggiero e Lopes (1988) pag. 94.

$$A = \begin{bmatrix}
-\alpha & 0 & 0 & 1 & \alpha & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-\alpha & 0 & -1 & 0 & -\alpha & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -\alpha & -1 & 0 & 0 & \alpha & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \alpha & 0 & 1 & 0 & \alpha & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -\alpha & 0 & 0 & 1 & \alpha & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -\alpha & 0 & -1 & 0 & -\alpha & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & \alpha & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -\alpha & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\alpha & -1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\alpha & -1
\end{bmatrix}$$

$$b = \begin{bmatrix} 0 & 0 & 0 & 10 & 0 & 0 & 0 & 15 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 \end{bmatrix}^t$$

com $\alpha = \text{sen } 45^0 = \sqrt{2}/2 \cong 0.7071$.

Seguindo Dax e Berkowitz (1990) e Bartels et all (1989), usamos também para os testes numéricos, os seguintes sistemas lineares gerados aleatoriamente.

Exemplo 5: A é uma matriz m x n e esparsa, onde cada linha a_i^t de A possui η_i elementos não nulos, η_i é um número aleatório do conjunto {1, 2, 3, 4, 5}, a localização destes

elementos não nulos de a_i^t são números aleatórios do conjunto $\{ 1, 2, \cdots, n \}$ (A possui n colunas) e a_{ij} e b_i são números aleatórios do intervalo [-1 , 1].

Todos os números aleatórios foram gerados através da rotina RANDOM (ranval), onde ranval é o número aleatório fornecido e pertencente ao intervalo [0 , 1). A rotina SEED (seedval) é utilizada para variar o ponto inicial dos números pseudo-aleatórios gerados. Se seedval = −1 , então a seqüência de valores de RONDOM será sempre diferente.

Para este exemplo, o algoritmo de Dax não convergiu após 2.000 iterações com tolerância TOL = 10^{-4} e ε = 1, 0.1, 0.01 e 0.001 . Se A é de ordem 1000 x 100, o tempo gasto pelo método de Dax nas 2.000 iterações foi de aproximadamente 6,5 minutos.

Os resultados fornecidos pelo algoritmo MINIMAXPAR estão dispostos na tabela 10.

Exemplo 6: A matriz $A = 1/n\ G.G^t$ onde G é uma matriz quadrada de ordem n com g_{ij}, i, j = 1, 2, ⋯ , n , gerado aleatoriamente e tal que $g_{ij} \in [0 , 1)$. Assim A é uma matriz quadrada, de ordem n, simétrica e semidefinida positiva com $a_{ij} \in [0 , 1)$.

O vetor b é gerado também de maneira aleatória com $b_i \in [-1 , 1)$ para i = 1, 2, ⋯ , n.

Para este exemplo, o algoritmo de Dax não convergiu após 2.000 iterações com tolerância TOL = 10^{-4} e ε = 1 , 0.1 e 0.01 .

Os resultados apresentados pelo algoritmo MINIMAXPAR estão dispostos na tabela 11.

As tabelas a seguir apresentam os resultados das experiências computacionais efetuadas.

Tabela 1: Algoritmo de Dax para o exemplo 1 com TOL = 10^{-7} e diferentes valores de ε.

ε	ITER	VO	SOLUÇÃO
10^0	33	.10E + 01	(.20E + 01, .20E + 01)
10^{-1}	296	.10E + 01	(.20E + 01, .20E + 01)
10^{-2}	*	-	-

Tabela 2: Algoritmo MINIMAXPAR para o exemplo 1 com TOL = 10^{-7} e diferentes valores de ε.

ε	ITER	VO	SOLUÇÃO
10^{0}	534	.10E + 01	(.199998E + 01, .20E + 01)
10^{-1}	1021	.10E + 01	(.20E + 01, .20E + 01)
10^{-2}	*	-	-

Tabela 3: Algoritmo de Dax para o exemplo 2 com TOL = 10^{-7} e diferentes valores de ε.

ε	ITER	VO	SOLUÇÃO
10^{0}	20	.95E - 06	(.3636361, .4545454, .4545453, .3636362)
10^{-1}	17	.51E - 05	(.3636347, .4545440, .4545449, .3636361)
10^{-2}	14	.51E - 04	(.3636199), .4545313, .4545380, .3636346)
10^{-3}	11	.55E - 03	(.3634610, .4543945, .4544683, .3636170)
10^{-4}	8	.58E - 02	(.3617733, .4529407, .4537250, .3634313)

Tabela 4: Algoritmo MINIMAXPAR para o exemplo 2 com TOL = 10^{-7} e diferentes valores de ε.

ε	ITER	VO	SOLUÇÃO
10^{0}	146	.64E - 05	(.3636350, .4545429, .4545428, .3636348)
10^{-1}	118	.64E - 04	(.3636207, .4545191, .4545189, .3636208)
10^{-2}	90	.65E - 03	(.3634747, .4542726, .4542726, .3634746)
10^{-3}	63	.62E - 02	(.3620975, .4519546, .4519548, .3620974)
10^{-4}	35	.64E - 01	(.3477381, .4277822, .4277822, .3477381)

Tabela 5: Algoritmos de Dax e MINIMAXPAR para o exemplo 3 com n = 20, TOL = 10^{-4} e diferentes valores de ε.

	DAX			MINIMAXPAR		
ε	ITER	VO	TEMPO	ITER	VO	TEMPO
10^{1}	43	.4013E + 00	0:00.05	1043	.4156E + 00	0:00.27
10^{0}	231	.6299E - 01	0:00.06	1997	.1459E + 00	0:00.44
10^{-1}	308	.3704E - 01	0:00.06	94	.1522E + 01	0:00.06
10^{-2}	39	.4141E + 00	-	6	.7487E + 01	-

Tabela 6: Algoritmos de Dax e MINIMAXPAR para o exemplo 3 com n = 20, TOL = 10^{-6} e diferentes valores de ε.

	DAX			MINIMAXPAR		
ε	ITER	VO	TEMPO	ITER	VO	TEMPO
10^{1}	43	.4013E + 00	0:00.05	1129	.4015E + 00	0:00.28
10^{0}	309	.5561E - 01	0:00.05	*	-	-
10^{-1}	*	-	-	*	-	-
10^{-2}	*	-	-	1959	.1459E + 00	0:00.44
10^{-3}	308	.3686E - 01	0:00.06	94	.1522E + 01	0:00.06

Tabela 7: Algoritmos de Dax e MINIMAXPAR para o exemplo 3 com n = 100, TOL = 10^{-4} e diferentes valores de ε.

	DAX			MINIMAXPAR		
ε	ITER	VO	TEMPO	ITER	VO	TEMPO
10^{1}	209	.4000E + 00	0:01.04	*	-	-
10^{0}	1148	.7215E - 01	0:05.72	540	.3963E + 01	0:03.02
10^{-1}	214	.3930E + 00	0:01.04	9	.3512E + 02	0:00.05
10^{-2}	65	.2151E + 01	0:00.39	5	.9849E + 02	-

Tabela 8: Algoritmos de Dax e MINIMAXPAR para o exemplo 3 com n = 100, TOL = 10^{-3} e diferentes valores de ε.

	DAX			MINIMAXPAR		
ε	ITER	VO	TEMPO	ITER	VO	TEMPO
10^{1}	214	.4000E + 00	0:01.10	*	-	-
10^{0}	1484	.5599E - 01	0:07.42	*	-	-
10^{-1}	*	-	-	*	-	-
10^{-2}	*	-	-	540	.3963E + 01	0:03.02
10^{-3}	214	.3935E + 00	0:01.10	9	.3512E + 02	0:00.05

Tabela 9: Algoritmos de Dax e MINIMAXPAR para o exemplo 4 com TOL = 10^{-4} e diferentes valores de ε.

	DAX			MINIMAXPAR		
ε	ITER	VO	TEMPO	ITER	VO	TEMPO
10^{3}	*	-	-	343	.7631E + 01	0:00.22
10^{2}	77	.768E + 01	0:00.06	286	.7628E + 01	0:00.06
10^{1}	67	.7699E + 01	0:00.05	227	.7600E + 01	0:00.05
10^{0}	58	.7315E + 01	0:00.05	165	.7320E + 01	0:00.05
10^{-1}	*	-	-	145	.5546E + 01	0:00.05
10^{-2}	*	-	-	285	.2911E + 01	0:00.06
10^{-3}	*	-	-	207	.3235E + 01	0:00.06

Tabela 10: Algoritmo MINIMAXPAR para o exemplo 5 com TOL = 10^{-4}.

	m = 200 e n = 100			m = 1000 e n = 100		
ε	ITER	VO	TEMPO	ITER	VO	TEMPO
10^{0}	1015	.9826E + 00	0:01.55	139	.9981E + 00	0:18.56
10^{-1}	217	.9942E + 00	0:00.33	11	.1011E + 01	0:01.59
10^{-2}	45	.1046E + 01	0:00.06	34	.1221E + 01	0:04.67
10^{-3}	305	.1184E + 01	0:00.44	107	.2589E + 01	0:14.28

Tabela 11: Algoritmo MINIMAXPAR para o exemplo 6 com TOL = 10^{-4}.

	n = 20			n = 100		
ε	ITER	VO	TEMPO	ITER	VO	TEMPO
10^{0}	622	.8433E + 00	0:00.11	699	.9784E + 00	0:04.17
10^{-1}	738	.8104E + 00	0:00.16	123	.1022E + 01	0:00.77
10^{-2}	312	.7416E + 00	0:00.06	4	.1041E + 01	-

5. CONCLUSÕES FINAIS

De uma maneira geral, algoritmos seqüenciais (tipo Gauss-Seidel) como é o caso do algoritmo de Dax (1991) tem convergência mais rápida do que algoritmos simultâneos (tipo Jacobi) como é o caso do algoritmo MINIMAXPAR. Entretanto os métodos simultâneos são convenientes para a utilização em computadores com processamento paralelo. Se um método tipo Jacobi, gasta em um computador convencional (1 processador central), um tempo de CPU $t = t_0$ para um determinado problema, com uma matriz A possuindo m linhas, então o tempo gasto para esse mesmo problema em um computador com m processadores paralelos é da ordem de $0(t) = t_0/m$, existe um tempo de sincronização/ comunicação entre os processadores.

Com base em nossas experiências computacionais para os algoritmos de Dax e MINIMAXPAR e para os exemplos considerados podemos concluir que:

1. Para aqueles sistemas lineares onde a solução x^* era conhecida, exemplos 1 e 2 tanto o algoritmo de Dax como o MINIMAXPAR forneceram excelentes aproximações para x^* para $\varepsilon = 1$, $\varepsilon = 10^{-1}$ ou $\varepsilon = 10^{-2}$. Quanto menor é o valor de ε maior é o valor de VO, ou seja, a aproximação tende a ficar pior com a diminuição de ε.

2. Para o exemplo 3, o valor ótimo da função objetivo, mostrou-se apenas razoável. Embora não tabelado, dos testes observamos que as primeiras componentes do vetor solução apresentado pelos algoritmos eram bastante próximas de 1 enquanto que para as últimas componentes, isso não ocorreu. A consideração de uma menor tolerância, não trouxe ganhos significativos para os dois métodos, pois neste caso, o número de iterações aumentou significativamente enquanto que o valor ótimo VO permaneceu com a mesma ordem de grandeza.

Considerando n = 20 e VO da ordem de 10^0, o algoritmo de Dax gastou um tempo de CPU igual a $5x10^{-2}$ sec e o MINIMAXPAR $28x10^{-2}$ sec. Para nenhum valor de ε, o algoritmo MINIMAXPAR obteve VO da ordem 10^{-1}. (Tabelas 5 e 6). Para o caso onde n = 100 o algoritmo seqüencial de Dax teve um desempenho superior ao do algoritmo MINIMAXPAR (Tabelas 7 e 8).

3. Para o exemplo 4, o desempenho do algoritmo MINIMAXPAR foi superior ao de Dax. O algoritmo de Dax apresentou o melhor resultado para ε = 1 com um tempo de CPU igual a $5x10^{-2}$ sec em 58 iterações e VO da ordem de 10^1, enquanto que o método MINIMAXPAR obteve para ε = 10^{-1} em 145 iterações o mesmo tempo de CPU e a mesma ordem de grandeza de VO do que o método de Dax.

4. Para os exemplos 5 e 6, o algoritmo de Dax não convergiu após 2.000 iterações para os valores de ε considerados. A utilização de uma menor tolerância neste caso, não traz ganhos significativos para a ordem de grandeza de VO, para o algoritmo MINIMAXPAR.

Do exposto, e de nossas experiências computacionais, consideramos que o algoritmo MINIMAXPAR é conveniente para ser utilizado quando da disponibilidade de um computador com arquitetura paralela e o sistema linear $\boldsymbol{Ax} = \boldsymbol{b}$ a ser resolvido é inconsistente, com $\boldsymbol{A}$ de grande porte, esparsa e sem uma estrutura detectável.

6. BIBLIOGRAFIA

[1] R.H. BARTELS, A.R. CONN and Y. LI, "*Primal methods are better than dual methods for solving overdetermined linear systems in the l_∞ sense*,"? SIAM J. Numer. Anal, 26(1989), pp.693-726.

[2] Y. CENSOR, "*Row-action methods for huge and sparse systems and their applications*", SIAM Review, 23(1981), pp.444-464.

[3] Y. CENSOR and S. ZENIOS, "*Introduction to methods of parallel optimization*", 19° Colóquio Brasileiro de Matemática, IMPA-CNPq, Rio de Janeiro-RJ, (1993).

[4] E.W. CHENEY, "*Introduction to aproximation theory*", Chelsea Publishing Company, New York, N.Y. (1982).

[5] A. DAX, "*A row relaxation method for large minimax problems*", Hidrological Service, P.O.B. 6381, Jerusalem 91060, Israel, (1991).

[6] A. DAX. *"Sucessive refinemnt of large multicell models"*, SIAM, J. Numer. Anal., 22(1985), pp.865-887.

[7] A. DAX and B. BERKOWITZ, *"Column relaxation methods for least norm problems"*, SIAM J. Sci. Stat. Comput., 11(1990), pp.975-989.

[8] J.M. LOPES e A.R. DE PIERRO, *"Um método iterativo paralelo para estimação de mínimo valor absoluto"*, Pesquisa Operacional. Sobrapo, Rio de Janeiro, R.J., 12(1992) pp.31-46.

[9] O.L. MANGARSARIAN, *"Iterative solution of linear programs"*, SIAM J. Numer. Anal., 18(1981), pp.606-614.

[10] K.G. Murty, *"Linear complementarity, linear and nonlinear programming"*, Heldermann Verlag, Berlin, (1988).

[11] M.A.G. RUGGIERO e V.L.R. LOPES, *"Cáculo numérico - aspectos teóricos e computacionais"*, McGraw-Hill, São Paulo-SP (1988).

[12] V.A. SPOSITO, *"Linear and nonlinear programming"*, The Iowa State University Press, N.I. (1975).

[13] A.N. TIKHNOV and V.Y. ARSENIN, *"Solutions of ill-posed problems"*, V.H. Winston & Sons, Washington, (1977).

7. APÊNDICE

Apresentamos neste apêndice os programas FORTRAN elaborados para os testes numéricos.

A subrotina GETTIM é utilizada para a leitura do relógio do microcomputador.

6.1 - Algoritmo de Dax.

```
      DIMENSION A(100,100),B(100),Y(100),R(100),XNORMA(100),SOL(100)
      OPEN(unit=8,file='prn')
      OPEN(2,FILE='WDADOS',STATUS='OLD')
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCC
C     ESTIMACAO NO ESPACO L(INFINITO) PELO METODO DE DAX
C
C                         VARIAVEIS UTILIZADAS
C
C   A=MATRIZ DE RESTRICOES
```

```
C    M=NUMERO DE LINHAS DA MATRIZ A
C    N=NUMERO DE COLUNAS DA MATRIZ A
C    B=VETOR INDEPENDENTE
C    Y=SOLUCAO DO PROBLEMA DUAL
C    SOL=SOLUCAO DO PROBLEMA PRIMAL
C    R=VETOR RESIDUAL
C    VO=VALOR OTIMO
C    IND=INDICADOR DE CONVERGENCIA
C    KMAX=NUMERO MAXIMO DE ITERACOES PERMITIDAS
C    XNORMA=(NORMA DAS LINHAS DA MATRIZ A)**0.5
C    TOL=TOLERANCIA UTILIZADA NO CRITERIO DE PARADA
C    EPSILON=CONSTANTE POSITIVA
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
C
      KMAX=2000
      TOL=1.0E-4
      WRITE(*,*)' FORNECA O VALOR DE EPSILON'
      READ(*,*)EPSILON
      READ(2,*)M,N,A,B
      WRITE(*,*)' ESTIMACAO NO ESPACO L(INFINITO)-METODO DE DAX96-V1'
      WRITE(*,11)M,N,KMAX,EPSILON,TOL
      WRITE(8,5)
    5 FORMAT(///,' ESTIMACAO NO ESPACO L(INFINITO)-METODO DE DAX96-V1')
      WRITE(8,11)M,N,KMAX,EPSILON,TOL
   11 FORMAT(/,' M=',I4,3X,' N=',I4,3X,' KMAX=',I5,3X,' EPSILON=',F14.7,
     *3X,' TOL=',F9.7)
      DO 13 I=1,M
      WRITE(8,16)(A(I,J),J=1,N)
   13 CONTINUE
      WRITE(8,16)(B(I),I=1,M)
   16 FORMAT(1X,5F14.7)
C
C         SOLUCAO INICIAL
C
      DO 20 I=1,M
      Y(I)=0.0
   20 CONTINUE
C
C        LEITURA DO RELOGIO DO SISTEMA
C
   25 CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(*,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(8,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
C
C        CALCULO DO VETOR RESIDUAL R E DE RO
C
      DO 40 I=1,N
```

```
      AUX=0.0
      DO 30 J=1,M
      AUX=AUX+A(J,I)*Y(J)
 30   CONTINUE
      R(I)=AUX
 40   CONTINUE
C
C       CALCULO DA NORMA**0.5 DAS LINHAS DA MATRIZ A
C
      DO 70 I=1,M
      AUX=0.0
      DO 60 J=1,N
      AUX=AUX+A(I,J)*A(I,J)
 60   CONTINUE
      XNORMA(I)=AUX
 70   CONTINUE
C
C       ITERACAO PRINCIPAL
C
      DO 240 K=1,KMAX
      RO=0.0
      DO 50 I=1,M
      RO=RO+ABS(Y(I))
 50   CONTINUE
      RO=RO-1.0
      IND=0
      DO 140 I=1,M
C
C       CALCULO DE TETA
C
       AUX=0.0
      DO 80 J=1,N
      AUX=AUX+A(I,J)*R(J)
 80   CONTINUE
      TETA=-(AUX-EPSILON*B(I))/XNORMA(I)
      XNETA=Y(I)+TETA
      RO=RO-ABS(Y(I))
      AUX1=RO+ABS(XNETA)
      IF(AUX1)100,100,90
 90   IF (XNETA.GE.0.0) THEN
      TETA=TETA+AMAX1(-XNETA,-(XNETA+RO)/(XNORMA(I)+1.0))
      ELSE IF (XNETA.LT.0.0) THEN
      TETA=TETA+AMIN1(-XNETA,-(XNETA-RO)/(XNORMA(I)+1.0))
      END IF
C
C       CALCULO DOS NOVOS VETORES R E RO E DA NOVA SOLUCAO Y
C
100  DO 110 J=1,N
      R(J)=R(J)+TETA*A(I,J)
110  CONTINUE
      AUX=Y(I)+TETA
      RO=RO+ABS(Y(I))
```

```
      IF(ABS(AUX-Y(I))-TOL)130,130,120
  120 IND=1
  130 Y(I)=AUX
  140 CONTINUE
      IF(IND)240,150,240
C
C         CALCULO DO VETOR SOLUCAO E DO VALOR DA FUNCAO OBJETIVO
PRIMAL
C
  150 CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(8,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(*,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      DO 170 I=1,N
      AUX=0.0
      DO 160 J=1,M
      AUX=AUX+A(J,I)*Y(J)
  160 CONTINUE
      SOL(I)=AUX/EPSILON
  170 CONTINUE
      AUX=0.0
      DO 180 J=1,N
      AUX=AUX+A(1,J)*SOL(J)
  180 CONTINUE
      VO=ABS(AUX-B(1))
      DO 210 I=2,M
      AUX=0.0
      DO 190 J=1,N
      AUX=AUX+A(I,J)*SOL(J)
  190 CONTINUE
      VO1=ABS(AUX-B(I))
      IF(VO1-VO)210,210,200
  200 VO=VO1
  210 CONTINUE
      WRITE(*,220)K,VO
      WRITE(8,220)K,VO
  220 FORMAT(/,' SOLUCAO APOS',I5,' ITERACOES',5X,' VALOR OTIMO=',E14.7)
      WRITE(8,223)(SOL(I),I=1,N)
  223 FORMAT(5F14.7)
      GO TO 260
  240 CONTINUE
      CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(8,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(*,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      WRITE(*,250)KMAX,TOL
      WRITE(8,250)KMAX,TOL
  250 FORMAT(/,'O ALGORITMO NAO CONVERGIU APOS',I5,'ITERACOES COM
```


```
     *TOLERANCIA',E14.7')
260 CLOSE(2)
      STOP
      END
```

6.2 - Algoritmo MINIMAXPAR

```
      DIMENSION A(100,100),B(100),Y(100),R(100),XNORMA(100),SOL(100),TET
     *A(100),XRO(100)
      OPEN(unit=8,file='prn')
      OPEN(2,FILE='W5DADOS',STATUS='OLD')
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
C
C       ESTIMACAO NO ESPACO L(INFINITO) PELO METODO MINIMAXPAR
C                       (VERSAO PARALELA DO METODO DE DAX)
C
C       VARIAVEIS UTILIZADAS
C
C     A=MATRIZ DE RESTRICOES
C     M=NUMERO DE LINHAS DA MATRIZ A
C     N=NUMERO DE COLUNAS DA MATRIZ A
C     B=VETOR INDEPENDENTE
C     Y=SOLUCAO DO PROBLEMA DUAL
C     SOL=SOLUCAO DO PROBLEMA PRIMAL
C     R=VETOR RESIDUAL
C     VO=VALOR OTIMO
C     IND=INDICADOR DE CONVERGENCIA
C     KMAX=NUMERO MAXIMO DE ITERACOES PERMITIDAS
C     XNORMA=(NORMA DAS LINHAS DA MATRIZ A)**0.5
C     TOL=TOLERANCIA UTILIZADA NO CRITERIO DE PARADA
C     EPSILON=CONSTANTE POSITIVA
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCC
      KMAX=2000
      TOL=1.0E-4
      WRITE(*,*)' FORNECA O VALOR DE EPSILON'
      READ(*,*)EPSILON
      READ(2,*)M,N,A,B
      WRITE(*,*)' ESTIMACAO NO ESPACO L(INFINITO)-MET. DE DAX-PARALELO'
      WRITE(*,11)M,N,KMAX,EPSILON,TOL
      WRITE(8,5)
  5    FORMAT(//,' ESTIMACAO NO ESPACO L(INFINITO)-MET. DE DAX-
PARALELO')
      WRITE(8,11)M,N,KMAX,EPSILON,TOL
 11   FORMAT(/,' M=',I4,3X,' N=',I4,3X,' KMAX=',I5,3X,' EPSILON=',F14.7,
     *3X,' TOL=',F9.7)
      WRITE(8,*)'MATRIZ A E VETOR B'
```

```
      DO 15 I=1,M
   15 WRITE(8,16)(A(I,J),J=1,N)
   16 FORMAT(/,5F14.7)
      WRITE(8,16)(B(I),I=1,M)
C
C       SOLUCAO INICIAL E PESOS LAMBDA
C
      DO 20 I=1,M
      Y(I)=0.0
   20 CONTINUE
      XLAMBDA=1.0/FLOAT(M)
C
C       LEITURA DO RELOGIO DO SISTEMA
C
      CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(*,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(8,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
C
C       CALCULO DO VETOR RESIDUAL R E DE RO
C
      DO 40 I=1,N
      AUX=0.0
      DO 30 J=1,M
      AUX=AUX+A(J,I)*Y(J)
   30 CONTINUE
      R(I)=AUX
   40 CONTINUE
C
C       CALCULO DA NORMA**0.5 DAS LINHAS DA MATRIZ A
C
      DO 70 I=1,M
      AUX=0.0
      DO 60 J=1,N
      AUX=AUX+A(I,J)*A(I,J)
   60 CONTINUE
      XNORMA(I)=AUX
   70 CONTINUE
C
C       ITERACAO PRINCIPAL
C
      DO 250 K=1,KMAX
      RO=0.0
      DO 50 I=1,M
      RO=RO+ABS(Y(I))
   50 CONTINUE
      RO=RO-1.0
      IND=0
      DO 130 I=1,M
C
```

```
C      CALCULO DE TETA(I,K) E DO NOVO VETOR Y
C
       AUX=0.0
       DO 80 J=1,N
       AUX=AUX+A(I,J)*R(J)
  80   CONTINUE
       TETA(I)=-(AUX-EPSILON*B(I))/XNORMA(I)
       XNETA=Y(I)+TETA(I)
       XRO(I)=RO-ABS(Y(I))
       A1=(TETA(I)-XNETA)/XLAMBDA
       AUX1=XRO(I)+ABS(XNETA)
       IF(AUX1)100,100,90
  90   IF (XNETA.GE.0) THEN
       A2=TETA(I)-(XNETA+XRO(I))/(XNORMA(I)+1.0)
       TETA(I)=AMAX1(A1,A2)
       ELSE IF (XNETA.LT.0) THEN
       A2=TETA(I)-(XNETA-XRO(I))/(XNORMA(I)+1.0)
       TETA(I)=AMIN1(A1,A2)
       END IF
 100   AUX=Y(I)+XLAMBDA*TETA(I)
       IF(ABS(AUX-Y(I))-TOL)120,120,110
 110   IND=1
 120    Y(I)=AUX
 130   CONTINUE
C
C      CALCULO DO NOVO VETOR R
C
       DO 150 I=1,N
       AUX=0.0
       DO 140 J=1,M
       AUX=AUX+XLAMBDA*TETA(J)*A(J,I)
 140   CONTINUE
       R(I)=R(I)+AUX
 150   CONTINUE
       IF(IND)250,160,250
C
C        CALCULO DO VETOR SOLUCAO E DO VALOR DA FUNCAO OBJETIVO
PRIMAL
C
 160   CALL GETTIM(ihr,imin,isec,i100th)
       WRITE(8,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
      *h
       CALL GETTIM(ihr,imin,isec,i100th)
       WRITE(*,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
      *h
       DO 180 I=1,N
       AUX=0.0
       DO 170 J=1,M
       AUX=AUX+A(J,I)*Y(J)
 170   CONTINUE
       SOL(I)=AUX/EPSILON
 180   CONTINUE
```

```
      AUX=0.0
      DO 190 J=1,N
      AUX=AUX+A(1,J)*SOL(J)
190   CONTINUE
      VO=ABS(AUX-B(1))
      DO 220 I=2,M
      AUX=0.0
      DO 200 J=1,N
      AUX=AUX+A(I,J)*SOL(J)
200    CONTINUE
      VO1=ABS(AUX-B(I))
      IF(VO1-VO)220,220,210
210   VO=VO1
220   CONTINUE
      WRITE(*,230)K,VO
      WRITE(8,230)K,VO
230   FORMAT(/,' SOLUCAO APOS',I7,' ITERACOES',5X,' VALOR OTIMO=',E14.7)
      WRITE(*,240)(SOL(I),I=1,N)
      WRITE(8,240)(SOL(I),I=1,N)
240   FORMAT(1X,'VETOR SOLUCAO',/,2X,5E14.7)
      GO TO 270
250   CONTINUE
      WRITE(*,260)KMAX,TOL
      WRITE(8,260)KMAX,TOL
260     FORMAT(/,'O ALGORITMO NAO CONVERGIU APOS',I5,'ITERACOES COM
TOLERANCIA',E14.7)
270   CLOSE(2)
      STOP
      END
```

6.3 - Sistema Linear gerado aleatoriamente.

```
     DIMENSION A(1000,100),B(1000),NI(1000)
     OPEN(2,FILE='WDADOS',STATUS='NEW')
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
C
C       ESTE PROGRAMA GERA UM SISTEMA LINEAR AX=B COM UMA MATRIZ A
C       ESPARSA ONDE A(I,J) E  B(I) SAO NUMEROS ALEATORIOS DO INTERVALO
C       [-1,1].CADA LINHA DE A POSSUI NI ELEMENTOS NAO NULOS ONDE NI E
C       UM NUMERO ALEATORIO PERTENCENTE AO CONJUNTO {1,2,3,4,5}
C
C               VARIAVEIS UTILIZADAS
C
C       A=MATRIZ DE RESTRICOES
C       M=NUMERO DE LINHAS DA MATRIZ A
C       N=NUMERO DE COLUNAS DA MATRIZ A
C       B=VETOR INDEPENDENTE
```

```
C      NI=NUMERO DE ELEMENTOS NAO NULOS DA LINHA I
C
C            SUBROTINAS UTILIZADA
C
C    RANDOM(ranval) GERA NUMEROS ALEATORIOS NO INTERVALO [0.0,1.0)
C    SEED(-1) SEMENTE PARA A SUBROTINA RANDOM
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
C
    WRITE(*,*)'FORNECA OS VALORES DE M E N'
    READ(*,*)M,N
C
C      GERACAO DA MATRIZ A
C
    CALL SEED(-1)
    DO 10 I=1,M
    DO 10 J=1,N
10  A(I,J)=0.0
    DO 40 I=1,M
20  CALL RANDOM(ranval)
    NI(I)=10.0*ranval
    IF(NI(I))25,20,25
25  IF(NI(I)-5)30,30,20
30  DO 40 I1=1,NI(I)
    CALL RANDOM(ranval)
    J=IFIX(N*(ranval+1.0/FLOAT(N)))
    CALL RANDOM(ranval)
    A(I,J)=2*(ranval-0.5)
40  CONTINUE
C
C      GERACAO DO VETOR B
C
    DO 50 I=1,M
    CALL RANDOM(ranval)
    B(I)=2.0*(ranval-0.5)
50  CONTINUE
    WRITE(2,*)M,N,A,B
    CLOSE(2)
     END
```

6.4 Sistema Linear gerado aleatoriamente.

```
    DIMENSION A(200,200),B(200),G(200,200)
    OPEN(2,FILE='WDADOS',STATUS='NEW')
    OPEN(unit=8,file='prn')
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
```


```
C
C     ESTE PROGRAMA GERA UM SISTEMA LINEAR ALEATORIO ONDE A E UMA
C          MATRIZ QUADRADA DE ORDEM N, SIMETRICA E SEMI DEFINIDA
POSITIVA,
C     COM A(I,J) PERTENCENTES A [0,1) E B(I) PERTENCENTES A [-1,1].
C
C        VARIAVEIS UTILIZADAS
C
C     A=MATRIZ DO SISTEMA LINEAR
C     B=VETOR INDEPENDENTE
C     N=ORDEM DA MATRIZ A
C
C        SUBROTINAS UTILIZADAS
C
C     RANDOM(RANVAL) GERA UM NUMERO ALEATORIO NO INTERVALO [0.1).
C     SEED(-1) SEMENTE PARA A SUBROTINA RANDOM .
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC
C
    WRITE(*,*)'FORNECA O VALOR DE N'
    READ(*,*)N
    CALL SEED(-1)
    DO 10 I=1,N
    DO 10 J=1,N
    CALL RANDOM(ranval)
    G(I,J)=ranval
10  CONTINUE
    DO 20 I=1,N
    DO 20 J=1,N
    A(I,J)=0.0
    DO 15 K=1,N
    A(I,J)=A(I,J)+G(I,K)*G(J,K)
15  CONTINUE
    A(I,J)=A(I,J)/FLOAT(N)
20  CONTINUE
    DO 30 I=N,1,-1
    CALL RANDOM(ranval)
    B(I)=2*(ranval-0.5)
30  CONTINUE
    WRITE(8,*)' MATRIZ A'
    DO 40 I=1,N
    WRITE(8,50)(A(I,J),J=1,N)
40  CONTINUE
50  FORMAT(1X,5E14.7)
    WRITE(8,*)' VETOR B'
    WRITE(8,50)(B(I),I=1,N)
    WRITE(2,*)N,A,B
    CLOSE(2)
    END
```


1. INTRODUÇÃO

Apresentamos neste trabalho um algoritmo iterativo paralelo para resolver o problema

$$\text{minimize } F(\boldsymbol{x}) = \| \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \|_p^p \qquad (1.1)$$

onde $1 < p < \infty$, $\boldsymbol{A}$ é uma matriz real $m \times n$, $\boldsymbol{b} = (b_1, b_2, \ldots, b_m)^t \in \Re^m$, $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^t \in \Re^n$ é o vetor de incógnitas e t denota transposto.

A norma ℓ_p é definida como

$$\| \mathbf{y} \|_p = \left[\sum_{i=1}^{m} |y_i|^p \right]^{1/p} \quad \text{se } 1 < p < +\infty, \text{ com } \boldsymbol{y} = (y_1, y_2, \ldots, y_m)^t \in \Re^m .$$

Estamos interessados no caso onde o sistema linear $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ não tem solução, $\boldsymbol{A}$ é uma matriz esparsa, de grande porte e sem uma estrutura detectável. Nestes casos métodos que usam modificações da matriz $\boldsymbol{A}$ não são convenientes e até mesmo inaplicáveis dependendo da ordem de $\boldsymbol{A}$. Uma alternativa para resolver o problema (1.1) são os métodos de relaxação coluna (linha) como aqueles propostos em Dax-Berkowitz (1990) ou Lopes (1992), estes métodos utilizam apenas uma coluna (linha) da matriz $\boldsymbol{A}$ em cada passo de cada iteração. Se o método é paralelo então todas as colunas (variáveis) podem ser processadas simultaneamente quando da utilização de um computador com processadores em paralelo.

2. DEFINIÇÃO DO ALGORITMO

Consideremos por um momento um problema mais geral do que (1.1); ou seja

$$\text{minimize } F(\boldsymbol{x}) = \sum_{i=1}^{m} f_i(< \boldsymbol{a}_i, \boldsymbol{x} > - b_i) \qquad (2.1)$$

onde as funções f_i são convexas, $\boldsymbol{a}_i$ é a i-ésima linha da matriz $\boldsymbol{A}$ e $<.,.>$ denota o produto escalar euclidiano em $\Re^n$.

Seja x^k um ponto arbitrário em $\Re^n$ e sejam para $i = 1, 2, \ldots, m$, $\{\lambda_j^i\}$ para $j=1, \ldots, n$ números reais não negativos tais que

$$\sum_{j=1}^{n} \lambda_j^i = 1. \tag{2.2}$$

Para cada linha i da matriz A, consideramos n pesos λ_j^i com soma igual a 1. Com o objetivo de simplificar a notação consideraremos $\lambda_j^i \neq 0$ para qualquer i, j.

A função objetivo $F(x)$ em (2.1) pode ser escrita como

$$F(x) = \sum_{i=1}^{m} f_i \left[\sum_{j=1}^{n} \lambda_j^i \left(\frac{1}{\lambda_j^i} a_{ij} x_j + < a_i , x^k > - \frac{1}{\lambda_j^i} a_{ij} x_j^k - b_i \right) \right]$$

$$\leq \sum_{j=1}^{n} \sum_{i=1}^{m} \lambda_j^i f_i \left[\frac{a_{ij} x_j}{\lambda_j^i} + d_j^{ik} \right] \tag{2.3}$$

onde,

$$d_j^{ik} = < a_i , x^k > - \frac{1}{\lambda_j^i} a_{ij} x_j^k - b_i \tag{2.4}$$

e a desigualdade segue da convexidade de f_i.

Definindo,

$$\varphi_j \left(x_j , x^k \right) = \sum_{i=1}^{m} \lambda_j^i f_i \left[\frac{a_{ij} x_j}{\lambda_j^i} + d_j^{ik} \right] \tag{2.5}$$

e

$$\varphi \left(x , x^k \right) = \sum_{j=1}^{n} \varphi_j \left(x_j , x^k \right) \tag{2.6}$$

temos de (2.3) que

$$F(x) \leq \varphi \left(x , x^k \right) \tag{2.7}$$

assim, podemos aproximar o mínimo de $F(x)$ através do mínimo de $\varphi \left(x , x^k \right)$.

Observações:

2.1. Se $x = x^k$ então a igualdade se verifica em (2.7). As variáveis de $\varphi(x, x^k)$ estão separadas o que facilita em muito a determinação de seu ponto de mínimo. Veja a figura (2.1).

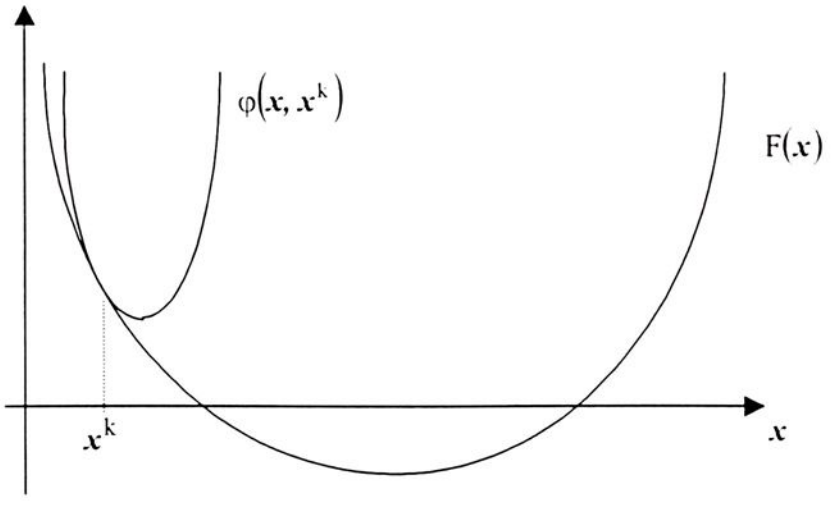


Figura (2.1): Relação entre as funções $F(x)$ e $\varphi(x, x^k)$ para o caso unidimensional.

2.2. A função $\varphi_j(x_j, x^k)$ definida em (2.5) é função de uma única variável, a saber, x_j e no caso que estamos considerando aqui, $f_i(x) = |x|^p$; ou seja;

$$\varphi_j(x_j, x^k) = \sum_{i=1}^{m} \lambda_j^i \left| \frac{a_{ij} x_j}{\lambda_j^i} + d_j^{ik} \right|^p . \qquad (2.8)$$

Para um j fixo, $j = 1, \ldots, n$, os elementos da matiz A que aparecem na definição da função φ_j são $a_{1j}, a_{2j}, \ldots, a_{mj}$; ou seja; a j-ésima coluna da matriz A, por isso, este é um método do tipo relaxação por coluna. Observe que d_j^{ik} é uma constante.

2.3. A função objetivo $F(x)$ em (1.1) é uma função convexa pois é uma norma e de (2.6) segue que $\varphi(x, x^k)$ é também uma função convexa pois é dada pela soma de funções convexas.

Definimos a seguir o novo algoritmo o qual será denominado LPPAR.

ALGORITMO LPPAR

1. Passo inicial: Sejam $x^0 \in \Re^n$ uma aproximação inicial arbitrária e ε um número real pequeno positivo dado.

2. Passo principal: Para $k = 0, 1, 2, \ldots$ calcule x_j^{k+1} para cada j, $j = 1, 2, \ldots, n$. Determine inicialmente $\overline{x}_j^{k+1}$ tal que

$$\varphi_j\left(x_j^k, x^k\right) = \varphi_j\left(\overline{x}_j^{k+1}, x^k\right); \tag{2.9}$$

ou seja; se

$$f(x) = \varphi_j\left(x, x^k\right) - \varphi_j\left(x_j^k, x^k\right) \tag{2.10}$$

devemos determinar um zero de f(x) diferente de x_j^k.

Defina a direção

$$d_j^k = f'\left(x_j^k\right) \tag{2.11}$$

e determine o zero $\overline{\alpha}$ de

$$g(\alpha) = f\left(x_j^k - \alpha d_j^k\right) \tag{2.12}$$

pelo método de Newton, considerando como aproximação inicial α_0, o primeiro valor de $\alpha = 4^\ell$, $\ell = 0, 1, 2, \ldots$. tal que $g(\alpha) > 0$.

Assim,

$$\overline{x}_j^{k+1} = x_j^k - \overline{\alpha} d_j^k \tag{2.13}$$

e

$$x_j^{k+1} = x_j^k + \beta\left(\bar{x}_j^{k+1} - x_j^k\right) \tag{2.14}$$

onde

$$\beta \in (0,1). \tag{2.15}$$

1. Critério de parada:

$$\left\| \boldsymbol{x}^{k+1} - \boldsymbol{x}^k \right\|_\infty < \varepsilon \tag{2.16}$$

onde $\| \boldsymbol{x} \|_\infty = \underset{1 \le i \le n}{\text{máximo}} \left\{ | x_i | \right\}$ para $\boldsymbol{x} \in \Re^n$.

Observações:

2.4. Cada iteração do algoritmo LPPAR consiste de n passos j, $j = 1, 2, \ldots, n$. Assim dado $\boldsymbol{x}^k = (x_1^k, x_2^k, \ldots, x_n^k)$ o próximo iterado é $\boldsymbol{x}^{k+1} = (x_1^{k+1}, x_2^{k+1}, \ldots, x_n^{k+1})$ onde para a determinação da componente x_j^{k+1} utilizamos a função de uma variável $\varphi_j\left(x_j, \boldsymbol{x}^k\right)$. O algoritmo é do tipo simultâneo pois as variáveis x_j estão separadas, logo com a utilização de um computador com processamento paralelo, todas as componentes x_j^{k+1} para $j = 1, 2, \ldots, n$ podem ser obtidas simultaneamente. Podemos ainda dizer que o algoritmo LPPAR é do tipo Jacobi.

2.5. A figura (2.2) ilustra geometricamente a determinação da raiz $\bar{x}_j^{k+1}$ de f(x) e do próximo iterado x_j^{k+1} do passo j da iteração k+1.

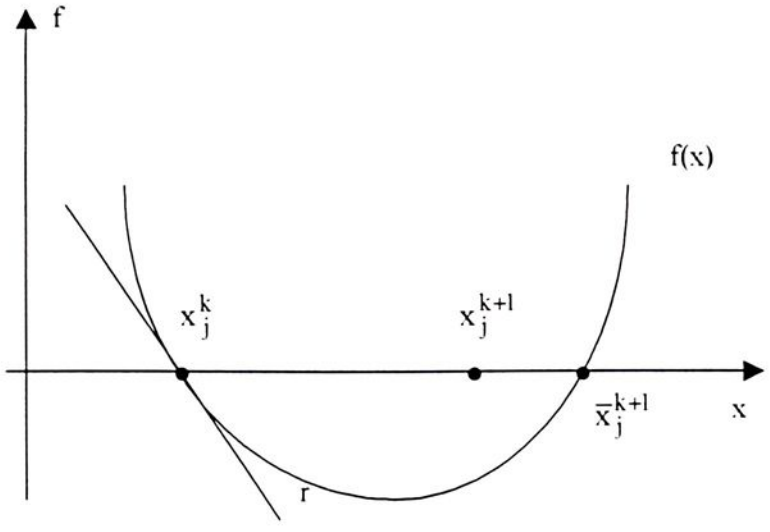


Figura (2.2): Determinação de x_j^{k+1}.

De (2.14), x_j^{k+1} é uma combinação convexa entre x_j^k e $\bar{x}_j^{k+1}$ assim, x_j^{k+1} é um ponto do segmento de reta determinado por x_j^k e $\bar{x}_j^{k+1}$.

Na figura (2.2), $d_j^k = \operatorname{tg}\theta$, onde θ é o ângulo formado pela reta r e o eixo das abscissas $\vec{0x}$.

Descrevemos a seguir, o método proposto por Dax-Berkowitz (1990), o qual será denotado por D-B e será utilizado nos testes numéricos.

Supondo o problema

$$\text{minimize } F(\boldsymbol{x}) = \| \boldsymbol{Ax} - \boldsymbol{b} \|_p \tag{2.17}$$

cada iteração do algoritmo D-B é composta de n passos. Para o j-ésimo passo $j = 1, 2, \ldots, n$, apenas x_j é modificada na tentativa de reduzir o valor da função objetivo, enquanto todas as outras variáveis permanecem fixadas. Denotando $\boldsymbol{x}^{k,j} \in \Re^n$, a estimativa atual para o início do j-ésimo passo da k-ésima iteração; $k = 1, 2, \ldots$

$$\boldsymbol{r}^{k,j} = \boldsymbol{Ax}^{k,j} - \boldsymbol{b} \tag{2.18}$$

o correspondente vetor residual, $\boldsymbol{c}_j$ a j-ésima coluna da matriz $\boldsymbol{A}$ e $\boldsymbol{e}_j$ a j-ésima coluna da matriz identidade então $\boldsymbol{x}^{k,j+1}$ é obtido de $\boldsymbol{x}^{k,j}$ pela regra

$$\boldsymbol{x}^{k,j+1} = \boldsymbol{x}^{k,j} + \lambda^{k,j}\boldsymbol{e}_j \tag{2.19}$$

onde $\lambda^{k,j}$ é computado por um algoritmo de busca linear para minimizar a função unidimensional

$$f(\lambda) = \left\| \boldsymbol{A}(\boldsymbol{x}^{k,j} + \lambda\boldsymbol{e}_j) - \boldsymbol{b} \right\|_p^p = \left\| \lambda\boldsymbol{c}_j + \boldsymbol{r}^{k,j} \right\|_p^p. \tag{2.20}$$

Seja N_j o subconjunto de $\{1, 2, \ldots, m\}$ que contém os índices das componentes não nulas de $\boldsymbol{c}_j$ e sejam α_i e β_i, $i = 1, \ldots, m$ as componentes de $\boldsymbol{c}_j$ e $\boldsymbol{r}^{k,j}$ respectivamente. Então (2.20) pode ser reescrita na forma

$$f(\lambda) = \sum_{i \in N_j} \left| \lambda\alpha_i + \beta_i \right|^p \tag{2.21}$$

O ponto $\lambda^{k,j}$ não é necessariamente o mínimo exato de $f(\lambda)$. Quatro casos são considerados:

(i) Se $p = 2$ então o ponto de mínimo de $f(\lambda)$ ocorre para o ponto

$$\hat{\lambda} = -\frac{\boldsymbol{c}_j^t \boldsymbol{r}^{k,j}}{\boldsymbol{c}_j^t \boldsymbol{c}_j}$$

assim $\lambda^{k,j} = \hat{\lambda}$, ou na versão com relaxação,

$$\lambda^{k,j} = \omega\hat{\lambda} \tag{2.22}$$

onde $0 < \omega < 2$.

(ii) Se $2 < p < \infty$ então $f(\lambda)$ é duas vezes continuamente diferencial e o análogo a (2.22) é

$$\tilde{\lambda} = \omega \hat{\lambda} \tag{2.23}$$

onde $\hat{\lambda} = \dfrac{-f'(0)}{f''(0)}$ é o tamanho do passo do método de Newton para minimizar $f(\lambda)$. O ponto $\lambda^{k,j}$ é escolhido como sendo o primeiro número na seqüência $\{\rho_\ell\}$, com $\rho_\ell = \left(\dfrac{1}{2}\right)^\ell \tilde{\lambda}$, $\ell = 0, 1, 2, \ldots$ o qual satisfaz $f'(\rho_\ell) \leq 0$. Neste caso

$$f'(\lambda) = p \sum_{i \in N_j} \alpha_i \left| \lambda \alpha_i + \beta_i \right|^{p-2} (\lambda \alpha_i + \beta_i) \tag{2.24}$$

e

$$f''(\lambda) = p(p-1) \sum_{i \in N_j} \alpha_i^2 \left| \lambda \alpha_i + \beta_i \right|^{p-2} . \tag{2.25}$$

(iii) Se $1 < p < 2$ então as derivadas segunda de $F(\boldsymbol{x})$ não são definidas para pontos onde o vetor residual $\boldsymbol{Ax - b}$ tem componentes nulas. Neste caso $F(\boldsymbol{x})$ é substituída pela função hiperbólica

$$H(\boldsymbol{x}) = \left\{ \sum_{i=1}^{m} \left[\left(\boldsymbol{a}_i^t \boldsymbol{x} - b_i \right)^2 + \varepsilon^2 \right]^{p/2} \right\}^{1/p} \tag{2.26}$$

onde $\boldsymbol{a}_i^t$ denota a i-ésima linha da matriz $\boldsymbol{A}$ e ε é uma constante pequena positiva. A função $H(\boldsymbol{x})$ é convexa, duas vezes continuamente diferenciável e o algoritmo pode ser agora aplicado.

Seja $\lambda^{k,j}$ o primeiro número na seqüência

$$\rho_\ell = \left(\frac{1}{2}\right)^\ell \omega \hat{\lambda}, \quad \ell = 0, 1, 2, \ldots .$$

o qual satisfaz $h'(\rho_f) \leq 0$.

onde

$$\hat{\lambda} = \frac{-h'(0)}{h''(0)} \tag{2.27}$$

e

$h(\lambda) = \sum_{i \in N_j} \left[(\alpha_i \lambda + \beta_i)^2 + \varepsilon^2 \right]^{p/2}$ substitui a função unidimensional $f(\lambda)$. Neste caso

$$h'(\lambda) = p \sum_{i \in N_j} \alpha_i (\alpha_i \lambda + \beta_i) \left[(\alpha_i \lambda + \beta_i)^2 + \varepsilon^2 \right]^{-\frac{(2-p)}{2}} \tag{2.28}$$

e

$$h''(\lambda) = p \sum_{i \in N_j} \gamma_i(\lambda) \alpha_i^2 \left[(\alpha_i \lambda + \beta_i)^2 + \varepsilon^2 \right]^{-\frac{(2-p)}{2}} \tag{2.29}$$

com

$$\gamma_i(\lambda) = 1 - (2-p)(\alpha_i \lambda + \beta_i)^2 / \left[(\alpha_i \lambda + \beta_i)^2 + \varepsilon^2 \right].$$

Apresentamos a seguir algumas propriedades gerais de convergência do algoritmo LPPAR. No que se segue, $\{ \boldsymbol{x}^k \}$ denota qualquer seqüência gerada por este algoritmo. A proposição a seguir estabelece que a função $F(\boldsymbol{x})$ é não crescente sobre a seqüência de iterados $\{ \boldsymbol{x}^k \}$.

Proposição 2.1. Para todo k, $k = 0, 1, 2, \ldots$ e todo j, $j = 1, 2, \ldots, n$ são válidas as relações:

(i) $F(\boldsymbol{x}^k) = \varphi(\boldsymbol{x}^k, \boldsymbol{x}^k)$

(ii) $\varphi_j(x_j^{k+1}, \boldsymbol{x}^k) \leq \varphi_j(x_j^k, \boldsymbol{x}^k)$

(iii) $\varphi(\boldsymbol{x}^{k+1}, \boldsymbol{x}^k) \leq \varphi(\boldsymbol{x}^k, \boldsymbol{x}^k)$

(iv) $F(\boldsymbol{x}^{k+1}) \leq F(\boldsymbol{x}^{k})$

Prova: (i) De (2.6) segue que

$$\begin{aligned}
\varphi\left(\boldsymbol{x}^{k}, \boldsymbol{x}^{k}\right) &= \sum_{j=1}^{n} \varphi_j\left(x_j^{k}, \boldsymbol{x}^{k}\right) \\
&= \sum_{j=1}^{n}\sum_{i=1}^{m} \lambda_j^{i} f_i\left[\frac{a_{ij}x_j}{\lambda_j^{i}} + d_j^{ik}\right] \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} \lambda_j^{i} f_i\left[<\boldsymbol{a}_i, \boldsymbol{x}^{k}> - b_i\right] \\
&= \sum_{i=1}^{m} f_i\left[<\boldsymbol{a}_i, \boldsymbol{x}^{k}> - b_i\right] \\
&= F(\boldsymbol{x}^{k})
\end{aligned}$$

onde a segunda igualdade ocorre de (2.5), a terceira de (2.4), a quarta segue de (2.2) e a última igualdade segue da definição da função objetivo; ou seja; de (2.1).

(ii) Para cada j, $j = 1, 2, \ldots, n$, de (2.14) e da convexidade de φ_j temos que

$$\begin{aligned}
\varphi_j\left(x_j^{k+1}, \boldsymbol{x}^{k}\right) &= \varphi_j\left(x_j^{k} + \beta\left(\overline{x}_j^{k+1} - x_j^{k}\right), \boldsymbol{x}^{k}\right) \\
&\leq \beta\varphi_j\left(\overline{x}_j^{k+1}, \boldsymbol{x}^{k}\right) + (1-\beta)\,\varphi_j\left(x_j^{k}, \boldsymbol{x}^{k}\right)
\end{aligned}$$

ou

$$\varphi_j\left(x_j^{k+1}, \boldsymbol{x}^{k}\right) - \varphi_j\left(x_j^{k}, \boldsymbol{x}^{k}\right) \leq \beta\left[\varphi_j\left(\overline{x}_j^{k+1}, \boldsymbol{x}^{k}\right) - \varphi_j\left(x_j^{k}, \boldsymbol{x}^{k}\right)\right] = 0 \qquad (2.30)$$

onde a última igualdade segue de (2.9). Assim de (2.30) temos o resultado desejado.

(iii) De (2.6) e (ii) temos que

$$\varphi\left(\boldsymbol{x}^{k+1}, \boldsymbol{x}^{k}\right) = \sum_{j=1}^{n} \varphi_j\left(x_j^{k+1}, \boldsymbol{x}^{k}\right) \leq \sum_{j=1}^{n} \varphi_j\left(x_j^{k}, \boldsymbol{x}^{k}\right) = \varphi\left(\boldsymbol{x}^{k}, \boldsymbol{x}^{k}\right)$$

(iv) $F(\boldsymbol{x}^{k+1}) \leq \varphi(\boldsymbol{x}^{k+1}, \boldsymbol{x}^{k}) \leq \varphi(\boldsymbol{x}^{k}, \boldsymbol{x}^{k}) = F(\boldsymbol{x}^{k})$

onde a primeira desigualdade ocorre de (2.7), a segunda de (iii) e a igualdade de (i).

□

Considerando o item (iv) da *proposição (2.1)*, se F tem os conjuntos de nível limitados; ou seja; $S_\alpha = \{ x : F(x) \leq \alpha \}$ é limitado para $\alpha \in \Re$, então $\{ x^k \} \subseteq S_\alpha$ com $\alpha = F(x_0)$ e portanto a seqüência será limitada. Este é o caso se F é definida por (1.1) e a matriz A tem posto completo, hipótese que será considerada de agora em diante. Assumiremos também que as funções φ_j são simétricas com respeito ao seu ponto de mínimo, semelhantemente as p-normas.

Por outro lado, se f_ℓ é duas vezes continuamente diferenciável e para todo ℓ

$$f_\ell''(x) \geq \mu > 0 \qquad (2.31)$$

provamos a proposição abaixo.

Proposição 2.2. $F(x^k) - F(x^{k+1}) \geq c \left\| x^{k+1} - x^k \right\|^2$ para alguma constante positiva c.

Prova: Se x é tal que $F(x) \leq F(x^k)$ então

$$F(x^k) - F(x) \geq \varphi(x^k, x^k) - \varphi(x, x^k) .$$

$$= \sum_{j=1}^{n} \left[\varphi_j'(x_j, x^k)(x_j^k - x_j) + \frac{1}{2}(x_j^k - x_j)^2 \varphi_j''(\tilde{x}_j^k, x^k) \right] \qquad (2.32)$$

onde a primeira desigualdade segue do item (i) da proposição (2.1) e de (2.7) enquanto que a igualdade segue da expansão de Taylor de φ em torno de x para algum $\tilde{x}^k$.

Consideremos agora cada φ_j separadamente e suponhamos que $x_j^{k+1} > x_j^k$ (o caso $x_j^{k+1} < x_j^k$ é tratado de maneira análoga).

Seja x_k^* o ponto de mínimo de φ_j. Dois casos devem ser considerados:

1º caso: $x_j^{k+1} \leq x_k^*$

Tomando $x_j = x_j^{k+1}$, o primeiro termo dentro do colchete em (2.32) é não negativo pois $\varphi_j'\left(x_j^{k+1}, x^k \right) \leq 0$ e $x_j^k - x_j^{k+1} < 0$, usando agora (2.31) temos que

$$\varphi_j'\left(x_j^{k+1}, x^k\right)\left(x_j^k - x_j^{k+1}\right) + \frac{1}{2}\left(x_j^k - x_j^{k+1}\right)^2 \varphi_j''\left(\tilde{x}_j^k, x^k\right) \geq \frac{\mu}{2}\left(x_j^{k+1} - x_j^k\right)^2 \qquad (2.33)$$

2º caso: $x_j^{k+1} > x_k^*$

Faça neste caso, x_j igual a $\bar{x}_j^k$, o ponto simétrico de x_j^{k+1} com respeito a x_k^*. Temos como no caso anterior, usando a simetria de φ_j que

$$\varphi_j\left(x_j^k, x^k\right) - \varphi_j\left(x_j^{k+1}, x^k\right) = \varphi_j\left(x_j^k, x^k\right) - \varphi_j\left(\bar{x}_j^k, x^k\right)$$

$$\geq \frac{\mu}{2}\left|\bar{x}_j^k - x_j^k\right|^2$$

$$\geq \frac{\mu}{2}\gamma^2\left|x_j^{k+1} - x_j^k\right|^2 \qquad (2.34)$$

onde a igualdade acima ocorre da definição de $\bar{x}_j^k$, a primeira desigualdade segue de (2.32) usando $x_j = \bar{x}_j^k$ e a segunda do fato que $\left|\bar{x}_j^k - x_j^k\right| \geq \gamma\left|x_j^{k+1} - x_j^k\right|$ para algum $0 < \gamma < 1$.

Tomando $c = \min\left\{\frac{\mu}{2}, \frac{\mu}{2}\gamma^2\right\}$, o resultado se verifica.

□

A figura (2.3.) ilustra geometricamente o 1º caso da demonstração da *proposição(2.2)*.

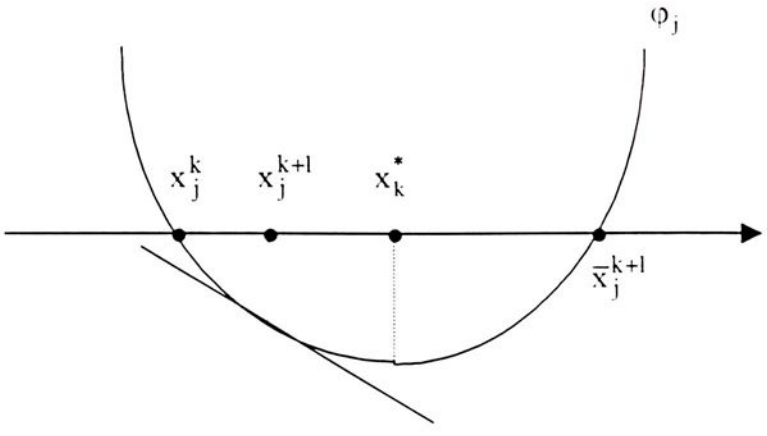


Figura (2.3): Neste caso $\varphi_j'\left(x_j^{k+1}, \boldsymbol{x}^k \right) \leq 0$ e $x_j^k - x_j^{k+1} < 0$.

O teorema a seguir estabelece a convergência do algoritmo LPPAR. Para a sua prova usaremos o seguinte resultado sobre funções convexas, o qual pode ser encontrado por exemplo em Roberts e Varbeg(1993) p. 99.

"Seja $f : U \rightarrow \Re$ uma função contínua e diferenciável sobre um conjunto convexo aberto $U \subseteq \Re^n$. Então f é convexa se e somente se f' é monótona crescente sobre U".

Necessitamos também da condição

$$\varphi_j'\left(x_j^k, \boldsymbol{x}^k \right) = -\varphi_j'\left(\bar{x}_j^{k+1}, \boldsymbol{x}^k \right) \tag{2.35}$$

a qual segue diretamente da convexidade e da simetria de φ_j com relação ao seu ponto de mínimo.

Teorema 2.1. Se F tem conjuntos de nível limitado e (2.31) se verifica, então cada subseqüência do algoritmo LPPAR definida por (2.14) converge para uma solução do problema (2.1).

Prova: Como F é limitada abaixo, $F(\boldsymbol{x}^{k+1}) - F(\boldsymbol{x}^{k}) \underset{k\to\infty}{\longrightarrow} 0$; portanto pela *proposição (2.2)* $\boldsymbol{x}^{k+1} - \boldsymbol{x}^{k} \underset{k\to\infty}{\longrightarrow} 0$. Assim,

$$\boldsymbol{x}^{k_\ell} \to \boldsymbol{x}^{*} \text{ implica que } \boldsymbol{x}^{k_\ell+1} \to \boldsymbol{x}^{*}. \tag{2.36}$$

Seja j um índice arbitrário, $j = 1, 2, \ldots, n$, suponhamos que $x_j^{k_\ell} < \overline{x}_j^{k_\ell+1}$ (o caso $x_j^{k_\ell} > \overline{x}_j^{k_\ell+1}$ é análogo), então de (2.14) e (2.15) temos que

$$x_j^{k_\ell} < x_j^{k_\ell+1} < \overline{x}_j^{k_\ell+1}. \tag{2.37}$$

Como φ_j é convexa então φ_j' é monótona crescente, assim de (2.37), para $\beta \in (0,1)$ vem

$$\varphi_j'\left(x_j^{k_\ell}, \boldsymbol{x}^{k_\ell}\right) \le \varphi_j'\left(x_j^{k_\ell+1}, \boldsymbol{x}^{k_\ell}\right)$$

ou

$$(1-\beta)\ \varphi_j'\left(x_j^{k_\ell}, \boldsymbol{x}^{k_\ell}\right) \le (1-\beta)\,\varphi_j'\left(x_j^{k_\ell+1}, \boldsymbol{x}^{k_\ell}\right) \tag{2.38}$$

e

$$\varphi_j'\left(x_j^{k_\ell+1}, \boldsymbol{x}^{k_\ell}\right) \le \varphi_j'\left(\overline{x}_j^{k_\ell+1}, \boldsymbol{x}^{k_\ell}\right)$$

ou

$$\beta\,\varphi_j'\left(x_j^{k_\ell+1}, \boldsymbol{x}^{k_\ell}\right) \le \beta\,\varphi_j'\left(\overline{x}_j^{k_\ell+1}, \boldsymbol{x}^{k_\ell}\right) \tag{2.39}$$

Adicionando (2.38) e (2.39) obtemos

$$\begin{aligned}(2\beta-1)\,\varphi_j'\left(x_j^{k_\ell+1}, \boldsymbol{x}^{k_\ell}\right) &\le (\beta-1)\,\varphi_j'\left(x_j^{k_\ell}, \boldsymbol{x}^{k_\ell}\right) + \beta\,\varphi_j'\left(\overline{x}_j^{k_\ell+1}, \boldsymbol{x}^{k_\ell}\right) \\ &= -\varphi_j'\left(x_j^{k_\ell}, \boldsymbol{x}^{k_\ell}\right)\end{aligned} \tag{2.40}$$

onde a igualdade acima ocorre de (2.35).

Temos assim de (2.40) que

$$\varphi_j'\left(x_j^{k_\ell}, \boldsymbol{x}^{k_\ell}\right) \leq (1-2\beta)\varphi_j'\left(x_j^{k_\ell+1}, \boldsymbol{x}^{k_\ell}\right) \tag{2.41}$$

Passando o limite em (2.41) quando $k_\ell \to \infty$, usando (2.36) e a continuidade de φ_j' obtemos

$$\varphi_j'\left(x_j^*, \boldsymbol{x}^*\right) \leq (1-2\beta)\varphi_j'\left(x_j^*, \boldsymbol{x}^*\right) \tag{2.42}$$

para qualquer $0 < \beta < 1$.

Logo a desigualdade (2.42) se verifica se e somente se

$$\varphi_j'\left(x_j^*, \boldsymbol{x}^*\right) = 0 \tag{2.43}$$

Portanto

$$0 = \varphi_j'\left(x_j^*, \boldsymbol{x}^*\right) = \sum_{i=1}^{m} a_{ij} f_i'\left(<\boldsymbol{a}_i, \boldsymbol{x}^*> - b_i\right) = \left(\nabla F\left(\boldsymbol{x}^*\right)\right)_j$$

ou seja; da convexidade de F temos que $\boldsymbol{x}^*$ é um mínimo do problema (2.1).

□

3. CONVERGÊNCIA PARA O CASO DIFERENCIÁVEL

3.1. O caso $2 \leq p < \infty$.

Com o objetivo de simplificar a notação nas demonstrações, vamos denotar, para cada j e k, x_j^k como x_k e $\varphi_j\left(x_j, \boldsymbol{x}^k\right)$ como $\varphi(x)$.

Assim para o problema (Γ.1) com $2 \leq p < \infty$ temos que

$$\varphi(x) = \sum_{i=1}^{m} \lambda^i \mid \alpha_i x + \beta_i \mid^p \tag{3.1}$$

onde $\alpha_i = \frac{a_{ij}}{\lambda^i}$, o qual estamos assumindo diferente de zero (caso contrário, o correspondente termo é constante e pode ser eliminado), $\lambda^i = \lambda^i_j$, $\beta_i = d^{ik}_j$ e

$$\varphi'(x) = p \sum_{i=1}^{m} \alpha_i \lambda^i \mid \alpha_i x + \beta_i \mid^{p-2} (\alpha_i x + \beta_i) \tag{3.2}$$

$$\varphi''(x) = p(p-1) \sum_{i=1}^{m} \lambda^i \alpha_i^2 \mid \alpha_i x + \beta_i \mid^{p-2}. \tag{3.3}$$

Neste caso, de (3.1) temos que $\varphi(x)$ é uma função convexa e simétrica com respeito ao seu ponto de mínimo.

Para as provas de alguns itens desta seção e também da seção seguinte, seguimos Dax e Berkowitz (1990).

Teorema 3.1. O método LPPAR está bem definido e cada ponto de acumulação da seqüência $\{x^k\}$ por ele gerada, resolve o problema (1.1).

Prova: Como a matriz ***A*** é de posto completo, de (iv) da *proposição (2.1)* segue que $\{x^k\}$ é limitada; ou seja; varia num conjunto compacto C, portanto da continuidade da derivada segunda e da convexidade resulta que

$$0 \le \varphi''(x) \le \gamma, \forall\, x \in C, \tag{3.4}$$

sendo γ uma constante positiva.

Se x_k^* é o ponto de mínimo de $\varphi(x)$, suponha que $x_k^* \ge x_k$ e portanto $\varphi'(x_k) \le 0$ (o caso $x_k^* < x_k$ é tratado e maneira análoga).

Defina a função quadrática

$$\psi(x) = \varphi(x_k) + \varphi'(x_k)(x - x_k) + \frac{\gamma}{2}(x - x_k)^2. \qquad (3.5)$$

Assim para $x_k \leq x \leq x_k^*$, de (3.4) e (3.5) segue que

$$\psi(x) \geq \varphi(x_k) + \varphi'(x_k)(x - x_k) + \frac{\varphi''(x_k)}{2}(x - x_k)^2 = \varphi(x)$$

ou seja;

$$\varphi(x) \leq \psi(x) \text{ para } x_k \leq x \leq x_k^*. \qquad (3.6)$$

Seja $y_k^* = x_k + \eta_k$ o ponto de mínimo de ψ, onde $\eta_k = \frac{-\varphi'(x_k)}{\gamma}$ então y_k^* satisfaz a desigualdade abaixo

$$y_k^* \leq x_k^*. \qquad (3.7)$$

Suponha por um momento que $y_k^* > x_k^*$, então expandindo $\varphi(x)$ em série de Taylor no ponto x_k e como x_k^* é o ponto de mínimo de $\varphi(x)$, temos que para $\xi \in (x_k, x_k^*)$, então

$$y_k^* = x_k - \frac{\varphi'(x_k)}{\gamma} > x_k^* = x_k - \frac{\varphi'(x_k)}{\varphi''(\xi)}$$

ou seja;

$$\frac{-\varphi'(x_k)}{\gamma} > \frac{-\varphi'(x_k)}{\varphi''(\xi)}. \qquad (3.8)$$

Agora de (3.8) e como $\varphi'(x_k) \leq 0$ temos que $\varphi''(\xi) > \gamma$, o que contradiz (3.4). Logo a desigualdade (3.7) se verifica.

Da definição do algoritmo, isto é de (2.9) e (2.14) temos que $\varphi(x_k) = \varphi(\bar{x}_{k+1})$ e $x_{k+1} \in (x_k + \bar{x}_{k+1})$.

Dois casos devem ser considerados:

(i) $x_{k+1} \leq x_k^*$

Neste caso, observando que $\eta_k \geq 0$ pois $\varphi'(x_k) \leq 0$ e $\gamma > 0$, temos que

$$x_k \leq y_k^* = x_k + \frac{\eta_k}{c_1} \leq x_{k+1} \quad . \tag{3.9}$$

para alguma constante $c_1 \geq 1$.

De (3.6), (3.9) e como $x_{k+1} \leq x_k^*$, da convexidade de φ segue que

$$\psi\left(x_k + \frac{\eta_k}{c_1}\right) \geq \varphi\left(x_k + \frac{\eta_k}{c_1}\right) \geq \varphi(x_{k+1}) \quad . \tag{3.10}$$

para $c_1 \geq 1$.

(ii) $x_{k+1} > x_k^*$

Sejam d_1 a distância entre os pontos y_k^* e x_k^* e d_2 a distância entre os pontos x_k^* e x_{k+1}.

Se $d_1 \geq d_2$, então da simetria de φ em relação a x_k^* temos que $\varphi(y_k^*) = \varphi(x_k + \eta_k) \geq \varphi(x_{k+1})$ e obviamente

$$\psi\left(x_k + \frac{\eta_k}{c_2}\right) \geq \varphi(x_{k+1}) \quad . \tag{3.11}$$

para alguma constante $c_2 \geq 1$, pois neste caso $x_k + \frac{\eta_k}{c_2} \leq x_k + \eta_k$ e ambos $x_k + \eta_k$ e $x_k + \frac{\eta_k}{c_2}$ estão a esquerda do ponto de mínimo x_k^*.

Se $d_1 < d_2$, seja x^*_{k+1} o ponto simétrico em relação a x^*_k. Assim $x^*_{k+1} < x_k + \eta_k = y^*_k$ e para alguma constante positiva $c_3 \geq 1$ temos que $x_k + \frac{\eta_k}{c_3} \leq x^*_{k+1}$; ou seja

$$\varphi\left(x_k + \frac{\eta_k}{c_3} \right) \geq \varphi\left(x^*_{k+1} \right) = \varphi(x_{k+1}) \quad . \tag{3.12}$$

para $c_3 \geq 1$.

De (3.10) – (3.12) e fazendo $c = \text{máx}\{ c_1, c_2, c_3 \}$ teremos que

$$\varphi\left(x_k + \frac{\eta_k}{c} \right) \geq \varphi(x_{k+1}) \tag{3.13}$$

para alguma constante $c \geq 1$.

De (3.6) e (3.13) vem

$$\begin{aligned} \varphi(x_k) - \varphi(x_{k+1}) &\geq \varphi(x_k) - \varphi\left(x_k + \frac{\eta_k}{c} \right) \\ &\geq \varphi(x_k) - \psi\left(x_k + \frac{\eta_k}{c} \right) \\ &= \psi(x_k) - \psi\left(x_k + \frac{\eta_k}{c} \right) \\ &= [\varphi'(x_k)]^2 \left(\frac{2c - 1}{2\gamma c^2} \right) \\ &\geq 0. \end{aligned} \tag{3.14}$$

Do item (ii) da *proposição (2.1)* e de (3.14) temos que $| \varphi'(x_k) |$ tende a zero para k tendendo a infinito.

Vamos provar agora que a diferença entre iterados consecutivos também tende a zero para $k \to \infty$.

De (2.14) temos que

$$
\begin{aligned}
x_{k+1} - x_k &= \beta\left(\bar{x}_{k+1} - x_k\right) \\
&= -\beta\,\bar{\alpha}\, d_k \\
&= -\beta\,\bar{\alpha}\, f'(x_k) \\
&= -\beta\,\bar{\alpha}\,\varphi'(x_k)
\end{aligned} \tag{3.15}
$$

onde a segunda igualdade ocorre de (2.13), a terceira de (2.11) e a quarta de (2.10).

Assim, como $\varphi'(x_k) \xrightarrow[k\to\infty]{} 0$, de (3.15) segue que $(x_{k+1} - x_k) \xrightarrow[k\to\infty]{} 0$.

Portanto se $\{x_{i_k}\}$ é uma subseqüência de $\{x_k\}$ tal que $\lim_{k\to\infty} x_{i_k+1} = x^*$, então $\lim_{k\to\infty} x_{i_k} = x^*$ desde que $\lim_{k\to\infty} (x_{k+1} - x_k) = 0$. Agora como $\varphi'(x_k) \xrightarrow[k\to\infty]{} 0$, obtemos por continuidade que

$$
0 = \varphi_j'\left(x_j^*, \boldsymbol{x}^*\right) = \left(\nabla F\left(\boldsymbol{x}^*\right)\right)_j \tag{3.16}
$$

e da convexidade de F temos que $\boldsymbol{x}^*$ é um mínimo do problema (1.1).

3.2. O caso $1 < p < 2$.

Neste caso, as derivadas segunda de $F(\boldsymbol{x})$ não são definidas para pontos onde o vetor residual $\boldsymbol{Ax} - \boldsymbol{b}$ possui componentes nulas, logo o *teorema (3.1)* não se aplica.

Como sugerido por Eyster, White e Wierwille (1973), ou Francis e White (1974) e também utilizado por Daz e Berkowitz (1990), vamos aproximar $F(\boldsymbol{x})$ pela função hiperbólica (HAP) da forma

$$
H(\boldsymbol{x}) = \sum_{i=1}^{m} \left[\left(\langle \boldsymbol{a}_i, \boldsymbol{x} \rangle - b_i \right)^2 + \varepsilon^2 \right]^{p/2} \tag{3.17}
$$

onde ε é uma constante positiva pequena.

A função HAP definida em (3.17) é convexa, duas vezes continuamente diferenciável e satisfaz

$$|F(x) - H(x)| \leq m^{1-p}\varepsilon \qquad (3.18)$$

para todo $x \in \Re^n$, ainda, HAP é estritamente convexa se e somente se A tem posto completo. (Veja Dax e Berkowitz (1990)).

Assim não existe dificuldade na aplicação do método LPPAR para minimizar $H(x)$.

De um outro lado, quando ε aproxima-se de zero, o problema de minimizar $H(x)$ pode tornar-se mal condicionado. Assim, na prática, nós usualmente começamos com um valor relativamente grande de ε e gradualmente o reduzimos. Uma questão ainda não resolvida, é se um ponto de mínimo de $H(x)$ é de fato próximo de um ponto de mínimo de $F(x)$.

A vantagem de usar $H(x)$ em vez de $F(x)$ é que para um $\varepsilon > 0$ fixado, $H(x)$ possui segundas derivada contínuas as quais são uniformemente limitadas em $\Re^n$. De fato, neste caso a função $\varphi(x)$ é definida por

$$\varphi(x) = \sum_{i=1}^{m} \lambda^i \left[(\alpha_i x + \beta_i)^2 + \varepsilon^2 \right]^{p/2} \qquad (3.19)$$

com α_i e β_i como definidos anteriormente. Igualmente

$$\varphi'(x) = p \sum_{i=1}^{m} \lambda^i \alpha_i (\alpha_i x + \beta_i) \left[(\alpha_i x + \beta_i)^2 + \varepsilon^2 \right]^{\frac{-(2-p)}{2}} \qquad (3.20)$$

e

$$\varphi''(x) = p \sum_{i=1}^{m} \lambda^i \gamma_i(x) \alpha_i^2 \left[(\alpha_i x + \beta_i)^2 + \varepsilon^2 \right]^{\frac{-(2-p)}{2}} \qquad (3.21)$$

onde

$$\gamma_i(x) = 1 - (2-p)(\alpha_i x + \beta_i)^2 / \left[(\alpha_i x + \beta_i)^2 + \varepsilon^2 \right] \qquad (3.22)$$

e temos a relação

$$p - 1 \leq \gamma_i(x) \leq 1. \tag{3.23}$$

Logo

$$\begin{aligned} 0 \leq \varphi''(x) &= p \sum_{i=1}^{m} \lambda^i \gamma_i(x) \alpha_i^2 \left[(\alpha_i x + \beta_i)^2 + \varepsilon^2 \right]^{\frac{-(2-p)}{2}} \\ &\leq p \sum_{i=1}^{m} \lambda^i \alpha_i^2 \left[(\alpha_i x + \beta_i)^2 + \varepsilon^2 \right]^{\frac{-(2-p)}{2}} \\ &\leq p \alpha \lambda \sum_{i=1}^{m} \left[(\alpha_i x + \beta_i)^2 + \varepsilon^2 \right]^{\frac{-(2-p)}{2}} \\ &\leq p \alpha \lambda m \left\{ \min_{1 \leq i \leq m} \left[(\alpha_i x + \beta_i)^2 + \varepsilon^2 \right] \right\}^{\frac{-(2-p)}{2}} \\ &= p \alpha \lambda m \varepsilon^{p-2} \end{aligned} \tag{3.24}$$

para todo $x \in \Re$, onde $\alpha = \max_{1 \leq i \leq m} \left\{ \alpha_i^2 \right\}$ e $\lambda = \max_{1 \leq i \leq m} \left\{ \lambda^i \right\}$.

Considerando (3.24), de maneira similar ao *teorema (3.1)* podemos demonstrar o seguinte teorema.

Teorema 3.2. O método LPPAR está bem definido. A seqüência $\left\{ H\left(x^k \right) \right\}$ é monótona decrescente e cada ponto de acumulação da seqüência $\left\{ x^k \right\}$ é um ponto de mínimo de $H(x)$. Além disso, se as colunas da matriz A são linearmente independentes, $H(x)$ tem um único ponto de mínimo e a seqüência $\left\{ x^k \right\}$ converge para este ponto.

4. RESULTADOS NUMÉRICOS

Apresentamos nesta seção, algumas experiências computacionais para o novo algoritmo LPPAR e o algoritmo de Dax-Berkowitz (D-B). Os algoritmos foram implementados em FORTRAN visual Workbench 1.0 da Microsoft (1993) e os testes realizados em um computador Pentium-S com CPU a 133 Mhz e 64 megabytes de memória RAM.

Antes de fornecermos os resultados numéricos, algumas considerações sobre a implementação computacional fazem-se necessárias. Os algoritmos LPPAR e D-B são facilmente implementáveis; entretanto como o algoritmo paralelo LPPAR, utiliza como um sub-produto o método de Newton e devido a natureza das funções envolvidas, alguns cuidados foram tomados durante a implementação deste algoritmo. Mais especificamente:

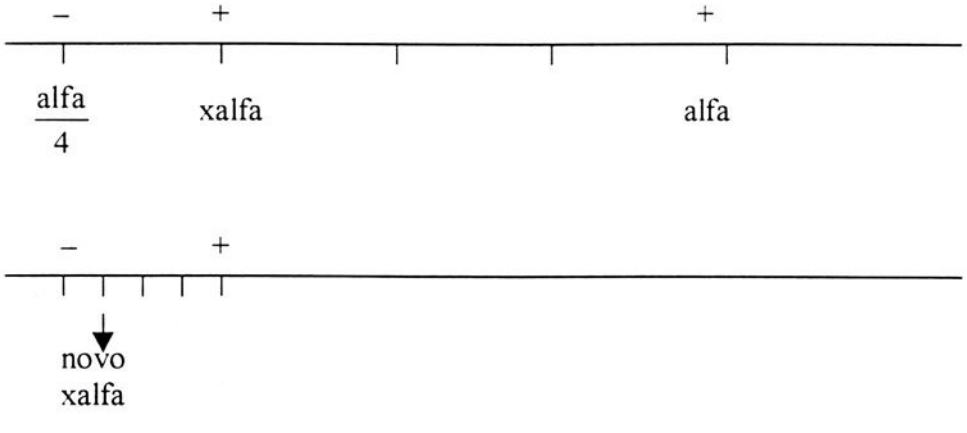
(i) como se sabe, o método de Newton pode não convergir, se a aproximação inicial não for suficientemente próxima da raiz da equação. Assim o alfa inicial, para cada passo $j\,(j=1,\ \ldots,n)$ de cada iteração para o cálculo da raiz da equação $g(\alpha)=0$ é obtido como:

faça $\text{alfa}=4^{\ell}$ para $\ell=0,1,2,\ldots$ até que para algum ℓ, $g(\text{alfa})>0$.

Supondo $g(\text{alfa})>0$, defina

$$\text{xalfa}=\text{alfa}-\frac{3}{4}\left(\text{alfa}-\frac{\text{alfa}}{4}\right) \qquad (4.1)$$

Se $g(\text{xalfa})<0$, escolha como aproximação inicial $\alpha_0=\text{alfa}$; se $g(\text{xalfa})>0$, calcule o novo xalfa, utilizando os pontos $\frac{\text{alfa}}{4}$ e xalfa. Veja o esquema abaixo para este último caso.



Observe que alfa $\in \{1, 4, 16, 64, \ldots\}$, $g\left(\frac{\text{alfa}}{4}\right) < 0$ e $g(\text{alfa}) > 0$. Assim o intervalo inicial considerado é $\left[\frac{\text{alfa}}{4},\ \text{alfa}\right]$ e a cada novo cálculo de xalfa, obtemos um novo intervalo com $\frac{1}{4}$ da amplitude do anterior.

O processo de escolha de xalfa, foi motivado pelo método da Bissecção. A utilização pura e simples do método da Bissecção mostrou-se ineficiente. Observe finalmente que a aproximação inicial α_0 está sempre a direita da raiz da equação $g(\alpha) = 0$.

(ii) denotando α_j^k, para $j = 1, \ldots, n$ e $k = 1, 2, \ldots$, a raiz da equação $g(\alpha) = 0$, no passo j da iteração k, então se $p = 2$, $\alpha_j^{k+1} = \alpha_j^k$ para qualquer $k = 0, 1, 2, \ldots$ e se $p \neq 2$ então $\alpha_j^{k+1} \cong \alpha_j^k$ para k suficientemente grande.

Assim para $p \neq 2$, utilizamos após a vigésima iteração, a raiz $\left(\alpha_j^k\right)^*$ obtida pelo método de Newton como a aproximação inicial para o passo j da iteração $k+1$.

Esse comportamento assintótico das raízes α_j foi observado através da experimentação numérica. Com isso, o número de iterações gastos pelo método de Newton em cada passo é muito pequeno (uma ou duas iterações já são suficientes).

(iii) o critério de parada utilizado no método de Newton (em cada passo j, $j = 1, \ldots, n$) para a determinação da raiz da equação $g(\alpha) = 0$ foi:

$$|\alpha_{k+1} - \alpha_k| < 10^{-10} \qquad \text{ou} \qquad |g(\alpha_{k+1})| < 10^{-10}.$$

A tolerância usada no método de Newton deve ser menor do que a tolerância utilizada no critério de parada (4.2). Caso contrário, a partir de uma determinada iteração, a função objetivo pode não descer; ou seja; $F\left(\boldsymbol{x}^{k+1}\right) > F\left(\boldsymbol{x}^k\right)$, para algum grande valor de k.

(iv) devido da natureza das funções envolvidas e dos coeficientes do sistema $\boldsymbol{Ax} = \boldsymbol{b}$, é comum a ocorrência de "overflow" para grandes valores de p. Para evitar que tal fato ocorra, pode-se considerar uma normalização dos dados do sistema linear, a qual consiste em dividir cada equação do sistema por M onde

$$M = \max \{ a_{ij} , b_i \}$$

para todo $i = 1, 2, \ldots, m$ e todo $j = 1, 2, \ldots, n$.

Para os algoritmos LPPAR e D-B utilizamos como solução inicial, o vetor $x^{(0)} = (0, 0, \ldots, 0) \in \Re^n$ e o critério de parada foi

$$\left\| x^{(k+1)} - x^{(k)} \right\|_\infty = \max_{1 \le i \le n} \left\{ \left| x_i^{(k+1)} - x_i^{(k)} \right| \right\} < TOL \qquad (4.2)$$

onde TOL representa a tolerância fixada a priori.

Para o algoritmo LPPAR utilizamos em todos os casos $\lambda_j^i = \frac{1}{\eta_i}$ onde η_i denota o número de elementos não nulos da i-ésima linha da matriz **A**.

O valor ótimo é indicado por VO calculado como

$$VO = \left\| Ax^* - b \right\|_p^p$$

onde x^* é a solução fornecida pelo respectivo algoritmo.

A comparação entre os algoritmos foi feita através do número de iterações gastas, o qual está indicado por ITER, pelo valor da função objetivo VO e pelo tempo de CPU, o qual é indicado por TEMPO e representado como

$$\min : \sec . 10^{-2} \sec .$$

Para problemas onde a dimensão da matriz **A** é pequena, não foi possível medir o tempo de CPU, tendo em vista que a menor unidade de tempo aqui considerada é a de centésimos de segundo. Os casos de maior interesse são aqueles onde **A** é de médio a grande porte e assim o tempo de CPU pode ser medido convenientemente.

Consideramos a seguir, alguns sistemas lineares onde a matriz **A** é de dimensão pequena. Quando o sistema é possível e determinado indicamos sua solução por x^*.

Exemplo 1. (Cheney (1982) – p. 44)

$$A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 2 \\ 2 & 4 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} \quad , \quad b = \begin{bmatrix} 3 \\ 1 \\ 7 \\ 11.1 \\ 6.9 \\ 7.2 \end{bmatrix}$$

Exemplo 2.

$$A = \begin{bmatrix} 4 & 2 & 1 \\ -1 & 2 & 0 \\ 2 & 1 & 4 \end{bmatrix} \quad , \quad b = \begin{bmatrix} 11 \\ 3 \\ 16 \end{bmatrix} \text{ onde } x^* = (1, 2, 3)^t.$$

Exemplo 3.

$$A = \begin{bmatrix} 3 & -4 & 1 \\ 1 & 2 & 2 \\ 4 & 0 & -3 \end{bmatrix} \quad , \quad b = \begin{bmatrix} 9 \\ 3 \\ -2 \end{bmatrix} \text{ onde } x^* = (1, -1, 2)^t.$$

Exemplo 4.

$$A = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 4 \end{bmatrix} \quad , \quad b = \begin{bmatrix} 3 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 2 \\ 3 \end{bmatrix}$$

onde $x^* = (1,1,1,1,1,1,1,1,1,1)^t$.

Dos quatro exemplos anteriormente citados, apenas o primeiro é um sistema linear inconsistente e deve ser resolvido por um método do tipo daqueles aqui considerados. Para os outros três exemplos os sistemas lineares são quadrados e possuem solução única, logo existem métodos mais adequados para a sua solução (como por exemplo o método de

eliminação de Gauss), estes estão colocados aqui, apenas como uma forma de verificar a exatidão da implementação dos respectivos algoritmos.

Seguindo a linha de experimentação de Dax e Berkowitz (1990), usamos também para os testes numéricos, sistemas lineares gerados aleatoriamente.

Exemplo 5. **A** é uma matriz esparsa m x n, onde cada linha a_i^t de A possui η_i elementos não nulos, η_i é um número aleatório do conjunto {1, 2, 3, 4, 5}, a localização destes elementos não nulos de a_i^t são números aleatórios do conjunto {1,2, ... , n} (A possui n colunas) e a_{ij} e b_i são números aleatórios do intervalo [-1, 1].

Todos os números aleatórios foram gerados através da rotina RANDOM (ranval), onde ranval é o número aleatório fornecido e pertencente ao intervalo [0, 1]. A rotina SEED (seedval) é utilizada para variar o ponto inicial dos números pseudo–aleatórios gerados. Se seedval = -1, então a seqüência de valores de RANDOM será sempre diferente.

Os resultados fornecidos pelos algoritmos LPPAR e D-B encontram-se dispostos nas tabelas 5 e 6.

As tabelas a seguir fornecem os resultados das experiências computacionais realizadas. Nestas, w indica o parâmetro de relaxação para o respectivo algoritmo; para o algoritmo D-B $w = \omega$ e para o algoritmo LPPAR $w = \beta$.

Tabela 1. Algoritmos LPPAR e D-B para o exemplo 1 com $TOL = 10^{-7}$ e diferentes valores de p.

p	w	ALG.	SOLUÇÃO	VO	ITER
2.0	0.9	LPPAR	(2.0741 , 1 . 8078)	4.6912	37
	1.0	D-B	(2.0741 , 1 . 8078)	4.6912	23
4.0	0.9	LPPAR	(2.0466 , 1 . 9207)	4.3619	37
	1.0	D-B	(2.0466 , 1 . 9207)	4.3619	26
6.0	0.9	LPPAR	(2.0390 , 1 . 9498)	4.0201	36
	1.0	D-B	(2.0390 , 1 . 9498)	4.0201	31

Os valores fornecidos em Cheney (1982) são os seguintes:

	p = 2	p = 4	p = 6
x_1	2.0741	2.0466	2.0390
x_2	1.8078	1.9207	1.9498
VO	4.6911	4.3623	4.0206

Tabela 2. Algoritmos LPPAR e D-B para o exemplo 2 com $TOL = 10^{-7}$ e diferentes valores de p.

p	w	ALG.	SOLUÇÃO	VO	ITER
2.0	0.9	LPPAR	(1 , 2 , 3)	.1654D-11	62
	1.0	D-B	(1 , 2 , 3)	.1567D-13	22
5.0	0.9	LPPAR	(1.014657 , 1.1.979193 , 3.044134)	.2184D-10	26
	1.0	D-B	(1 , 2 , 3)	.2389D-29	101
10.0	0.9	LPPAR	(0.993297 , 1.934226 , 2.991303)	.1821D-17	24
	1.0	D-B	(1 , 2 , 2.999999)	.8478D-55	232

Para a aplicação do algoritmo LPPAR neste exemplo, usamos os dados na forma normalizada, ou seja, dividimos cada equação do sistema linear por 16.

Tabela 3. Algoritmos LPPAR e D-B para o exemplo 3 com $TOL = 10^{-7}$ e diferentes valores de p.

p	w	ALG.	SOLUÇÃO	VO	ITER
2.0	0.9	LPPAR	(0.999999 , -1, 2)	.8435D-12	41
	1.0	D-B	(1 , -1 , 2)	.1051D-13	15
5.0	0.9	LPPAR	(0.999984 , -1.000015 , 1.999992)	.1000D-20	23
	1.0	D-B	(0.999999 , -1 , 2)	.2357D-30	63
10.0	0.9	LPPAR	(1.000600 , -1.000152 , 1.999603)	.3567D-24	17
	1.0	D-B	(1.000001 , -0.999999 , 2)	.3300D-55	144

Tabela 4. Algoritmos LPPAR e D-B para o exemplo 4 com $TOL = 10^{-7}$ e diferentes valores de p.

P	w	ALG.	SOLUÇÃO	VO	ITER	TEMPO
1.5	0.99	LPPAR	(0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999)	.3331D-08	66	0:00.16
	0.9	D-B	(1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1 , 1)	.5635D-10	34	0:00.05
2.0	0.99	LPPAR	(0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999)	.6216D-11	84	0:00.23
	0.9	D-B	(0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 1 , 1 , 1 , 1)	.2838D-12	39	0:00.06
5.0	0.99	LPPAR	(0.999999, 0.999999, 0.999998, 0.999998, 0.999998, 0.999998, 0.999998, 0.999998, 0.999998, 0.999999)	.3848D-26	251	0:00.44
	0.9	D-B	(0.999998, 0.999997, 0.999996, 0.999996, 0.999996, 0.999996, 0.999996, 0.999996, 0.999996, 0.999998)	.5631D-27	181	0:00.22
10.0	0.99	LPPAR	(0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999, 0.999999)	.7210D-50	514	0:00.82
	0.9	D-B	(0.999998, 0.999997, 0.999997, 0.999997, 0.999997, 0.999997, 0.999997, 0.999997, 0.999998, 0.999998)	.4078D-51	402	0:00.38

Para todas as tabelas a seguir usamos $w = 0.99$ no algoritmo LPPAR e $w = 1.0$ no algoritmo D-B.

Tabela 5. Algoritmos LPPAR e D-B para o exemplo 5 com $TOL = 10^{-5}$, $m = 100$, $n = 10$ e diferentes valores de p.

	LPPAR			D-B		
P	ITER	VO	TEMPO	ITER	VO	TEMPO
2.0	16	.2211D+02	0:00.16	7	.2211D+02	0:00.05
5.0	25	.7545D+01	0:00.83	11	.7545D+01	0:00.11
10.0	33	.2487D+01	0:00.88	18	.2487D+01	0:00.22
20.0	80	.4810D+00	0:01.54	25	.4810D+00	0:00.24
50.0	368	.1227D-01	0:04.89	30	.1227D-01	0:00.27
100.0	379	.7045D-04	0:04.94	59	.7045D-04	0:00.54

Tabela 6. Algoritmos LPPAR e D-B para o exemplo 5 com $TOL = 10^{-5}$, $m = 500$, $n = 50$ e diferentes valores de p.

	LPPAR			D-B		
P	ITER	VO	TEMPO	ITER	VO	TEMPO
2.0	27	.1423D+03	0:18.12	9	.1423D+03	0:07.64
5.0	38	.6036D+02	0:31.42	12	.6036D+02	0:10.10
10.0	60	.2539D+02	0:47.29	20	.2539D+02	0:16.86
20.0	107	.7860D+01	1:21.18	28	.7860D+01	0:23.51
50.0	216	.6564D+00	2:38.79	67	.6564D+00	0:55.96
100.0	403	.2946D-01	4:48.97	151	.2946D-01	2:07.92

5. CONCLUSÕES FINAIS

De uma maneira geral, algoritmos seqüências, como é o caso do algoritmo de Dax-Berkowit (1990) tem convergência mais rápida do que algoritmos simultâneos, como é o caso do algoritmo LPPAR aqui proposto. Agora métodos simultâneos são convenientes para a utilização em computadores com arquitetura paralela. Se um método simultâneo, gasta em um computador convencional (01 processador central), um tempo de CPU $t = t_0$, para resolver um sistema linear $\boldsymbol{Ax} = \boldsymbol{b}$, com $\boldsymbol{A}$ possuindo m linhas, então o tempo gasto para resolver esse mesmo problema em um computador com m processadores paralelos é da ordem de $0(t) = \frac{t_0}{m}$, existe também um tempo de sincronização/comunicação entre os processadores.

Como já mencionado, as soluções dos sistemas lineares apresentados nos exemplos 2, 3 e 4 devem ser obtidas por métodos mais adequados e convenientes para este tipo de problema. Apenas a título de ilustração, observamos que as soluções fornecidas pelos algoritmos LPPAR e D-B são da mesma ordem de grandeza e que o algoritmo LPPAR é superior ao de D-B quando da utilização de uma máquina com arquitetura paralela.

Para o exemplo 1, os resultados fornecidos pelos algoritmos LPPAR e D-B são praticamente idênticos aqueles apresentados em Cheney (1982).

No caso do exemplo 5, o mais interessante desta seção, os valores da função objetivo fornecidos pelos dois métodos são exatamente iguais e o tempo de CPU gasto pelo algoritmo LPPAR seria significativamente menor do que o algoritmo D-B quando da utilização de uma máquina com arquitetura paralela. Assim neste caso o algoritmo LPPAR mostrou-se superior ao algoritmo D-B. Para este exemplo, a utilização de um menor valor para a tolerância, não produz ganhos significativos para a ordem de grandeza de VO.

6. BIBLIOGRAFIA

[01] CHENEY, E. W., " *Introduction to Approximation Theory*", Chelsea Publishing Company, New York, N.Y., (1982).

[02] DAX, A. and BERKOWITZ,B., " *Column Relaxation Methods for Least Norm Problems*", SIAM Journal Science Statistics Computation, Vol. 11, n 5, p. 975-989, (1990).

[03] EYSTER, J. W., WHITE,J. A. and WIERWILLE,W. W., " *On solving Multifacility Location Problems using a Hiperboloid Aproximation Procedure*", AIIE Trans., Vol, 5.p. 1-6, (1973).

[04] FRANCIS, R. L. and WHITHE, J. A ., " *Facility Layout and Location, an Analytical Approch*", Prentice –Hall, Englewood Cliffs, N.J. (1974).

[05] LOPES. J. M., " *Métodos Iterativos para Problemas de Complementariedade Linear e de Norma mínima*", Tese de doutorado, Pontificia Universidade Católica do Rio de Janeiro, (1992).

[06] ROBERTS, A . W. and VARBERG D. E. " *Convex Functions*", Academic Press, New York, (1973).

7. APÊNDICE

Apresentamos neste apêndice os programas elaborados em FORTRAN para os testes numéricos.

```
      DIMENSION A(100,50),B(100),X(50),R(100)
      COMMON A,B,X,R,M,N
      OPEN(2,FILE="DADOS",STATUS="OLD")
C
C=============================================================
C
C       ESTE PROGRAMA DETERMINA O ESTIMADOR L2 PELO METODO DE
C       DAX-BERKOWITZ.
C
C       VARIAVEIS UTILIZADAS
C
C     A=MATRIZ DO SISTEMA LINEAR AX=B
C     B=VETOR INDEPENDENTE
C     X=VETOR SOLUCAO
C     R=VETOR RESIDUAL
C     M=NUMERO DE LINHAS DA MATRIZ A
C     N=NUMERO DE COLUNAS DA MATRIZ A
C     VO=VALOR DA FUNCAO OBJETIVO PARA A SOLUCAO OTIMA
C     KMAX=NUMERO MAXIMO DE ITERACOES PERMITIDAS
C     TOL=TOLERANCIA UTILIZADA
C     IND=INDICADOR DE CONVERGENCIA
C
C       SUB-ROTINA
```

```
C
C    RESIDUO=CALCULO DO VETOR RESIDUAL
C
C=============================================================
C
C       LEITURA DOS DADOS
C
      WRITE(*,*)' METODO DE DAX BERKOWITZ COM P=2'
      WRITE(*,*)' FORNECA O PARAMETRO DE RELAXACAO W'
      READ(*,*)W
      TOL=1.0E-07
      KMAX=3000
      READ(2,*)M,N
      DO 5 I=1,M
    5 READ(2,*)(A(I,J),J=1,N),B(I)
C
C       IMPRESSAO DOS DADOS
C
      WRITE(*,10)M,N,TOL,KMAX,W
   10 FORMAT(1X,'METODO DE DAX-BERKOWITZ COM P=1',/,1X,' M=',I4,'
       N=',I4
     *,' TOL= ',E14.7,' KMAX=',I4,' W=',E14.7)
      DO 8 I=1,M
    8 WRITE(*,3)(A(I,J),J=1,N),B(I)
    3 FORMAT(1X,10F8.2)
C
C       SOLUCAO INICIAL E LEITURA DO RELOGIO DO SISTEMA
C
      CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(*,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      DO 500 ICONT=1,100
      DO 20 I=1,N
      X(I)=1.0
   20 CONTINUE
C
C       ITERACAO PRINCIPAL
C
      DO 90 K=1,KMAX
      IND=0
      DO 80 J=1,N
      CALL RESIDUO
      L=0
      S1=0.0
      S2=0.0
      DO 40 I=1,M
      IF(A(I,J))30,40,30
   30 L=1
      S1=S1+A(I,J)*R(I)
      S2=S2+A(I,J)**2
   40 CONTINUE
      IF(L-1)80,50,80
```

```
 50 XLAMBDA=-W*(S1/S2)
    AUX=X(J)+XLAMBDA
    IF(ABS(AUX-X(J))-TOL)70,70,60
 60 IND=1
 70 X(J)=AUX
 80 CONTINUE
C
C      VERIFICACAO DA CONVERGENCIA
C
    IF(IND)90,110,90
 90 CONTINUE
    WRITE(*,100)KMAX,TOL
100 FORMAT(1X,' O ALGORITMO NAO CONVERGIU APOS',I4,' ITERACOES
    *COM TOLERANCIA',F14.7)
    GO TO 160
C
C      CALCULO DO VALOR OTIMO
C
110 NITER=K
500 CONTINUE
     CALL GETTIM(ihr,imin,isec,i100th)
     WRITE(*,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100th
     VO=0.0
     DO 130 I=1,M
     AUX=0.0
     DO 120 J=1,N
     AUX=AUX+A(I,J)*X(J)
120 CONTINUE
     VO=VO+ABS(AUX-B(I))**2
130 CONTINUE
C
C      IMPRESSAO DOS RESULTADOS
C
    WRITE(*,140)NITER,VO
140 FORMAT(/,1X,' NUMERO DE ITERACOES =',I4,' VALOR OTIMO=',E14.7)
     WRITE(*,150)(X(I),I=1,N)
150 FORMAT(/,8F10.4)
160 CLOSE(2)
     STOP
    END
C
C      SUB-ROTINA
C
    SUBROUTINE RESIDUO
    DIMENSION A(100,50),B(100),X(50),R(100)
    COMMON A,B,X,R,M,N
    DO 20 I=1,M
    R(I)=0.0
    DO 10 J=1,N
    R(I)=R(I)+A(I,J)*X(J)
 10 CONTINUE
    R(I)=R(I)-B(I)
```


```
 20 CONTINUE
    RETURN
    END


     DIMENSION A(1000,100),B(1000),X(100),R(1000),NI(1000)
     DOUBLE PRECISION A,B,X,R,VOP,AUX,TOL,DER1,DER2,XLAMBDA,
    *DER1N,W,BAU,X.A1
     OPEN(unit=8,file='prn')
     OPEN(2,FILE='DADOS.TXT',STATUS='OLD')
C
C ==============================================================
C
C    ESTE PROGRAMA DETERMINA O ESTIMADOR LP PELO METODO DE  C
DAX-BERKOWITZ
C
C       VARIAVEIS UTILIZADAS
C
C    A=MATRIZ DO SISTEMA LINEAR AX=B
C    B=VETOR INDEPENDENTE
C    X=VETOR SOLUCAO
C    R=VETOR RESIDUAL
C    M=NUMERO DE LINHAS DA MATRIZ A
C    N=NUMERO DE COLUNAS DA MATRIZ A
C    VO=VALOR DA FUNCAO OBJETIVO PARA A SOLUCAO OTIMA
C    KMAX=NUMERO MAXIMO DE ITERACOES PERMITIDAS
C    TOL=TOLERANCIA UTILIZADA
C    IND=INDICADOR DE CONVERGENCIA
C    W=PARAMETRO DE RELAXACAO
C
C==============================================================
C
C       LEITURA DOS DADOS
C
     TOL=1.0E-07
     KMAX=1000
     READ(2,*)M,N
     DO 3 I=1,M
   3 READ(2,*)(A(I,J),J=1,N),B(I),NI(I)

C     READ(2,*)M,N,A,B,NI
     WRITE(*,*)'ESTIMACAO NO ESPACO LP-METODO DE DAX-BERKOWITZ'
     WRITE(*,*)' FORNECA OS VALORES DE P E DE W '
     READ(*,*)P,W
C
C       IMPRESSAO DOS DADOS
C
     WRITE(8,*)' METODO DE DAX-BERKOWITZ-ESTIMACAO NO ESPACO
    *LP '
     WRITE(8,5)M,N,TOL,KMAX,P,W
   5 FORMAT(1X,' M=',I3,' N=',I3,' TOL=',D13.7,' KMAX=',I5,2X,' P=',F6.
    *2,2X,' W=',E14.7)
```


```
      WRITE(8,*)' MATRIZ  A'
      DO 550 I=1,M
  550 WRITE(8,560)(A(I,J),J=1,N)
  560 FORMAT(/,1X,5D14.7)
      WRITE(8,*)' VETOR B'
      WRITE(8,560)(B(I),I=1,N)
      WRITE(8,565)
  565 FORMAT(/,' VETOR NI')
      WRITE(8,570)(NI(I),I=1,M)
  570 FORMAT(/,1X,10I5)

C
C       SOLUCAO INICIAL E LEITURA DO RELOGIO DO SITEMA
C
      CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(8,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      WRITE(*,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      DO 7 I=1,N
      X(I)=0.0
    7 CONTINUE
C       ITERACAO PRINCIPAL
C
      DO 140 K=1,KMAX
      IND=0
      DO 130 J=1,N
      DO 20 I=1,M
      R(I)=0.0
      DO 10 JK=1,N
      R(I)=R(I)+A(I,JK)*X(JK)
   10 CONTINUE
      R(I)=R(I)-B(I)
   20 CONTINUE
C
C       CALCULO DO PASSO LAMBDA
C
      IC=0
      DER1=0.0
      DER2=0.0
      DO 40 I=1,M
      IF(A(I,J))30,40,30
   30 IC=IC+1
      AUX=DABS(R(I))**(P-2)
      DER1=DER1+AUX*A(I,J)*R(I)
      DER2=DER2+AUX*A(I,J)**2
   40 CONTINUE
      DER2=DER2*(P-1.0)
      XLAMBDA=-1.0*W*(DER1/DER2)
      IF(DER2)45,120,45
   45 IF(IC)50,60,50
   50 IF(DER1)110,110,70
```


```
 60 XLAMBDA=0.0
    GO TO 110
C
C       VERIFICACAO DO TAMANHO DO PASSO
C
 70 L=0
 75 DER1N=0.0
    DO 90 I=1,M
    IF(A(I,J))80,90,80
 80 BAUX=XLAMBDA*A(I,J)+R(I)
    A1=DABS(BAUX)**(P-2.0)
    DER1N=DER1N+A(I,J)*BAUX*A1
 90 CONTINUE
    DER1N=P*DER1N
    IF(DER1N)110,110,95
 95 IF(DER1N*XLAMBDA-TOL)110,110,100
100 L=L+1
     XLAMBDA=XLAMBDA/(2.0**L)
     GO TO 75
110 AUX=X(J)+XLAMBDA
     IF(DABS(AUX-X(J))-TOL)120,120,115
 115 IND=1
 120 X(J)=AUX
 130 CONTINUE
      IF(IND)140,160,140
 140 CONTINUE
     WRITE(*,150)KMAX,TOL
 150 FORMAT(/,' O ALGORITMO NAO CONVERGIU APOS',I5,' ITERACOES
    *COM TOLERANCIA',F14.7)
     GO TO 200
C
C       CALCULO DO VALOR OTIMO
C
 160 VOP=0.0
      DO 180 I=1,M
      AUX=0.0
      DO 170 J=1,N
      AUX=AUX+A(I,J)*X(J)
 170 CONTINUE
      VOP=VOP+DABS(AUX-B(I))**P
 180 CONTINUE
      CALL GETTIM (ihr,imin,isec,i100th)
      WRITE(8,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      WRITE(*,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      WRITE(8,190)K,VOP
      WRITE(*,190)K,VOP
 190 FORMAT(/,' SOLUCAO APOS',I5,' ITERACOES COM VALOR OTIMO=', *D14.7)
      WRITE(8,560)(X(I),I=1,N)
      WRITE(*,560)(X(I),I=1,N)
 200 CLOSE(2)
```

```
      STOP
      END


     DIMENSION A(1000,100),B(1000),NI(1000)
     DOUBLE PRECISION A,B
     OPEN(unit=8,file='prn')
     OPEN(2,FILE='K2DADOS',STATUS='NEW')
C
C=============================================================
C
C       ESTE PROGRAMA GERA UM SISTEMA LINEAR AX=B COM UMA
C       MATRIZ A ESPARSA ONDE A(I,J) E  B(I) SAO NUMEROS ALEATORIOS C
DO INTERVALO [-1,1].CADA LINHA DE A POSSUI NI ELEMENTOS NAO C
NULOS ONDE NI E UM NUMERO ALEATORIO PERTENCENTE AO        C
CONJUNTO {1,2,3,4,5}
C
C               VARIAVEIS UTILIZADAS
C
C       A=MATRIZ DE RESTRICOES
C       M=NUMERO DE LINHAS DA MATRIZ A
C       N=NUMERO DE COLUNAS DA MATRIZ A
C       B=VETOR INDEPENDENTE
C       NI=NUMERO DE ELEMENTOS NAO NULOS DA LINHA I
C
C               SUBROTINA UTILIZADA
C
C     RANDOM(ranval) GERA NUMEROS ALEATORIOS NO INTERVALO           C
[0.0,1.0)
C     SEED(-1) SEMENTE PARA A SUBROTINA RANDOM
C
C=============================================================
C
      WRITE(*,*)'FORNECA OS VALORES DE M E N'
      READ(*,*)M,N
C
C       GERACAO DA MATRIZ A
C
      CALL SEED(-1)
      DO 10 I=1,M
      DO 10 J=1,N
   10 A(I,J)=0.0
      DO 40 I=1,M
   20 CALL RANDOM(ranval)
      NI(I)=10.0*ranval
      IF(NI(I))25,20,25
   25 IF(NI(I)-5)30,30,20
   30 DO 40 I1=1,NI(I)
      CALL RANDOM(ranval)
      J=IFIX(N*(ranval+1.0/FLOAT(N)))
      CALL RANDOM(ranval)
      A(I,J)=2*(ranval-0.5)
```

```
  40 CONTINUE
C
C       GERACAO DO VETOR B
C
      DO 50 I=1,M
      CALL RANDOM(ranval)
      B(I)=2.0*(ranval-0.5)
  50 CONTINUE
      DO 120 I=1,M
      NI(I)=0
      DO 110 J=1,N
      IF(A(I,J))105,110,105
 105 NI(I)=NI(I)+1
 110 CONTINUE
 120 CONTINUE
      WRITE(2,*)M,N,A,B,NI
      CLOSE(2)
      END


      DIMENSION A(1000,100),B(1000),NI(1000),D(1000),X(100),XNOVO(100),A
     *LFAX(100)
      COMMON A,B,NI,X,D,M,N,J
      DOUBLE PRECISION V,F,DERF,ALFA,ALFAN,TOL,AUX,BAUX,A1,
     *A2,FIJ,DIR,XF,X,XNOVO,A,B,D,XBARRA,BETA,DIFER,XALFA,
     *AMPLIT,AALFA
      OPEN(unit=8,file='prn')
      OPEN(2,FILE='DADOS.TXT',STATUS='OLD')
C
C=============================================================
C       ESTE PROGRAMA RESOLVE UM PROLEMA DE MINIMIZACAO DE
C       NORMA NO ESPACO LP – ALGORITMO LPPAR
C
C       VARIAVEIS UTILIZADAS
C
C     A=MATRIZ DO SISTEMA LINEAR AX=B
C     B=VETOR INDEPENDENTE
C     M=NUMERO DE LINHAS DA MATRIZ A
C     N=NUMERO DE COLUNAS DA MATRIZ A
C     NI=NUMERO DE ELEMENTOS NAO NULOS DE CADA LINHA DA MATRIZ A
C     X=VETOR SOLUCAO
C     VO=VALOR DA FUNCAO OBJETIVO PARA A SOLUCAO OTIMA
C     DER1=DERIVADA PRIMEIRA
C     DER2=DERIVADA SEGUNDA
C     KMAX=NUMERO MAXIMO DE ITERACOES PERMITIDAS
C     TOL=TOLERANCIA UTILIZADA NO CRITERIO DE PARADA
C     BETA=TAMANHO DO PASSO
C     IBAT=NUMERO DE ITERACOES USADAS PELO MEWTODO DE NEWTON
C
C       SUB-ROTINAS UTILIZADAS
C
C       DJIK=CALCULO DO VETOR D(J,IK)
```

```
C
C=============================================================
C
C       LEITURA DOS DADOS
C
      TOL=1.0D-7
      KMAX=1000
C     READ(2,*)M,N,A,B,NI
      READ(2,*)M,N
      DO 5 I=1,M
    5 READ(2,*)(A(I,J),J=1,N),B(I),NI(I)
      WRITE(*,*)' ESTIMACAO NO ESPACO LP - FORNECA O VALOR DE P E
     *BETA'
      WRITE(*,*)' ONDE P > ou = a 1 E BETA PERTENCE AO INTERVALO[0,1]'
      WRITE(8,255)
  255 FORMAT(///,' ESTIMACAO NO ESPACO LP - METODO NOVO')
      READ(*,*)P,BETA
      WRITE(8,9)M,N,TOL,KMAX,P,BETA
    9 FORMAT(/,' M=',I3,' N=',I3,' TOL=',D14.7,' KMAX=',I6,' P=',
     *F8.2,'   BETA=',F6.3)
C
C       NORMALIZACAO DOS DADOS
C
      DO 122 I=1,M
      DO 121 J=1,N
      A(I,J)=A(I,J)/16.0
  121 CONTINUE
      B(I)=B(I)/16.0
  122 CONTINUE

      WRITE(8,*)' MATRIZ  A'
      DO 550 I=1,M
  550 WRITE(8,560)(A(I,J),J=1,N)
  560 FORMAT(/,1X,5D14.7)
      WRITE(8,*)' VETOR B'
      WRITE(8,561)(B(I),I=1,M)
  561 FORMAT(/,1X,5D14.7)
      WRITE(8,565)
  565 FORMAT(/,' VETOR NI')
      WRITE(8,570)(NI(I),I=1,M)
  570 FORMAT(/,1X,10I5)
C
C       LEITURA DO RELOGIO DO SISTEMA
C
      CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(8,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
      WRITE(*,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
C
C       SOLUCAO INICIAL
C
```


```
      DO 10 I=1,N
      X(I)=0.0
   10 CONTINUE
C
C       ITERACAO PRINCIPAL
C
      DO 150 K=1,KMAX
      IND=0
      DO 110 J=1,N
      CALL DJIK
C
C       CALCULO DA DIRECAO DIR=F'(X(K))
C
      FIJ=0.0
      DIR=0.0
      DO 30 I=1,M
      IF(A(I,J))20,30,20
   20 AUX=A(I,J)*X(J)*NI(I)+D(I)
      BAUX=ABS(AUX)
      A2=BAUX**P
      A1=BAUX**(P-2.0)
      DIR=DIR+A(I,J)*AUX*A1
      FIJ=FIJ+(1.0/NI(I))*A2
   30 CONTINUE
      DIR=DIR*P
C
C       ESCOLHA DE ALFA INICIAL TAL QUE F SEJA MAIOR OU IGUAL A
C       ZERO  ( PONTO MEDIO DOS DOIS ULTIMOS VALORES DE ALFA )
C
      IF(K-20)35,35,37
   37 ALFA=ALFAX(J)
      GO TO 89
   35 L=0
      AALFA=4.0**L
      ALFA=AALFA
   40 F=0.0
      DO 60 I=1,M
      IF(A(I,J))50,60,50
   50 AUX=A(I,J)*(X(J)-ALFA*DIR)*NI(I)+D(I)
      BAUX=DABS(AUX)
      A1=(1.0/NI(I))*BAUX**P
      F=F+A1
   60 CONTINUE
      F=F-FIJ
      IF(F)77,80,80
   77 L=L+1
      ALFA=4.0**L
      GO TO 40
   80 IF(ALFA-AALFA)85,81,85
   81 XALFA=AALFA/4.0
   82 XF=0.0
      DO 600 I=1,M
```

```
      IF(A(I,J))500,600,500
  500 AUX=A(I,J)*(X(J)-XALFA*DIR)*NI(I)+D(I)
      BAUX=DABS(AUX)
      A1=(1.0/NI(I))*BAUX**P
      XF=XF+A1
  600 CONTINUE
      XF=XF-FIJ
      IF(XF)83,883,84
  883 ALFAN=XALFA
  888 INEWT=0
      DERF=1.0
      GO TO 100
   83 INEWT=1
      GO TO 89
   84 IF(XF-1.0D-15)83,83,884
  884 ALFA=XALFA
      XALFA=XALFA/4.0
      GO TO 82
   85 AMPLIT=(ALFA-ALFA/4.0)/4.0
      XALFA=ALFA-3.0*AMPLIT
   86 XF=0.0
      DO 606 I=1,M
      IF(A(I,J))505,606,505
  505 AUX=A(I,J)*(X(J)-XALFA*DIR)*NI(I)+D(I)
      BAUX=DABS(AUX)
      A1=(1.0/NI(I))*BAUX**P
      XF=XF+A1
  606 CONTINUE
      XF=XF-FIJ
      IF(XF)87,883,88
   87 INEWT=1
      GO TO 89
   88 IF(XF-1.0D-15)83,83,881
  881 AMPLIT=AMPLIT/4.0
      ALFA=XALFA
      XALFA=XALFA-3.0*AMPLIT
      GO TO 86
C
C      CALCULO DE XBARRA - METODO DE NEWTON
C
   89 INEWT=0
   91 F=0.0
      DERF=0.0
      DO 860 I=1,M
      IF(A(I,J))850,860,850
  850 AUX=A(I,J)*(X(J)-ALFA*DIR)*NI(I)+D(I)
      BAUX=DABS(AUX)
      A1=(1.0/NI(I))*BAUX**P
      A2=-A(I,J)*DIR*BAUX**(P-2.0)*AUX
      DERF=DERF+A2
      F=F+A1
  860 CONTINUE
```


```
      DERF=DERF*P
      F=F-FIJ
      ALFAN=ALFA-(F/DERF)
      DIFER=DABS(ALFAN-ALFA)
      IF(DIFER-1.0D-10)100,100,90
   90 IF(DABS(F)-1.0D-10)100,100,95
   95 ALFA=ALFAN
      INEWT=INEWT+1
      IF(INEWT-200)91,210,91
  100 XBARRA=X(J)-ALFAN*DIR
C
C      DETERMINACAO DO NOVO ITERADO
C
      XNOVO(J)=X(J)+BETA*(XBARRA-X(J))
      ALFAX(J)=ALFAN
  110 CONTINUE
C
C      TESTE DE PARADA
C
      DO 140 I=1,N
      IF(DABS(XNOVO(I)-X(I))-TOL)130,130,120
  120 IND=1
  130 X(I)=XNOVO(I)
  140 CONTINUE
      IF(IND)150,190,150
  150 CONTINUE
      WRITE(*,160)KMAX,TOL
  160 FORMAT(/,' O ALGORITMO NAO CONVERGIU APOS',I5,' ITERACOES
     *COM TOLERANCIA',D15.7)
      WRITE(8,160)KMAX,TOL
      GO TO 230
  210 WRITE(8,2000)K
 2000 FORMAT(/,1X,' MET. DE NEWTON EM LOOPPING',' K=',I3)
      GO TO 230
C
C      CALCULO DO VALOR OTIMO E IMPRESSAO DOS RESULTADOS
C       FINAIS
C
  190 V=0.0
      DO 155 I=1,M
      AUX=0.0
      DO 144 J=1,N
      AUX=AUX+A(I,J)*XNOVO(J)
  144 CONTINUE
      V=V+DABS(AUX-B(I))**P
  155 CONTINUE
      CALL GETTIM(ihr,imin,isec,i100th)
      WRITE(8,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100t
     *h
     WRITE(*,'(1X,I2.2,1H:,I2.2,1H:,I2.2,1H.,I2.2)')ihr,imin,isec,i100th
      WRITE(8,200)K,V
  200 FORMAT(/,' SOLUCAO APOS',I5,' ITERACOES COM VALOR
```


```
     *OTIMO=',D14.7)
      WRITE(*,200)K,V
      WRITE(8,205)(XNOVO(I),I=1,N)
      WRITE(*,205)(XNOVO(I),I=1,N)
  205 FORMAT(1X,5D14.7)
  220 CLOSE(2)
  230 STOP
      END
C
C      SUB-ROTINA DJIK
C
      SUBROUTINE DJIK
      DIMENSION A(1000,100),B(1000),NI(1000),D(1000),X(100)
      COMMON A,B,NI,X,D,M,N,J
      DOUBLE PRECISION A,B,X,D,AUX
      DO 20 I=1,M
      AUX=0.0
      DO 10 K=1,N
      AUX=AUX+A(I,K)*X(K)
   10 CONTINUE
      D(I)=AUX-NI(I)*A(I,J)*X(J)-B(I)
   20 CONTINUE
      RETURN
      END
```