UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO

Jeanne de Oliveira Pereira

# MÉTODO DE MÍNIMOS QUADRADOS NÃO LINEARES PARA USO EM SOFTWARE JAVA PARA POSICIONAMENTO ÓTIMO DE WLAN'S

Belém – PA
2017

Jeanne de Oliveira Pereira

# MÉTODO DE MÍNIMOS QUADRADOS NÃO LINEARES PARA USO EM SOFTWARE JAVA PARA POSICIONAMENTO ÓTIMO DE WLAN'S

Trabalho de Conclusão de Curso apresentado para obtenção do título de Bacharel em Ciência da Computação. Instituto de Ciências Exatas e Naturais. Faculdade de Computação. Universidade Federal do Pará.

Orientador Prof. Dr. Hermínio Simões Gomes

Belém – PA

2017

Dados Internacionais de Catalogação - na- Publicação (CIP)
Biblioteca de Pós-Graduação do ICEN/UFPA

---

Pereira, Jeanne de Oliveira
Método de mínimos quadrados não lineares para uso em software java para posicionamento ótimo de WLAN`S/Jeanne de Oliveira Pereira; orientador, Hermínio Simões Gomes.-2017.
84 f.: il. 29 cm
Inclui bibliografias

Trabalho de Conclusão de Curso para obtenção do título de Bacharel em Ciências da Computação, Universidade Federal do Pará Instituto de Ciências Exatas e Naturais, Faculdade de Computação, Belém, 2017.

1. Sistemas de comunicação sem fio. 2. Mínimos quadrados. 3. Java( linguagem de programação de computador). 4. Regressão linear. 5. Equações não lineares. I. Gomes, Hermínio Simões, orient. II.Título.

CDD 22 ed. 621.382

---

Jeanne de Oliveira Pereira

# MÉTODO DE MÍNIMOS QUADRADOS NÃO LINEARES PARA USO EM SOFTWARE JAVA PARA POSICIONAMENTO ÓTIMO DE WLAN'S

Trabalho de Conclusão de Curso apresentado para obtenção do título de Bacharel em Ciência da Computação. Instituto de Ciências Exatas e Naturais. Faculdade de Computação. Universidade Federal do Pará.

Data da aprovação: Belém-PA, 12/04/2017

Banca Examinadora

Prof. Dr. Hermínio Simões Gomes
UFPA – Orientador

Profª. Drª. Simone da Graça de Castro Fraiha
UFPA – Membro

Prof. Dr. Dionne Cavalcante Monteiro
UFPA – Membro

BELÉM – PA
2017

Aos meus pais, pessoas indispensáveis na minha vida.

## AGRADECIMENTOS

A Deus por ter me permitido vencer mais esta etapa em minha vida.

Aos meus pais José Carlos e Margarida por sempre me apoiarem nas coisas que realmente valeriam a pena para mim, pela educação que me deram e por estarem presentes nos mais diversos momentos da minha vida.

Ao meu irmão Ruan pela nossa convivência.

À Universidade Federal do Pará por ter proporcionado os meus estudos de graduação.

Ao Prof. Hermínio Gomes que me orientou neste trabalho e pelo incentivo durante o curso.

À Profa. Simone Fraiha pela oportunidade de participar como bolsista do projeto que contribuiu para este trabalho.

Ao Prof. Dionne Monteiro por ter aceitado participar da banca de avaliação deste trabalho e por ele ter contribuído com as suas aulas.

Aos meus professores da graduação que ao longo de minha trajetória no curso contribuíram para a construção do meu conhecimento.

Aos meus colegas da graduação que compartilharam comigo as experiências do curso.

A todos aqueles que contribuíram direta ou indiretamente para a realização deste trabalho.

**RESUMO**

Nos últimos anos a necessidade de comunicação sem fio tem aumentado muito. Comunicações *indoor* (ambientes internos) são realizadas através de sistemas WLAN (*Wireless Local Area Network*) que transmitem dados provenientes da internet para computadores e sistemas móveis. Para otimizar os serviços deste tipo de comunicação são usados, às vezes modelos empíricos, que necessitam de cálculo de parâmetros via métodos de otimização do tipo ajuste pelo método de mínimos quadrados não lineares. Neste trabalho o método citado é abordado com uso das variantes Gauss-Newton e Levenberg-Marquardt, com exemplos de ajuste de curvas. O método de Levenberg-Marquardt é abordado na versão irrestrita e também com restrições do tipo caixa, incluindo a solução de sistemas de equações não lineares. Por fim o Levenberg-Marquardt com restrições do tipo caixa foi implementado em um *software* Java para o posicionamento ótimo de WLANs com objetivo de maximizar a cobertura de sinal.

PALAVRAS-CHAVE: Mínimos Quadrados Lineares, Regressão Linear, Mínimos Quadrados Não Lineares, Gauss-Newton, Levenberg-Marquardt, Java, WLAN.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE EQUAÇÕES

## LISTA DE ALGORITMOS

## LISTA DE SÍMBOLOS

| | |
|---|---|
| MQL | Mínimos Quadrados Lineares |
| MQNL | Mínimos Quadrados Não Lineares |
| MMQL | Método de Mínimos Quadrados Lineares |
| MMQNL | Método de Mínimos Quadrados Não Lineares |
| PA | Ponto de Acesso |
| RBF | *Radial Basis Function* |
| WLAN | *Wireless Local Area Network* |

**SUMÁRIO**

# 1 INTRODUÇÃO

Recentemente observou-se uma grande importância e crescimento das redes WLAN (*Wireless* LAN), para transmissão de dados de internet em ambientes *indoor*, demandando uma necessidade de eficiência e bom desempenho no serviço. Convém realizar estudos de como um sinal de rádio proveniente de uma transmissão WLAN se comporta em ambientes fechados e com obstáculos. Para fazer a predição do comportamento deste sinal de rádio em ambientes *indoor*, um aplicativo Java foi criado com fundamentos em Fraiha (2009). De forma breve o aplicativo conta com as etapas: expansão dos dados, cálculo dos parâmetros de Padé, que servem para ajustar um modelo de previsão de perda de propagação em WLAN, e cálculo das probabilidades de recepção, Carneiro (2011) desenvolveu desde os estágios iniciais até a expansão dos dados, com isso o aplicativo necessitava de uma rotina de mínimos quadrados não lineares (MQNL) com restrições do tipo caixa para a determinação dos parâmetros de Padé.

## 1.1 Objetivos

Este trabalho tem como objetivo geral discutir o método de mínimos quadrados não lineares (MMQNL) e mostrar a aplicação deste no *software* Java para posicionamento ótimo de WLANs citado no primeiro parágrafo da introdução.

Este trabalho tem como objetivo específicos:

i. Mostrar exemplos de ajuste de curva com o método de mínimos quadrados lineares (MMQL) e com o MMQNL.
ii. Discutir a implementação de restrições do tipo caixa para o MMQNL.
iii. Explanar acerca da solução de sistemas de equações não lineares pelo MMQNL.

## 1.2 Metodologia

De início, o desenvolvimento foi feito por meio de pesquisa técnica em materiais técnicos de referência, em livros e em sites da internet. O estudo do material técnico coletado foi utilizado como base para o desenvolvimento teórico, que foi uma visão detalhada da problemática tanto dos métodos e algoritmos. A partir daí foi possível implementar o MMQNL no *software* Java para posicionamento ótimo de WLANs.

## 1.3 Organização

Este trabalho está dividido em 7 capítulos e organizado da seguinte forma:

i. O capítulo 1 trata de introduzir o assunto destacando a motivação que levou ao estudo exposto neste trabalho.
ii. O capítulo 2 traz um conceito geral do método de mínimos quadrados.
iii. O capítulo 3 faz uma abordagem do MMQL e de um caso especial desse método que é a regressão linear.
iv. O capítulo 4 aborda o MMQNL e discute os métodos de Gauss-Newton e Levenberg-Marquardt, neste último são incluídas restrições do tipo caixa.
v. O capítulo 5 trata do aplicativo Java para posicionamento ótimo de WLANs.
vi. O capítulo 6 fala brevemente dos resultados obtidos com o Levenberg-Marquardt com restrições do tipo caixa colocado no programa tratado no capítulo 5.
vii. O capítulo 7 fala das considerações finais com as conclusões acerca do trabalho.

**1.4 Observações**

Por facilidade de edição os valores tabelados e valores decimais foram tratados com separador decimal ponto ao invés de vírgula.

Parte dos gráficos de funções do trabalho foram conseguidos usando-se a biblioteca livre JFreeChart em sua versão 1.0.19 do Java.

## 2 O MÉTODO DE MÍNIMOS QUADRADOS

No conceito geral do método de mínimos quadrados dado a função $G(\alpha_1, \alpha_2, \dots, \alpha_n; \boldsymbol{x})$, com $G: \mathbb{R}^k \to \mathbb{R}^m$, $G = (g_1, \dots, g_m)^T$ onde $k$ é o número de componentes de $\boldsymbol{x}$, $m$ é o número de equações que compõem $G$ e a função possui $n$ parâmetros. Procura-se resolver o sistema linear ou não linear $G(\boldsymbol{\alpha}; \boldsymbol{x}) = 0$, como geralmente ele não tem solução, busca-se as soluções que minimizem $\sum_{i=1}^{m} g_i(\boldsymbol{\alpha}; \boldsymbol{x})$. Estas soluções são chamadas de soluções de mínimos quadrados lineares (MQL) ou de mínimos quadrados não lineares (MQNL) (CHONG, ŻAK, 2001).

Deseja-se, portanto resolver o problema:

$$\text{Min} \| G(\boldsymbol{\alpha}; \boldsymbol{x}_{teórico}) - G(\boldsymbol{\alpha}; \boldsymbol{x}_{observado}) \|^2 \tag{2.1}$$

Fonte: Elaborado pela própria autora

Onde a minimização ocorre no espaço gerado por $\boldsymbol{\alpha}$ (CHONG, ŻAK, 2001). O $\|\cdot\|$ representa a norma-2 de um vetor, com as equações que compõem $G$: $\|G\| = \sqrt{{g_1}^2 + \cdots + {g_m}^2}$ (MADSEN, NIELSEN, TINGLEFF, 2004). Na discussão dos capítulos 3 e 4 acerca do método de mínimos quadrados lineares (MMQL) e do método de mínimos quadrados não lineares (MMQNL) $\boldsymbol{x}$ não será tratado como sendo um vetor, mas sim como uma variável simples.

# 3 O MÉTODO DE MÍNIMOS QUADRADOS LINEARES

O MMQL publicado pela primeira vez por Adrien-Marie Legendre (1752-1833) em 1805, já era usado por Carl Friedrich Gauss (1777-1855) desde 1795 em seus cálculos na Astronomia. Este método é bastante usado em problemas práticos, pela sua simplicidade e por dados obtidos experimentalmente terem certo grau de incerteza (CUNHA, 2000).

O MMQL consiste em minimizar os resíduos (erros nos dados a serem aproximados) sendo as funções de aproximação lineares em relação aos parâmetros (CUNHA, 2000). Uma função para aproximar os dados será do tipo da equação (3.1), as funções $\phi_1, \phi_2, \dots, \phi_n$ são conhecidas, um exemplo são os monômios $x^i$ ou $\sin(\pi i x)$:

$$f(x) \cong g(x) = \alpha_1\phi_1(x) + \alpha_2\phi_2(x) + \cdots + \alpha_n\phi_n(x) \tag{3.1}$$

Fonte: (CUNHA, 2000, p. 104)

O desvio ou resíduo no ponto $x$ será:

$$\begin{aligned} r(x) &= g(x) - f(x) \\ &= \left[\left(\alpha_1\phi_1(x) + \alpha_2\phi_2(x) + \cdots + \alpha_n\phi_n(x)\right)\right] - f(x) \\ &= r(x; \alpha_1, \alpha_2, \dots, \alpha_n) \end{aligned} \tag{3.2}$$

Fonte: Adaptado de (CUNHA, 2000, p. 104)

No caso dos mínimos quadrados se quer minimizar:

$$\sum_{i=1}^{m} r^2(x_i) = \sum_{i=1}^{m}\left(g(x_i) - f(x_i)\right)^2 \tag{3.3}$$

Fonte: Adaptado de (CUNHA, 2000, p. 104)

Onde $f(x_i),\ i = 1{:}m$ representa os dados que serão aproximados, com a minimização pretende-se determinar os coeficientes $\alpha_1, \alpha_2, \dots, \alpha_n$ que fornecem a melhor aproximação (CUNHA, 2000).

Considerando $r$ um vetor pode se fazer uso da norma-2 de um vetor (MADSEN, NIELSEN, TINGLEFF, 2004):

$$\sum_{i=1}^{m} r^2(x_i) = \|\boldsymbol{r}\|^2 = \boldsymbol{r}^T\boldsymbol{r}$$

Fonte: Elaborado pela própria autora

É útil utilizar a notação de produto escalar entre duas funções (CUNHA, 2000):

$$< f, g > = < f(x), g(x) > = \sum_{i=1}^{m} f(x_i)g(x_i) \quad \text{(no caso discreto)} \tag{3.4}$$

Fonte: (CUNHA, 2000, p. 104)

A operação de produto escalar possui as propriedades (3.5) (CUNHA, 2000):

$$\text{i.} < \omega_1 g_1 + \omega_2 g_2, h \geq \omega_1 < g_1, h > + \omega_2 < g_2, h >, \ \text{linearidade} \tag{3.5}$$

ii. $< f, g >=< g, f >$ , comutatividade

iii. $< g, g > \geq 0$ e $< g, g >= 0 \Leftrightarrow g \equiv 0$, positividade

Fonte: (CUNHA, 2000, p. 105)

Fazendo uso da notação introduzida em (3.4) está se minimizando $< r, r >$ que é uma função de $\alpha_1, \alpha_2, \dots, \alpha_n$ (CUNHA, 2000).

A função que se deseja minimizar é $F$ (CUNHA, 2000):

$$F(\alpha_1, \alpha_2, \dots, \alpha_n) =< r, r > \\ =< \alpha_1\phi_1(x) + \cdots + \alpha_n\phi_n(x) - f(x), \alpha_1\phi_1(x) + \cdots + \alpha_n\phi_n(x) - f(x) > \quad (3.6)$$

Fonte: (CUNHA, 2000, p. 107)

$$\frac{\partial F}{\partial \alpha_j} = 2 < \alpha_1\phi_1(x) + \alpha_2\phi_2(x) + \cdots + \alpha_n\phi_n(x) - f(x), \phi_j(x) > \quad (3.7)$$

Fonte: (CUNHA, 2000, p. 108)

Na distribuição do produto escalar (CUNHA, 2000):

$$\alpha_1 < \phi_1, \phi_j > +\alpha_2 < \phi_2, \phi_j > + \cdots + \alpha_n < \phi_n, \phi_j >=< f, \phi_j >, \ j = 1{:}\, n \quad (3.8)$$

Fonte: (CUNHA, 2000, p. 108)

O sistema de equações normal é composto pelas equações em (3.8) com $j = 1, j = 2, \dots, j = n$ (CUNHA, 2000):

$$\begin{cases} < \phi_1, \phi_1 > \alpha_1 + < \phi_2, \phi_1 > \alpha_2 + \cdots + < \phi_n, \phi_1 > \alpha_n & = & < f, \phi_1 > \\ < \phi_1, \phi_2 > \alpha_1 + < \phi_2, \phi_2 > \alpha_2 + \cdots + < \phi_n, \phi_2 > \alpha_n & = & < f, \phi_2 > \\ \vdots & \vdots & \vdots \\ < \phi_1, \phi_n > \alpha_1 + < \phi_2, \phi_n > \alpha_2 + \cdots + < \phi_n, \phi_n > \alpha_n & = & < f, \phi_n > \end{cases} \quad (3.9)$$

Fonte: (CUNHA, 2000, p. 108)

A resolução do sistema normal fornece os coeficientes para a melhor aproximação. Ele é simétrico, já que $< \phi_i, \phi_j >=< \phi_j, \phi_i >$, é garantida a existência de uma única solução para o sistema se as funções $\phi_i(x)$ são linearmente independentes, o que torna a matriz das incógnitas do sistema normal positiva definida (CUNHA, 2000).

Na forma matricial:

$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{\alpha} = \boldsymbol{A}^T\boldsymbol{f} \quad (3.10)$$

Fonte: (CUNHA, 2000, p. 108)

Em que $\boldsymbol{A}_{m\times n}$ é uma matriz na qual a $j-$é$sima$ coluna é o vetor $\boldsymbol{u}_j = \left(\phi_j(x_1), \phi_j(x_2), \dots, \phi_j(x_m)\right)^T$ , $j = 1{:}\, n$, as $m$ componentes são os valores das funções nos pontos tabelados, $\boldsymbol{f} = \left(f(x_1), f(x_2), \dots, f(x_m)\right)^T$ é o vetor coluna dos dados e $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_n)^T$ é o vetor coluna dos coeficientes para a melhor aproximação (CUNHA, 2000).

A denominação “normal” é devido ao resíduo ser ortogonal a cada um dos vetores $\boldsymbol{u}_j$, a projeção ortogonal dos dados $\boldsymbol{f}$ no espaço gerado por $\boldsymbol{u}_j$, $j = 1{:}\,n$, colunas da matriz $\boldsymbol{A}$ é a melhor aproximação (CUNHA, 2000).

Para o caso dos MQL geral várias abordagens de resolução de sistemas de equações lineares podem ser utilizadas para resolver o sistema normal. Para encontrar o valor do $\boldsymbol{\alpha}$ pode se tentar resolver o sistema $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{\alpha} = \boldsymbol{A}^T\boldsymbol{f}$ com o auxílio da decomposição de *Cholesky* ou usar a fatoração QR da matriz $\boldsymbol{A}$ (NOCEDAL, WRIGHT, 2006).

### 3.1 Regressão Linear

Um caso particular do MMQL consiste na regressão linear. O termo regressão se deve a *Sir* Francis Galton quando em 1885 num estudo sobre a altura dos filhos, mostrou que ela tende a regredir para a média da população e não tende a refletir a altura dos pais (RODRIGUES, 2012) *apud* (MAROCO, 2003). A regressão linear é conhecida por aplicação na estatística e em outras áreas, nela se deseja ajustar dados a uma reta.

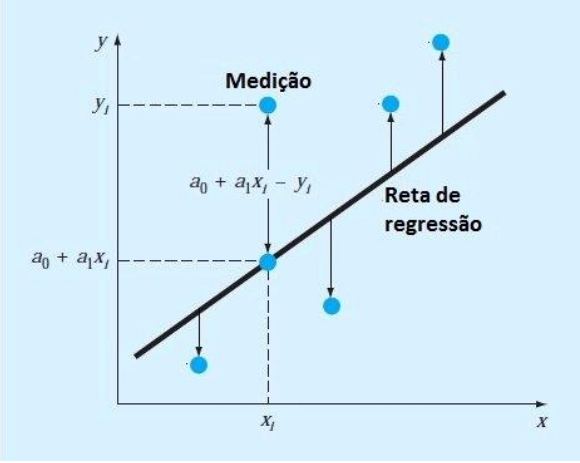
Fornecido um conjunto de $m$ valores relacionados $\big(x_i, f(x_i)\big)$, quer se encontrar a reta que melhor ajusta esses dados. As funções $\phi_1(x)$ e $\phi_2(x)$ serão 1 e $x$ respectivamente (CUNHA, 2000).

$$f(x) \cong g(x) = a_0 + a_1 x \tag{3.11}$$

Fonte: (CUNHA, 2000, p. 105)

O resíduo será $r(x; a_0, a_1) = a_0 + a_1 x - f(x)$ e representa a distância vertical entre o ponto dado e a reta de regressão como está na figura 1 (CHAPRA, CANALE, 2010).

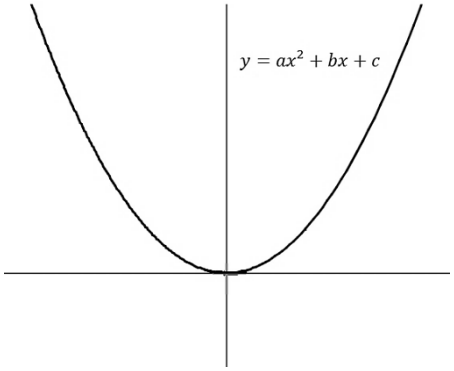
**Figura 1 – Pontos medidos e reta de regressão**




Fonte: Adaptado de (CHAPRA, CANALE, 2010, p. 459)

A escolha de uma função quadrática faz a condição de ponto crítico levar a um único ponto de mínimo, pode-se pensar no caso da parábola (CUNHA, 2000).

**Figura 2 – Parábola**

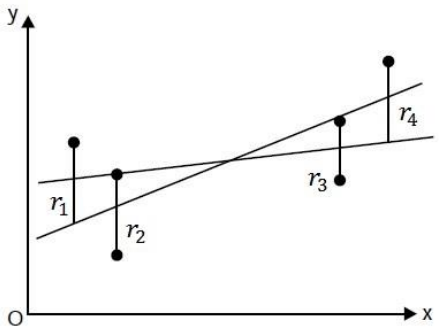



Fonte: Adaptado de (KILHIAN, 2015)

A simples escolha de $\sum_{i=1}^{m} r_i = \sum_{i=1}^{m} g(x_i) - f(x_i)$ para o cálculo da somatória poderia fazer com que ela se anulasse imediatamente no caso da soma dos resíduos positivos serem iguais a soma dos resíduos negativos (DUKKIPATI, 2010).

Já o uso da função modular $|g(x) - f(x)|$ seria inadequado. Para os mesmos quatro pontos na figura 3 as duas retas dão o mesmo erro total $E$. $E = \sum_{i=1}^{m} |r_i| = \sum_{i=1}^{m} |g(x_i) - f(x_i)| = \sum_{i=1}^{m} |a_0 + a_1 x_i - f(x_i)|$ (DUKKIPATI, 2010).

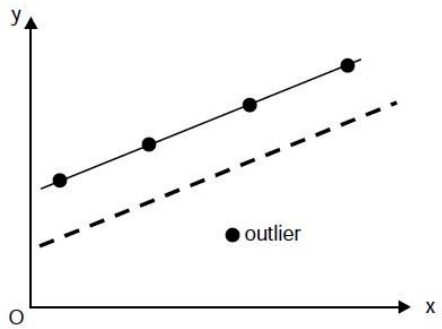
**Figura 3 – Duas retas com o mesmo erro total $E$**




Fonte: Adaptado de (DUKKIPATI, 2010, p. 193)

Outro critério para ajuste de reta é o critério *minmax*, o qual se baseia em minimizar a distância para um ponto fora da reta, porém tem grande influência de um ponto isolado e indesejado (*outlier)* (DUKKIPATI, 2010).

**Figura 4 – Minimização do máximo erro para um ponto individual**




Fonte: (DUKKIPATI, 2010, p. 194)

Procura-se $a_0$ e $a_1$ que minimizem a função $< r, r >$, do cálculo diferencial no ponto crítico as derivadas nele se anulam (CUNHA, 2000).

$$< r, r > (a_0, a_1) =< a_0 + a_1 x - f(x), a_0 + a_1 x - f(x) >= \\ = \sum_{i=1}^{m} \left(a_0 + a_1 x_i - f(x_i)\right)^2 \tag{3.12}$$

Fonte: Adaptado de (CUNHA, 2000, p. 105)

$$\frac{\partial < r, r >}{\partial a_0} = \frac{\partial < r, r >}{\partial a_1} = 0 \tag{3.13}$$

Fonte: (CUNHA, 2000, p. 105)

Derivando $< r, r >$ com relação a $a_0$ e $a_1$, obtém-se:

$$\sum_{i=1}^{m} 2\left(a_0 + a_1 x_i - f(x_i)\right) \text{ e } \sum_{i=1}^{m} 2x_i\left(a_0 + a_1 x_i - f(x_i)\right) \tag{3.14}$$

Fonte: Adaptado de (CUNHA, 2000, p. 105)

Isto dá um sistema linear $2 \times 2$ com incógnitas $a_0$ e $a_1$ (DUKKIPATI, 2010):

$$\begin{cases} ma_0 + a_1 \sum_{i=1}^{m} x_i & = & \sum_{i=1}^{m} f(x_i) \\ a_0 \sum_{i=1}^{m} x_i + a_1 \sum_{i=1}^{m} {x_i}^2 & = & \sum_{i=1}^{m} x_i f(x_i) \end{cases} \tag{3.15}$$

Fonte: Adaptado de (CUNHA, 2000, p. 105)

Como o sistema apresenta uma única solução, já que as funções $\phi_1(x) = 1$ e $\phi_2(x) = x$ são linearmente independentes, pode-se resolvê-lo com o auxílio de determinantes pela regra de Cramer (DUKKIPATI, 2010).

$$\Delta = m \sum_{i=1}^{m} {x_i}^2 - \left(\sum_{i=1}^{m} x_i\right)^2 \tag{3.16}$$

Fonte: Elaborado pela própria autora

$$\Delta a_1 = m \sum_{i=1}^{m} x_i f(x_i) - \sum_{i=1}^{m} x_i \sum_{i=1}^{m} f(x_i) \tag{3.17}$$

Fonte: Elaborado pela própria autora

$$a_1 = \frac{\Delta a_1}{\Delta} \tag{3.18}$$

Fonte: Elaborado pela própria autora

Com o valor de $a_1$ e com a primeira equação do sistema é possível encontrar o valor de $a_0$:

$$ma_0 + a_1 \sum_{i=1}^{m} x_i = \sum_{i=1}^{m} f(x_i) \tag{3.19}$$

Fonte: Elaborado pela própria autora

Dividindo tudo por $m$ para isolar $a_0$, a expressão tem a média dos valores de $x$ e a média dos valores de $y$:

$$a_0 + \frac{a_1 \sum_{i=1}^{m} x_i}{m} = \frac{\sum_{i=1}^{m} f(x_i)}{m}$$
$$a_0 + a_1 \overline{\boldsymbol{x}} = \overline{\boldsymbol{y}}, \text{ com } \boldsymbol{y} = \left(f(x_1), f(x_2), \dots, f(x_m)\right)^T \quad (3.20)$$
$$a_0 = \overline{\boldsymbol{y}} - a_1 \overline{\boldsymbol{x}}$$

Fonte: Elaborado pela própria autora

A partir daí dado um valor de $x$ é possível encontrar o valor do respectivo $y$:

$$\hat{y} = a_0 + a_1 x \quad (3.21)$$

Fonte: Elaborado pela própria autora

Observa-se que aparecem os seguintes somatórios:

$$S_x = \sum_{i=1}^{m} x_i \text{ e } S_y = \sum_{i=1}^{m} f(x_i)$$
$$S_{xx} = \sum_{i=1}^{m} {x_i}^2 \text{ e } S_{xy} = \sum_{i=1}^{m} x_i f(x_i) \quad (3.22)$$

Fonte: Elaborado pela própria autora

Assim o $a_1$ pode ser calculado da seguinte forma:

$$a_1 = \frac{\Delta a_1}{\Delta} = \frac{\sum_{i=1}^{m} x_i f(x_i) - \frac{\sum_{i=1}^{m} x_i \sum_{i=1}^{m} f(x_i)}{m}}{\sum_{i=1}^{m} {x_i}^2 - \frac{(\sum_{i=1}^{m} x_i)^2}{m}} = \frac{S_{xy} - \frac{S_x S_y}{m}}{S_{xx} - \frac{S_x S_x}{m}} \quad (3.23)$$

Fonte: Elaborado pela própria autora

Outra possibilidade para a regressão linear é utilizar a reta bissetriz das retas da regressão linear de $y$ em função de $x$ e da regressão linear de $x$ em função de $y$, e apresenta um dos melhores desempenhos assim como a regressão linear de $y$ em função de $x$ (BABU, FEIGELSON, 1992).

Primeiro é necessário obter os coeficientes da regressão linear de $x$ em função de $y$, para isso o cálculo da somatória dos resíduos passa a ser:

$$< r, r > (b_0, b_1) = < b_0 + b_1 f(x) - x, b_0 + b_1 f(x) - x > =$$
$$= \sum_{i=1}^{m} (b_0 + b_1 f(x_i) - x_i)^2 \quad (3.24)$$

Fonte: Elaborado pela própria autora

Derivando $< r, r >$ com relação a $b_0$ e $b_1$, obtém-se:

$$\sum_{i=1}^{m} 2(b_0 + b_1 f(x_i) - x_i) \text{ e } \sum_{i=1}^{m} 2f(x_i)(b_0 + b_1 f(x_i) - x_i) \tag{3.25}$$

Fonte: Elaborado pela própria autora

Igualando as derivadas a 0 o sistema obtido é similar ao da regressão linear de $y$ em função de $x$:

$$\begin{cases} mb_0 + b_1 \sum_{i=1}^{m} f(x_i) = \sum_{i=1}^{m} x_i \\ b_0 \sum_{i=1}^{m} f(x_i) + b_1 \sum_{i=1}^{m} f(x_i)^2 = \sum_{i=1}^{m} f(x_i) x_i \end{cases} \tag{3.26}$$

Fonte: Elaborado pela própria autora

O cálculo do coeficiente $b_1$ é dado por:

$$b_1 = \frac{\Delta b_1}{\Delta} = \frac{\sum_{i=1}^{m} f(x_i)x_i - \frac{\sum_{i=1}^{m} f(x_i) \sum_{i=1}^{m} x_i}{m}}{\sum_{i=1}^{m} f(x_i)^2 - \frac{\left(\sum_{i=1}^{m} f(x_i)\right)^2}{m}} = \frac{S_{xy} - \frac{S_x S_y}{m}}{S_{yy} - \frac{S_y S_y}{m}} \tag{3.27}$$

Fonte: Elaborado pela própria autora

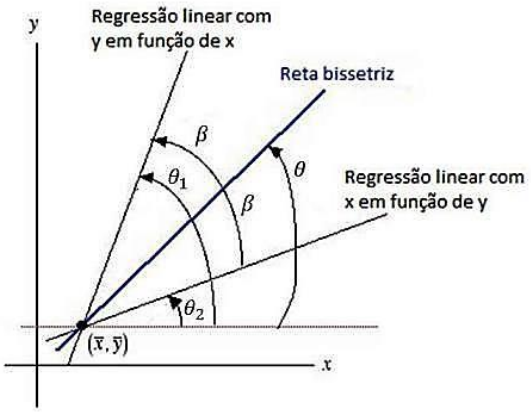
O $b_0$ é dado por:

$$b_0 = \frac{\sum_{i=1}^{m} x_i}{m} - \frac{b_1 \sum_{i=1}^{m} f(x_i)}{m} = \overline{\boldsymbol{x}} - b_1 \overline{\boldsymbol{y}} \tag{3.28}$$

Fonte: Elaborado pela própria autora

Como já visto anteriormente no cálculo de $a_0 = \overline{\boldsymbol{y}} - a_1 \overline{\boldsymbol{x}}$ e $b_0 = \overline{\boldsymbol{x}} - b_1 \overline{\boldsymbol{y}}$ as retas da regressão linear de $y$ em função de $x$ e da regressão linear de $x$ em função de $y$ passam pelo ponto $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}})$ e a reta bissetriz deve necessariamente passar pelo ponto $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}})$ (RICH, THOMAS, 2009). A figura 5 ilustra as três retas passando pelo ponto $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}})$.

**Figura 5 – Regressão linear com a bissetriz**




Fonte: Adaptado de (WEISS, 2003)

O coeficiente angular da reta bissetriz pode ser calculado como segue:

$$2\beta = \theta_1 - \theta_2$$

$$\beta = \frac{\theta_1 - \theta_2}{2}$$

$$\theta = \beta + \theta_2 = \frac{\theta_1 - \theta_2}{2} + \theta_2 = \frac{\theta_1 + \theta_2}{2} \tag{3.29}$$

$$\theta = \frac{\theta_1 + \theta_2}{2}$$

$$\theta_1 = \tan^{-1}(a_1)$$

$$\theta_2 = \tan^{-1}\left(\frac{1}{b_1}\right)$$

Fonte: Elaborado pela própria autora

O valor de $c_0$ e $c_1$ é então determinado:

$$c_1 = \tan(\theta) \quad \text{e} \quad c_0 = \overline{\boldsymbol{y}} - c_1\overline{\boldsymbol{x}} \tag{3.30}$$

Fonte: Elaborado pela própria autora

Um algoritmo para a regressão linear está descrito no algoritmo 3.1.

**Algoritmo 3.1 Regressão Linear**

**Dados** $x_i, f(x_i)$
01: $S_x = \sum_{i=1}^{m} x_i\,; \quad S_y = \sum_{i=1}^{m} f(x_i)$
02: $S_{xx} = \sum_{i=1}^{m} {x_i}^2\,; \quad S_{yy} = \sum_{i=1}^{m} f(x_i)^2$
03: $S_{xy} = \sum_{i=1}^{m} x_i f(x_i)$
04: $a_1 = \frac{S_{xy} - \frac{S_x S_y}{m}}{S_{xx} - \frac{S_x S_x}{m}}; \quad a_0 = \overline{\boldsymbol{y}} - a_1\overline{\boldsymbol{x}}$
05: $b_1 = \frac{S_{xy} - \frac{S_x S_y}{m}}{S_{yy} - \frac{S_y S_y}{m}}; \quad b_0 = \overline{\boldsymbol{x}} - b_1\overline{\boldsymbol{y}}$
06: $\theta_1 = \tan^{-1}(a_1)\,; \quad \theta_2 = \tan^{-1}\left(\frac{1}{b_1}\right)$
07: $\theta = \frac{\theta_1 + \theta_2}{2}$
08: $c_1 = \tan(\theta)\,; \quad c_0 = \overline{\boldsymbol{y}} - c_1\overline{\boldsymbol{x}}$
09: $v = (a_0, a_1, b_0, b_1, c_0, c_1)$
10: **retornar** $v$

Fonte: Elaborado pela própria autora

## 3.2 Exemplo de Ajuste de Dados a uma Reta

Utilizar regressão linear para ajustar os dados da tabela 1, o $x$ será representado pelo mês e o $y$ representa o faturamento (em 1000 reais) de uma empresa. A partir daí pede-se para calcular qual será o faturamento em junho $(\hat{y}(6) = ?)$.

**Tabela 1 – Faturamento da empresa em cada mês**

| Mês | Faturamento (em 1000 reais) |
|---|---|

| | |
|---|---|
| JAN | 10.1 |
| FEV | 12.3 |
| MAR | 15.2 |
| ABR | 16.4 |
| ⋮ | ⋮ |
| JUN | ? |

Fonte: Elaborado pela própria autora

A tabela 2 representa a tabela de trabalho.

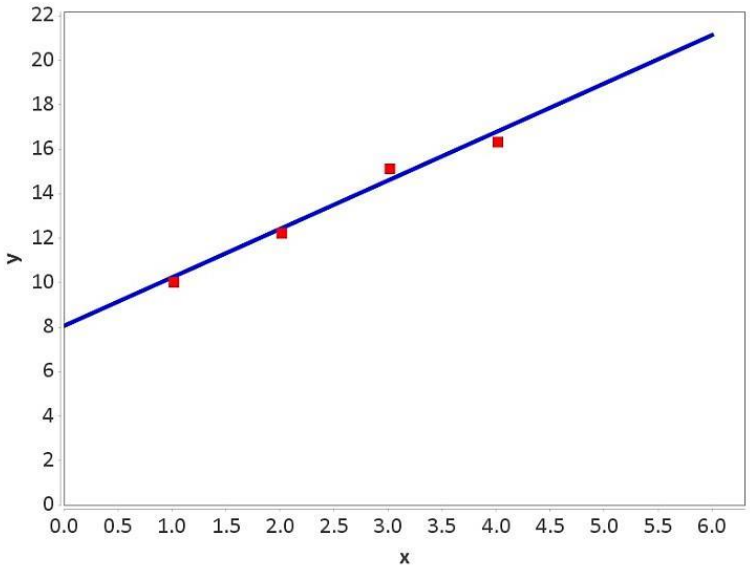
**Tabela 2 – Tabela de trabalho**

| $x$ | $y$ (em 1000) |
|---|---|
| 1 | 10.1 |
| 2 | 12.3 |
| 3 | 15.2 |
| 4 | 16.4 |
| ⋮ | ⋮ |
| 6 | ? |

Fonte: Elaborado pela própria autora

Os valores de $a_0$ e $a_1$ encontrados foram $a_0 = 8.05$, $a_1 = 2.18$ e $\hat{y}(6) = 8.05 + 2.18 \times 6 = 21.13$ mil, ou seja, o faturamento em junho será de $21.13$ mil reais. Na figura 6 os dados fornecidos representados pelos pontos em vermelho e a reta de ajuste em azul.

**Figura 6 – Reta de ajuste aos dados da tabela 1**




Fonte: Elaborado pela própria autora

Os resíduos estão na tabela 3, a solução dada pela reta bissetriz e a solução dada pela reta da regressão linear de $y$ em função de $x$ forneceram bons resultados. A soma dos quadrados dos resíduos desta última é 0.5380.

**Tabela 3 – Retas e resíduos para os dados da tabela 1**

| $x$ | $y$ (em 1000) | $8.05 + 2.18x$ | Resíduo | $7.9889 + 2.2044x$ |
|---|---|---|---|---|

| | | | $(8.05 + 2.18x - y)$ | (Bissetriz) |
|---|---|---|---|---|
| 1 | 10.1 | 10.2300 | 0.13 | 10.1933 |
| 2 | 12.3 | 12.4100 | 0.11 | 12.3978 |
| 3 | 15.2 | 14.5900 | -0.61 | 14.6022 |
| 4 | 16.4 | 16.7700 | 0.37 | 16.8067 |

Fonte: Elaborado pela própria autora

### 3.3 Linearização de Modelos Não Lineares

Nem sempre a relação entre a variável independente e dependente é linear, porém em certos casos é possível fazer uma transformação para os dados serem tratados por regressão linear, já que ela é uma alternativa mais simples do que usar um ajuste não linear. Um exemplo é o modelo exponencial: $y = a_0 e^{a_1 x}$, outros exemplos são $y = a_0 x^{a_1}$ e $y = a_0 \frac{x}{a_1 + x}$ (CHAPRA, CANALE, 2010).

Para a linearização de $y = a_0 e^{a_1 x}$ aplica-se logaritmo natural de ambos os lados:

$$y = a_0 e^{a_1 x} \tag{3.31}$$

$$\ln y = \ln(a_0 e^{a_1 x}) = \ln a_0 + \ln e^{a_1 x} = \ln a_0 + a_1 x \ln e = \ln a_0 + a_1 x$$

Fonte: Elaborado pela própria autora

$$\ln y = \ln a_0 + a_1 x \tag{3.32}$$

Fonte: (CHAPRA, CANALE, 2010, p. 468)

Para a linearização de $y = a_0 x^{a_1}$ aplica-se logaritmo em qualquer base de ambos os lados, podendo ser até mesmo o logaritmo natural:

$$y = a_0 x^{a_1} \tag{3.33}$$

$$\log y = \log(a_0 x^{a_1}) = \log a_0 + \log x^{a_1} = \log a_0 + a_1 \log x$$

Fonte: Elaborado pela própria autora

$$\log y = \log a_0 + a_1 \log x \tag{3.34}$$

Fonte: (CHAPRA, CANALE, 2010, p. 468)

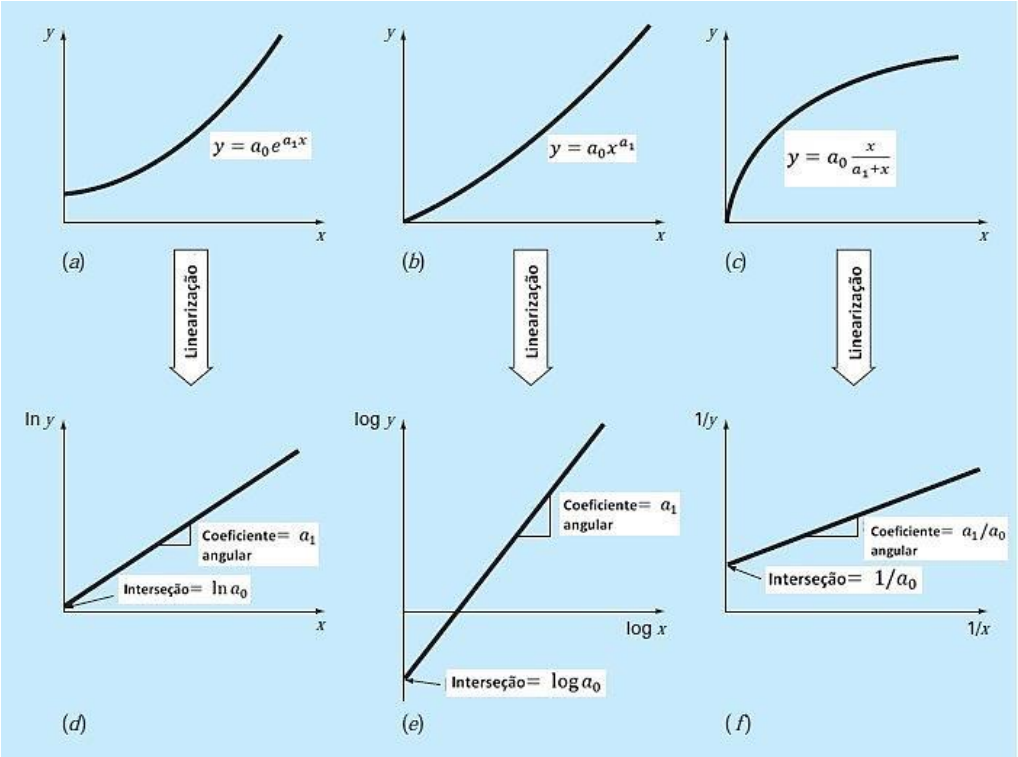
Para a linearização de $y = a_0 \frac{x}{a_1 + x}$, basta apenas inverter a expressão dada (CHAPRA, CANALE, 2010):

$$\frac{1}{y} = \frac{1}{a_0} \frac{a_1 + x}{x} = \frac{a_1}{a_0} \frac{1}{x} + \frac{1}{a_0} \tag{3.35}$$

Fonte: (CHAPRA, CANALE, 2010, p. 468)

A figura 7 ilustra as linearizações.

**Figura 7 – Linearização de alguns modelos não lineares**




Fonte: Adaptado de (CHAPRA, CANALE, 2010, p. 467)

A linearização é uma aplicação útil de composição de funções e de funções inversa em que a partir da relação não linear se deseja obter para a regressão linear uma relação linear de tal modo que $g(f(x)) = b + mh(x)$, em que $f(x) = y$, a função $g$ não é necessariamente a inversa de $y$ mas sim um tipo de "semi-inversa" (TEAGUE, 2005). Com os valores de $b$ e de $m$ determinados calcula-se então os valores dos parâmetros $a_0$ e $a_1$ necessários para o modelo não linear.

### 3.4 Exemplo de Ajuste de Dados com Linearização

Um exercício encontrado em (CHAPRA, CANALE, 2010, p. 485) é o ajuste de dados ao modelo que será linearizado $y = a_0 x e^{a_1 x}$.

$$y = a_0 x e^{a_1 x} \tag{3.36}$$

Fonte: Adaptado de (CHAPRA, CANALE, 2010, p. 485)

Dividindo tudo por $x$:

$$\frac{y}{x} = a_0 e^{a_1 x} \tag{3.37}$$

Fonte: Elaborado pela própria autora

Aplicando logaritmo de ambos os lados:

$$\ln\left(\frac{y}{x}\right) = \ln(a_0 e^{a_1 x}) \tag{3.38}$$

Fonte: Elaborado pela própria autora

Aplicando as propriedades logarítmicas:

$$\ln\left(\frac{y}{x}\right) = \ln a_0 + \ln(e^{a_1 x}) = \ln a_0 + a_1 x \ln e = \ln a_0 + a_1 x \tag{3.39}$$

Fonte: Elaborado pela própria autora

Por fim:

$$\ln\left(\frac{y}{x}\right) = \ln a_0 + a_1 x$$
$$z = \ln a_0 + a_1 x, \text{ com } z = \ln\left(\frac{y}{x}\right) \tag{3.40}$$

Fonte: Elaborado pela própria autora

Agora se pode trabalhar com a regressão linear para encontrar os parâmetros que melhor ajustam o modelo não linear aos dados, assim os resultados obtidos estão na tabela 4, a soma dos quadrados dos resíduos dos dados transformados é: $0.0555$, já dos resíduos dos dados sem transformação é: $0.0212$.

**Tabela 4 – Modelo original e resíduos**

| $x$ | $y$ | Modelo $9.6618xe^{-2.4733x}$ | Resíduo $(9.6618xe^{-2.4733x} - y)$ |
|---|---|---|---|
| 0.1 | 0.75 | 0.7545 | 0.0045 |
| 0.2 | 1.25 | 1.1783 | -0.0717 |
| 0.4 | 1.45 | 1.4370 | -0.0130 |
| 0.6 | 1.25 | 1.3144 | 0.0644 |
| 0.9 | 0.85 | 0.9388 | 0.0888 |
| 1.3 | 0.55 | 0.5042 | -0.0458 |
| 1.5 | 0.35 | 0.3548 | 0.0048 |
| 1.7 | 0.28 | 0.2452 | -0.0348 |
| 1.8 | 0.18 | 0.2027 | 0.0227 |

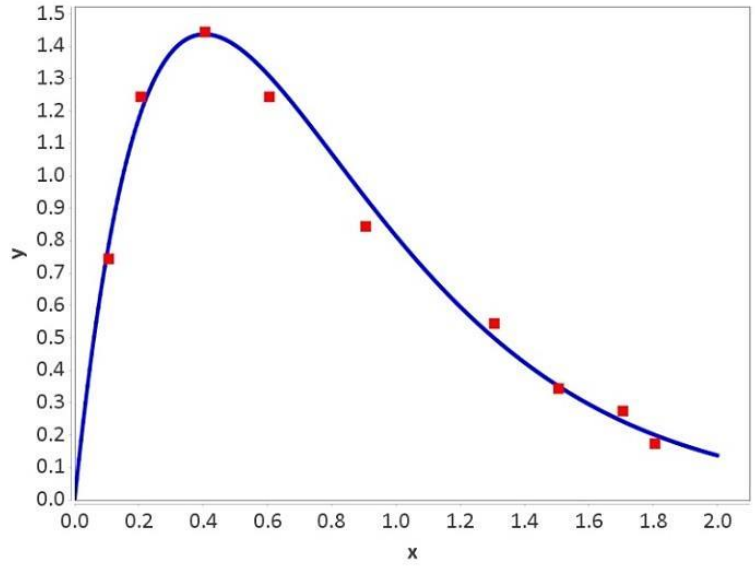
Fonte: Elaborado pela própria autora

**Tabela 5 – Modelo linearizado e resíduos**

| $x$ | $y$ | Modelo linearizado $2.2682 - 2.4733x$ | Resíduo dos dados transformados $(2.2682 - 2.4733x - y)$ |
|---|---|---|---|
| 0.1 | 0.75 | 2.0208 | 0.0059 |
| 0.2 | 1.25 | 1.7735 | -0.0590 |
| 0.4 | 1.45 | 1.2789 | -0.0090 |
| 0.6 | 1.25 | 0.7842 | 0.0502 |
| 0.9 | 0.85 | 0.0422 | 0.0994 |
| 1.3 | 0.55 | -0.9471 | -0.0869 |
| 1.5 | 0.35 | -1.4418 | 0.0135 |
| 1.7 | 0.28 | -1.9364 | -0.1329 |
| 1.8 | 0.18 | -2.1838 | 0.1188 |

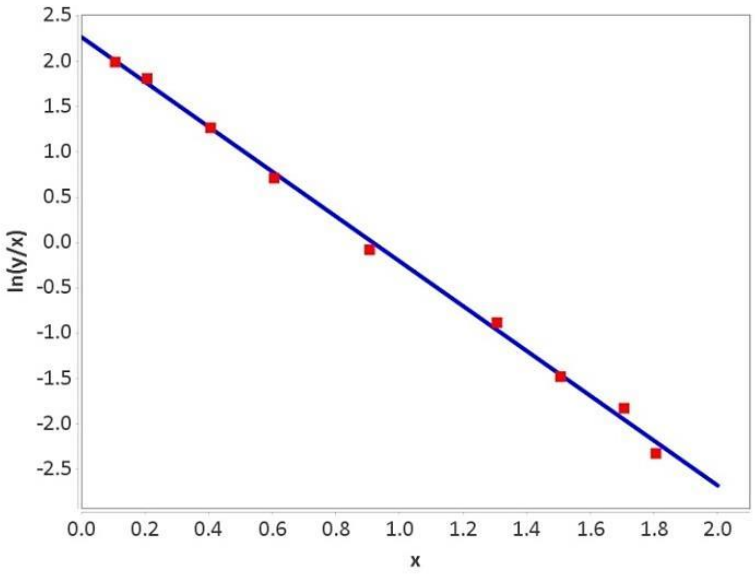
Fonte: Elaborado pela própria autora

**Figura 8 – Curva do modelo original**




Fonte: Elaborado pela própria autora

**Figura 9 – Reta do modelo linearizado**




Fonte: Elaborado pela própria autora

# 4 O MÉTODO DE MÍNIMOS QUADRADOS NÃO LINEARES

No MMQNL a ideia básica é definir uma função que representa a soma dos quadrados dos resíduos e procuram-se os parâmetros que minimizam esta função não linear devido ao ajuste de dados não ser linear nos parâmetros (CUNHA, 2000). “Um exemplo é o ajuste de dados obtidos experimentalmente, usando uma soma de Gaussianas” (CUNHA, 2000, p. 113):

$$f(x) \approx g(x) = \alpha_1 \, exp\left[-\frac{(\alpha_2 - x)^2}{\alpha_3}\right] + \alpha_4 exp\left[-\frac{(\alpha_5 - x)^2}{\alpha_6}\right] \tag{4.1}$$

Fonte: Adaptado de (Cunha, 2000, p. 113)

Em notação, $y(x;\, \alpha_1, \alpha_2, \dots, \alpha_n)$ é a função da variável $x$ e dos parâmetros $\alpha_1, \alpha_2, \dots, \alpha_n$ para o ajuste de um conjunto de dados $y_1, y_2, \dots, y_m$ e $m \gg n$. A função $r_i(\boldsymbol{\alpha})$ é o resíduo no ponto $x_i$ e $\boldsymbol{\alpha}$ é um vetor com parâmetros de ajuste (CUNHA, 2000).

$$r_i(\boldsymbol{\alpha}) = y(x_i; \alpha_1, \alpha_2, \dots, \alpha_n) - y_i \quad para \;\; i = 1{:}\, m \tag{4.2}$$

Fonte: Adaptado de (Cunha, 2000, p. 113)

No problema de minimização da soma dos quadrados dos resíduos, em $F(\boldsymbol{\alpha})$ a divisão por dois é para simplificação de cálculos futuros, se quer minimizar (CUNHA, 2000):

$$F(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i=1}^{m} {r_i}^2(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i=1}^{m} {r_i}^2(\alpha_1, \alpha_2, \dots, \alpha_n) = \frac{1}{2}\|\boldsymbol{r}(\boldsymbol{\alpha})\|^2 = \frac{1}{2}\boldsymbol{r}^T\boldsymbol{r} \tag{4.3}$$

Fonte: Adaptado de (Cunha, 2000, p. 113)

O $\boldsymbol{r}(\boldsymbol{\alpha}) = (r_1(\alpha_1, \alpha_2, \dots, \alpha_n), r_2(\alpha_1, \alpha_2, \dots, \alpha_n), \dots, r_m(\alpha_1, \alpha_2, \dots, \alpha_n))^T$ é um vetor coluna, relembrando o que já foi falado no capítulo 2 o $\|\cdot\|$ representa a norma-2 de um vetor, no caso $\|\boldsymbol{r}(\boldsymbol{\alpha})\| = \sqrt{{r_1}^2(\alpha_1, \alpha_2, \dots, \alpha_n) + \cdots + {r_m}^2(\alpha_1, \alpha_2, \dots, \alpha_n)}$ (MADSEN, NIELSEN, TINGLEFF, 2004).

Nos métodos a serem apresentados (Gauss-Newton e Levenberg-Marquardt), substitui-se $F(\boldsymbol{\alpha})$ por uma aproximação quadrática a ser minimizada em cada passo do método (CUNHA, 2000).

Utilizando a ideia de aproximação, a expansão com os três primeiros termos da série de Taylor em torno de $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \dots, \alpha_n)$ e associada ao incremento $\boldsymbol{h}$ é (MADSEN, NIELSEN, TINGLEFF, 2004):

$$F(\boldsymbol{\alpha} + \boldsymbol{h}) \cong F(\boldsymbol{\alpha}) + \boldsymbol{h}^T F'(\boldsymbol{\alpha}) + \frac{1}{2}\boldsymbol{h}^T F''(\boldsymbol{\alpha})\boldsymbol{h} \tag{4.4}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF, 2004, p. 03)

O termo $F'(\boldsymbol{\alpha})$ é o gradiente e o $F''(\boldsymbol{\alpha})$ é a Hessiana, podem também ser escritos como $\nabla F$ e $\nabla^2 F$ respectivamente. Num minimizador local que será também o minimizador global por ser uma função quadrática $F'(\boldsymbol{\alpha}^*) = 0$ (MADSEN, NIELSEN, TINGLEFF, 2004).

$$F'(\boldsymbol{\alpha}) = \begin{bmatrix} \frac{\partial F}{\partial \alpha_1}(\boldsymbol{\alpha}) \\ \frac{\partial F}{\partial \alpha_2}(\boldsymbol{\alpha}) \\ \vdots \\ \frac{\partial F}{\partial \alpha_n}(\boldsymbol{\alpha}) \end{bmatrix} \tag{4.5}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF, 2004, p. 03)

$$F''(\boldsymbol{\alpha}) = \left[ \frac{\partial^2 F}{\partial \alpha_j \partial \alpha_k}(\boldsymbol{\alpha}) \right] \tag{4.6}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF, 2004, p. 03)

A matriz Jacobiana é uma matriz que contém as derivadas parciais de primeira ordem de uma função vetorial ou pode ser escrita contendo os gradientes transpostos desta função. Neste caso a Jacobiana de $\boldsymbol{r}(\boldsymbol{\alpha})$ de $m$ linhas e $n$ colunas será útil na hora de calcular aproximações para $F'(\boldsymbol{\alpha})$ e $F''(\boldsymbol{\alpha})$ (CUNHA, 2000):

$$\mathbf{J}(\boldsymbol{\alpha}) = \begin{bmatrix} \frac{\partial r_1}{\partial \alpha_1}(\boldsymbol{\alpha}) & \frac{\partial r_1}{\partial \alpha_2}(\boldsymbol{\alpha}) & \cdots & \frac{\partial r_1}{\partial \alpha_n}(\boldsymbol{\alpha}) \\ \frac{\partial r_2}{\partial \alpha_1}(\boldsymbol{\alpha}) & \frac{\partial r_2}{\partial \alpha_2}(\boldsymbol{\alpha}) & \cdots & \frac{\partial r_2}{\partial \alpha_n}(\boldsymbol{\alpha}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_m}{\partial \alpha_1}(\boldsymbol{\alpha}) & \frac{\partial r_m}{\partial \alpha_2}(\boldsymbol{\alpha}) & \cdots & \frac{\partial r_m}{\partial \alpha_n}(\boldsymbol{\alpha}) \end{bmatrix} = \begin{bmatrix} \nabla r_1(\boldsymbol{\alpha})^T \\ \nabla r_2(\boldsymbol{\alpha})^T \\ \vdots \\ \nabla r_m(\boldsymbol{\alpha})^T \end{bmatrix} \tag{4.7}$$

Fonte: Adaptado de (CUNHA, 2000, p. 114)

A derivada parcial de primeira ordem da função $F$ em relação a algum $\alpha_j,\ para\ j = 1{:}\,n$ pode ser obtida com:

$$\frac{\partial F}{\partial \alpha_j}(\boldsymbol{\alpha}) = \sum_{i=1}^{m} r_i(\boldsymbol{\alpha}) \frac{\partial r_i}{\partial \alpha_j}(\boldsymbol{\alpha}) \tag{4.8}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF, 2004, p. 18)

O gradiente passa a ser então:

$$F'(\boldsymbol{\alpha}) = \mathbf{J}^T(\boldsymbol{\alpha}) \boldsymbol{r}(\boldsymbol{\alpha}) \tag{4.9}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF, 2004, p. 18)

Já a Hessiana toma a derivada parcial de segunda ordem da função $F$ em relação aos elementos nas posições $(j, k)$:

$$\frac{\partial^2 F}{\partial \alpha_j \partial \alpha_k} \sum_{i=1}^{m} \left( \frac{\partial r_i(\boldsymbol{\alpha})}{\partial \alpha_j} \frac{\partial r_i(\boldsymbol{\alpha})}{\partial \alpha_k} + r_i(\boldsymbol{\alpha}) \frac{\partial^2 r_i}{\partial \alpha_j \partial \alpha_k}(\boldsymbol{\alpha}) \right) \tag{4.10}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF, 2004, p. 18)

Assim,

$$F''(\boldsymbol{\alpha}) = \mathbf{J}^T(\boldsymbol{\alpha})\mathbf{J}(\boldsymbol{\alpha}) + \sum_{i=1}^{m} r_i(\boldsymbol{\alpha}) r_i\,''(\boldsymbol{\alpha}) \tag{4.11}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF, 2004, p. 18)

O termo $\sum_{i=1}^{m} r_i(\boldsymbol{\alpha}) r_i\,''(\boldsymbol{\alpha})$ pode ser expresso usando uma matriz $Q(\boldsymbol{\alpha})$, e por fim a série de Taylor que é uma função quadrática (CUNHA, 2000):

$$F(\boldsymbol{\alpha} + \boldsymbol{h}) \cong F(\boldsymbol{\alpha}) + \boldsymbol{h}^T F'(\boldsymbol{\alpha}) + \frac{1}{2}\boldsymbol{h}^T F''(\boldsymbol{\alpha})\boldsymbol{h} \tag{4.12}$$

$$F(\boldsymbol{\alpha} + \boldsymbol{h}) \cong F(\boldsymbol{\alpha}) + \boldsymbol{h}^T \mathbf{J}^T(\boldsymbol{\alpha})\boldsymbol{r}(\boldsymbol{\alpha}) + \frac{1}{2}\boldsymbol{h}^T\big(\mathbf{J}^T(\boldsymbol{\alpha})\mathbf{J}(\boldsymbol{\alpha}) + Q(\boldsymbol{\alpha})\big)\boldsymbol{h}$$

Fonte: Adaptado de (CUNHA, 2000, p. 115)

As matrizes $\mathbf{J}^T\mathbf{J}$ **e** $\mathbf{Q}$ tem $n$ linhas e $n$ colunas e ao somá-las o resultado será um número real (CUNHA, 2000).

A próxima etapa do processo de minimização consiste em achar um incremento que reduza a soma dos resíduos, isto é, encontrar $\boldsymbol{h}^k$ em cada passo $k$ (CUNHA, 2000):

$$F(\boldsymbol{\alpha}^k + \boldsymbol{h}^k) < F(\boldsymbol{\alpha}^k) \tag{4.13}$$

Fonte: (CUNHA, 2000, p. 116)

Se isto acontecer a nova aproximação $\boldsymbol{\alpha}^k + \boldsymbol{h}^k$ estará mais perto do mínimo do que $\boldsymbol{\alpha}^k$ (CUNHA, 2000).

Aproximar $F$ por uma função quadrática facilita já que se obtém a nova aproximação minimizando esta função quadrática. É válido tratar essa minimização com base nos métodos dos gradientes conjugados para a resolução de sistemas de equações lineares que funciona de forma iterativa. Na equação (4.14) a função $G(\boldsymbol{x})$ resolve o sistema $A\boldsymbol{x} = \boldsymbol{b}$, quando $grad\ G(\boldsymbol{x}) = 0$ o que equivale a minimizar a função $G(\boldsymbol{x})$. Fazendo uma comparação (CUNHA, 2000):

$$G(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} \tag{4.14}$$

Fonte: (CUNHA, 2000, p. 65)

$$F(\boldsymbol{\alpha}) + \boldsymbol{h}^T \mathbf{J}^T(\boldsymbol{\alpha})\boldsymbol{r}(\boldsymbol{\alpha}) + \frac{1}{2}\boldsymbol{h}^T\big(\mathbf{J}^T(\boldsymbol{\alpha})\mathbf{J}(\boldsymbol{\alpha}) + Q(\boldsymbol{\alpha})\big)\boldsymbol{h} \tag{4.15}$$

Fonte: Adaptado de (CUNHA, 2000, p. 115)

$$A = \mathbf{J}^T\mathbf{J} + \mathbf{Q} \text{ e } \boldsymbol{b} = \mathbf{J}^T\boldsymbol{r} \tag{4.16}$$

Fonte: (CUNHA, 2000, p. 116)

A adição do termo $F(\boldsymbol{\alpha}^k)$ independente de $\boldsymbol{h}$**,** não muda o cálculo do ponto de mínimo porque o termo "representa uma translação vertical da função a ser minimizada." (CUNHA, 2000, p. 116).

Baseando-se no método dos gradientes conjugados a solução do sistema $A\boldsymbol{h} = \boldsymbol{b}$ é a solução da equação $grad\ F(\boldsymbol{h}) = 0$ o qual minimiza a função quadrática. Em cada passo $k$ deve-se resolver o sistema linear (CUNHA, 2000):

$$(\mathbf{J}_k^T\mathbf{J}_k + \mathbf{Q}_k)\boldsymbol{h}_k = -\mathbf{J}_k^T\boldsymbol{r}_k \tag{4.17}$$

Fonte: (CUNHA, 2000, p. 116)

Com o valor de $\boldsymbol{h}$ calculado em $(\mathbf{J}_k^T\mathbf{J}_k + \mathbf{Q}_k)\boldsymbol{h}_k = -\mathbf{J}_k^T\boldsymbol{r}_k$, $\boldsymbol{\alpha}^{k+1}$ será igual a $\boldsymbol{\alpha}^k + \boldsymbol{h}$ (CUNHA, 2000):

$$\boldsymbol{\alpha}^{k+1} = \boldsymbol{\alpha}^k + \boldsymbol{h} \tag{4.18}$$

Fonte: (CUNHA, 2000, p. 116)

As variações nos métodos usados em problemas de MQNL dizem respeito à maneira de resolver o sistema (CUNHA, 2000): $(\mathbf{J}_k^T\mathbf{J}_k + \mathbf{Q}_k)\boldsymbol{h}_k = -\mathbf{J}_k^T\boldsymbol{r}_k$.

## 4.1 Cálculo Discreto da Matriz Jacobiana

A matriz Jacobiana de uma função pode ser calculada de maneira numérica a partir da aproximação da derivada parcial de uma função pela fórmula da diferença central (NOCEDAL, WRIGHT, 2006):

$$\frac{\partial f}{\partial x_i}(\boldsymbol{X}) \approx \frac{f(\boldsymbol{X} + he_i) - f(\boldsymbol{X} - he_i)}{2h} \tag{4.19}$$

Fonte: (NOCEDAL, WRIGHT, 2006, p. 194)

Onde $\boldsymbol{X}$ é o vetor contendo as $n$ variáveis $(x_1, x_2, \dots, x_n)$, $h$ é um valor positivo pequeno e da definição de derivada como um limite, $h$ deve tender a 0, já $e_i$ é um vetor que tem todos os elementos iguais a 0 exceto por um 1 na $i$-ésima posição (NOCEDAL, WRIGHT, 2006).

Adaptando a fórmula para o cálculo da matriz Jacobiana fica assim:

$$\mathbf{J}(i,j) = \frac{\partial r_i}{\partial \alpha_j}(\boldsymbol{\alpha}) \approx \frac{r_i(\boldsymbol{\alpha} + he_j) - r_i(\boldsymbol{\alpha} - he_j)}{2h} \tag{4.20}$$

Fonte: Elaborado pela própria autora

## 4.2 O Método de Gauss-Newton

O método de Gauss-Newton admite que $\boldsymbol{\alpha}^k$ está próximo do mínimo, então o resíduo $\boldsymbol{r}(\boldsymbol{\alpha}^k)$ está próximo de zero. Assim é aceitável dizer que a matriz $\mathbf{Q}$ pode ser desprezada quando comparada com $\mathbf{J}^T\mathbf{J}$ no sistema $\mathbf{J}^T\mathbf{J} + \mathbf{Q} = -\mathbf{J}^T\boldsymbol{r}$. A cada iteração $k$ o sistema a ser resolvido será (CUNHA, 2000):

$$(\mathbf{J}_k^T\mathbf{J}_k)\boldsymbol{h}_k = -\mathbf{J}_k^T\boldsymbol{r}_k \tag{4.21}$$

Fonte: (CUNHA, 2000, p. 117)

Este método também pode ser visto como um método de Newton modificado com busca linear. Em vez de usar as equações Newton padrão é usada a variação na aproximação de $\nabla^2 F$ para melhorar certos problemas encontrados no método de Newton plano (NOCEDAL, WRIGHT, 2006):

$$\nabla F = \mathbf{J}^T \boldsymbol{r} \text{ e } \nabla^2 F \approx \mathbf{J}^T \mathbf{J} \tag{4.22}$$

Fonte: Adaptado de (NOCEDAL, WRIGHT, 2006, p. 254)

Essas aproximações evitam o problema de computar Hessianas dos residuais individuais $\nabla^2 \boldsymbol{r}_j,\ para\ j = 1, 2, \ldots, m$, além disso, com o cálculo da Jacobiana no gradiente $\nabla F = \mathbf{J}^T \boldsymbol{r}$ a aproximação $\nabla^2 F$ não requer cálculos adicionais de derivadas economizando tempo computacional significativo em certas aplicações (NOCEDAL, WRIGHT, 2006).

Em determinado casos o método de Gauss-Newton fornece convergência quadrática como o método de Newton para otimização em geral (MADSEN, NIELSEN, TINGLEFF, 2004).

O algoritmo 4.1 mostra um algoritmo do método de Gauss-Newton.

**Algoritmo 4.1 Método de Gauss-Newton**

**Dados** $r_i, \frac{\partial r_i}{\partial a_j}, tol, \boldsymbol{\alpha}^0, k_{max}$
01: **para** $k := 0 : (k_{max} - 1)$
02: $A := \mathbf{J}^T \mathbf{J}$; g $:= \mathbf{J}^T \boldsymbol{r}$
03: Resolver $A\boldsymbol{h} := -g$
04: $\boldsymbol{\alpha}^{k+1} := \boldsymbol{\alpha}^k + \boldsymbol{h}$
05: **se** $max_{1 \le i \le n} \mid h_i \mid < tol$
06: $\boldsymbol{\alpha} := \boldsymbol{\alpha}^{k+1}$
07: **retornar** $\boldsymbol{\alpha}$
08: **retornar** $\boldsymbol{\alpha}$

Fonte: Adaptado de (CUNHA, 2000 p. 117)

## 4.3 Exemplo de Ajuste de Curva com Gauss-Newton

Um exemplo encontrado em Cunha (2000) é utilizar o MMQNL para realizar o ajuste de dados por uma soma de gaussianas com apenas 3 parâmetros: $f(x) \approx g(x) = \alpha_1\, exp\left[-\frac{(\alpha_2 - x)^2}{\alpha_3}\right]$.

Aplicando o método de Gauss-Newton aos dados fornecidos com chute inicial $\boldsymbol{\alpha}_0 = (1,0,1)$ os valores dos parâmetros encontrados foram $\boldsymbol{\alpha} = (0.4914, 0.1978, 0.7738)$. A soma dos quadrados dos resíduos é 0.0018.

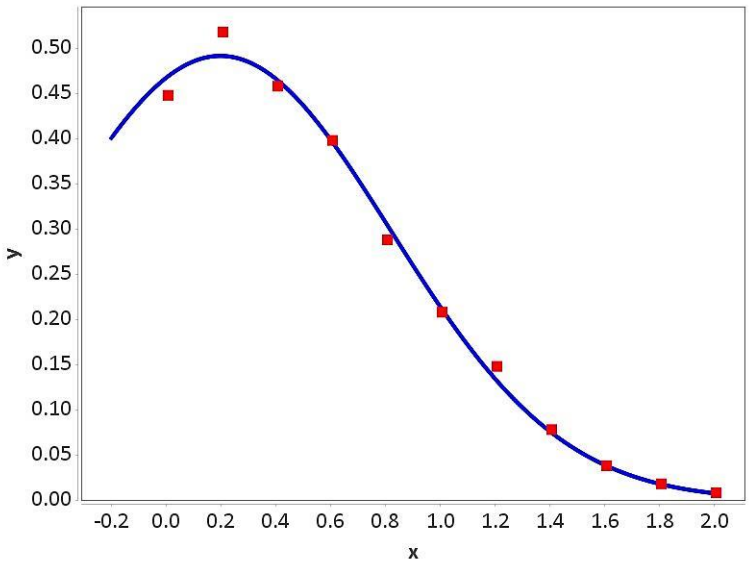
**Tabela 6 – Resíduos com Gauss-Newton**

| $x$ | $y$ | $f(x)$ | Resíduo $(f(x) - y)$ |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 0 | 0.45 | 0.4672 | 0.0172 |
| 0.2 | 0.52 | 0.4914 | -0.0286 |
| 0.4 | 0.46 | 0.4661 | 0.0061 |
| 0.6 | 0.40 | 0.3987 | -0.0013 |
| 0.8 | 0.29 | 0.3075 | 0.0175 |
| 1.0 | 0.21 | 0.2139 | 0.0039 |
| 1.2 | 0.15 | 0.1342 | -0.0158 |
| 1.4 | 0.08 | 0.0759 | -0.0041 |
| 1.6 | 0.04 | 0.0387 | -0.0013 |
| 1.8 | 0.02 | 0.0178 | -0.0022 |
| 2.0 | 0.01 | 0.0074 | -0.0026 |

Fonte: Elaborado pela própria autora

**Figura 10 – Curva obtida pelo método de Gauss-Newton**




Fonte: Elaborado pela própria autora

## 4.4 O Método de Levenberg-Marquardt

O método de Levenberg-Marquardt sugerido por Levenberg (1944) e mais tarde Marquardt (1963) é um método de Gauss-Newton amortecido e ao invés de usar a busca linear trabalha com uma estratégia de região de confiança com a mesma aproximação Hessiana (MADSEN, NIELSEN, TINGLEFF, 2004). Quando a matriz Jacobiana $\mathbf{J}(\boldsymbol{\alpha})$ tem um posto deficiente ou próximo disto, a estratégia de região de confiança se adequa melhor do que a usada pelo método de Newton. As propriedades de convergência local dos dois métodos são parecidas desde que as mesmas aproximações Hessianas sejam usadas (NOCEDAL, WRIGHT, 2006).

A seguinte modificação é introduzida no sistema $(\mathbf{J}_k^T\mathbf{J}_k)\boldsymbol{h}_k = -\mathbf{J}_k^T\boldsymbol{r}_k$ resolvido a cada iteração $k$, o número $\mu$ é um número real não negativo e a matriz $\mathbf{I}$ representa a matriz identidade de $n$ linhas por $n$ colunas (CUNHA, 2000):

$$(\mathbf{J}_k^T\mathbf{J}_k + \mu\mathbf{I})\boldsymbol{h}_k = -\mathbf{J}_k^T\boldsymbol{r}_k \tag{4.23}$$

Fonte: (CUNHA, 2000, p. 118)

A matriz dada por $(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})\boldsymbol{h}$ para $\mu > 0$ é uma matriz simétrica e positiva definida (MADSEN, NIELSEN, TINGLEFF, 2004).

"O parâmetro amortecedor $\mu$ tem vários efeitos" (MADSEN, NIELSEN, TINGLEFF, 2004, p. 23):

- Para todo $\mu > 0$ é garantido que $\boldsymbol{h}$ é uma direção de descida em virtude de a matriz dos coeficientes ser positiva definida (MADSEN, NIELSEN, TINGLEFF, 2004).
- Para grandes valores de $\mu$: $\boldsymbol{h} = -\frac{1}{\mu}F'(\boldsymbol{\alpha})$ é um pequeno incremento em uma direção íngreme de descida e é bom quando a iteração corrente está longe da solução (MADSEN, NIELSEN, TINGLEFF, 2004).
- Caso $\mu$ seja muito pequeno, $\boldsymbol{h}^{LM} \cong \boldsymbol{h}^{GN}$, isto é bom nas iterações finais, quando $\boldsymbol{\alpha}$ está perto de $\boldsymbol{\alpha}^*$ (MADSEN, NIELSEN, TINGLEFF, 2004).

O parâmetro $\mu$ influencia tanto a direção de busca quanto o tamanho do incremento. O valor inicial de $\mu$ é calculado a partir de $A_0 = \mathbf{J}^T(\boldsymbol{\alpha}^0)\mathbf{J}(\boldsymbol{\alpha}^0)$ (MADSEN, NIELSEN, TINGLEFF, 2004):

$$\mu_0 = \tau \cdot max\left\{a_{ii}^{(0)}\right\} \tag{4.24}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF, 2004, p. 25)

O valor de $\tau$ é escolhido pelo usuário e não faz tanta diferença para o algoritmo, via de regra deve ser um valor pequeno, $\tau = 10^{-6}$ se $\boldsymbol{\alpha}^0$ é uma boa aproximação para $\boldsymbol{\alpha}^*$**,** senão tomar $\tau = 10^{-3}$ e até mesmo $\tau = 1$ (MADSEN, NIELSEN, TINGLEFF, 2004).

O tamanho de $\mu$ é atualizado em cada iteração pela razão de ganho $\rho$ (MADSEN, NIELSEN, TINGLEFF, 2004):

$$\rho = \frac{F(\boldsymbol{\alpha}) - F(\boldsymbol{\alpha} + \boldsymbol{h})}{L(0) - L(\boldsymbol{h})} \tag{4.25}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF, 2004, p. 25)

O denominador trata-se de um ganho previsto pelo modelo linear já apresentado no método de Gauss-Newton (MADSEN, NIELSEN, TINGLEFF, 2004):

$$L(\boldsymbol{h}) \cong F(\boldsymbol{\alpha} + \boldsymbol{h}) \equiv F(\boldsymbol{\alpha}) + \boldsymbol{h}^T\mathbf{J}^T\boldsymbol{r}(\boldsymbol{\alpha}) + \frac{1}{2}\boldsymbol{h}^T\mathbf{J}^T(\boldsymbol{\alpha})\mathbf{J}(\boldsymbol{\alpha})\boldsymbol{h} \tag{4.26}$$

Fonte: Adaptado de (MADSEN, NIELSEN, TINGLEFF, 2004, p. 21)

A modificação introduzida pelo método de Levenberg-Marquardt leva a:

$$(\mathbf{J}^T\mathbf{J} + \mu\mathbf{I})\boldsymbol{h} = -g \; com \; g = \mathbf{J}^T\boldsymbol{r} \tag{4.27}$$

Fonte: Adaptado de (MADSEN, NIELSEN, TINGLEFF, 2004, p. 24)

O denominador para a razão de ganho $\rho$ então: (MADSEN, NIELSEN, TINGLEFF, 2004):

$$\begin{aligned} L(0) - L(\boldsymbol{h}) &= -\boldsymbol{h}^{\boldsymbol{T}}\mathbf{J}^T\boldsymbol{r} - \tfrac{1}{2}\boldsymbol{h}^T\mathbf{J}^T\mathbf{J}\boldsymbol{h} \\ &= -\frac{1}{2}\boldsymbol{h}^{\boldsymbol{T}}(2g + (\mathbf{J}^T\mathbf{J} + \mu\mathbf{I} - \mu\mathbf{I})\boldsymbol{h}) \\ L(0) - L(\boldsymbol{h}) &= \frac{1}{2}\boldsymbol{h}^T(\mu\boldsymbol{h} - g) \end{aligned} \tag{4.28}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF, 2004, p. 25)

A variável $\rho$ serve para indicar o quão boa é a aproximação de $L(\boldsymbol{h})$ para $F(\boldsymbol{\alpha} + \boldsymbol{h})$, valores grandes indicam uma boa aproximação e o valor de $\mu$ deve ser reduzido para aproximar o incremento do Levenberg-Marquardt do Gauss-Newton. Porém se o valor de $\rho$ é pequeno ou até mesmo negativo $L(\boldsymbol{h})$ é uma pobre aproximação, deve-se então aumentar o valor de $\mu$ para aproximar-se da direção de descida mais íngreme e reduzir o tamanho do passo (MADSEN, NIELSEN, TINGLEFF, 2004). Uma boa estratégia mostrada em (MADSEN, NIELSEN, TINGLEFF, 2004) *apud* (NIELSEN, 1999) está no algoritmo 4.2.

**Algoritmo 4.2 Estratégia de atualização do parâmetro** $\mu$

**se** $\rho > 0$
    $\mu \coloneqq \mu \times max\left\{\frac{1}{3}, (2\rho - 1)^3\right\}; v \coloneqq 2$
**senão**
    $\mu \coloneqq \mu \times v; v \coloneqq 2v$

Fonte: (MADSEN, NIELSEN, TINGLEFF, 2004, p. 14)

O critério de parada do algoritmo deve levar em conta que em um minimizador global $F'(\boldsymbol{\alpha}^*) = g(\boldsymbol{\alpha}^*) = 0$, então neste caso pode se estabelecer uma tolerância $\varepsilon_1$, positiva e de valor pequeno a ser definida pelo usuário (MADSEN, NIELSEN, TINGLEFF, 2004):

$$\|g\|_\infty \leq \varepsilon_1 \tag{4.29}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF,2004, p. 26)

Outro critério de parada é quando de uma iteração para outra a mudança no $\boldsymbol{\alpha}$ é pequena (MADSEN, NIELSEN, TINGLEFF, 2004):

$$\|\boldsymbol{\alpha}_{new} - \boldsymbol{\alpha}\| \leq \varepsilon_2(\|\boldsymbol{\alpha}\| + \varepsilon_2) \tag{4.30}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF,2004, p. 26)

O algoritmo é um processo iterativo então para evitar um loop infinito é necessário colocar um número máximo de iterações: $k_{max}$. Parar quando (MADSEN, NIELSEN, TINGLEFF, 2004):

$$k \geq k_{max} \tag{4.31}$$

Fonte: (MADSEN, NIELSEN, TINGLEFF,2004, p. 26)

O valor de $\varepsilon_1$, $\varepsilon_2$ e de $k_{max}$ são escolhidos pelo usuário. Os dois últimos critérios de parada são afetados se $\varepsilon_1$ é escolhido tão pequeno que erros de arredondamento influenciam muito. Com isso na razão de ganho $\rho$ o valor do ganho em $F$ e o ganho $L$ previsto pelo modelo linear tem pouca concordância levando o aumento de $\mu$ em cada passo. Aumentar $\mu$ de acordo com o algoritmo 4.2 o faz crescer rápido, resultando em uma pequena $\|\boldsymbol{h}\|$ e o processo será parado por $\|\boldsymbol{\alpha}_{new} - \boldsymbol{\alpha}\| \leq \varepsilon_2(\|\boldsymbol{\alpha}\| + \varepsilon_2)$ (MADSEN, NIELSEN, TINGLEFF, 2004). No algoritmo 4.3 o método de Levenberg-Marquardt.

**Algoritmo 4.3 Método de Levenberg-Marquardt**

**Dados** $r_i, \frac{\partial r_i}{\partial a_j}, \varepsilon_1, \varepsilon_2, \boldsymbol{\alpha}^0, k_{max}$
01: $k := 0$; $v := 2$; $\boldsymbol{\alpha} := \boldsymbol{\alpha}^0$
02: $A := \mathbf{J}^T\mathbf{J}$; $g := \mathbf{J}^T\boldsymbol{r}$
03: $encontrado := \|g\|_\infty \leq \varepsilon_1$; $\mu := \tau \cdot max\{a_{ii}\}$
04: **enquanto** (**não** $encontrado$) **e** ($k < k_{max}$)
05: $k := k + 1$; Resolver $(A + \mu\mathbf{I})\boldsymbol{h} := -g$
06: **se** $\|\boldsymbol{h}\| \leq \varepsilon_2(\|\boldsymbol{\alpha}\| + \varepsilon_2)$
07: $encontrado := \boldsymbol{verdadeiro}$
08: **senão**
09: $\boldsymbol{\alpha}_{new} := \boldsymbol{\alpha} + \boldsymbol{h}$
10: $\rho := (F(\boldsymbol{\alpha}) - F(\boldsymbol{\alpha} + \boldsymbol{h}))/(L(0) - L(\boldsymbol{h}))$
11: **se** $\rho > 0$ {passo aceitável}
12: $\boldsymbol{\alpha} := \boldsymbol{\alpha}_{new}$
13: $A := \mathbf{J}^T\mathbf{J}$; $g := \mathbf{J}^T\boldsymbol{r}$
14: $encontrado := \|g\|_\infty \leq \varepsilon_1$
15: $\mu := \mu \times max\left\{\frac{1}{3}, (2\rho - 1)^3\right\}$; $v := 2$
16: **senão**
17: $\mu := \mu \times v$; $v := 2v$
18: **retornar** $\boldsymbol{\alpha}$

Fonte: (MADSEN, NIELSEN, TINGLEFF,2004, p. 27)

## 4.5 O Método de Levenberg-Marquardt com Restrições do tipo Caixa

Em muitos problemas práticos que podem ser resolvidos por MQNL as variáveis estão sujeitas a restrições do tipo caixa, ou seja, todas as variáveis precisam estar entre um limite inferior (*lower bound*) e um limite superior (*upper bound*) da seguinte forma:

$$l_j \leq \alpha_j \leq u_j, \quad j = 1,2,\dots,n \tag{4.32}$$

Fonte: Adaptado de (NASH, 2014, p. 134)

Neste caso é possível utilizar uma transformação de variáveis para que se continue dentro dos limites impostos pelo intervalo. Em Nash (2014) utiliza-se uma transformação

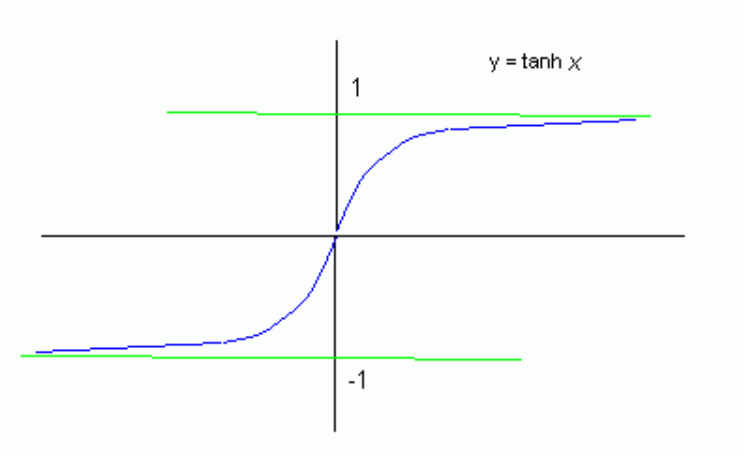
hiperbólica com a função $\tanh(\alpha_j)$, os seus valores estão no intervalo $]-1,1[$ e o parâmetro $\alpha'_j$ que está restrito pode ser obtido a partir de $\alpha_j$ da seguinte maneira:

$$\alpha'_j = l_j + 0.5(u_j - l_j)(1 + \tanh(\alpha_j)) \tag{4.33}$$

Fonte: (NASH, 2014, p. 134)

A expressão (4.33) equivale a fazer uma correspondência biunívoca entre os dois intervalos, o $\alpha'_j$ deve pertencer ao intervalo $[l_j, u_j]$ enquanto que os valores de $\tanh(\alpha_j)$ estão no intervalo $]-1,1[$.

**Figura 11 – Gráfico de** $\tanh(x)$




Fonte: (GOROSTIZAGA, 2007)

A partir disso:

$$\tanh(\alpha_j) \rightarrow ]-1,1[$$

$$\alpha'_j \rightarrow [l_j, u_j]$$

$$\alpha'_j - l_j = \frac{u_j - l_j}{1-(-1)}\big(\tanh(\alpha_j) - (-1)\big)$$

$$\alpha'_j - l_j = \frac{u_j - l_j}{2}(\tanh(\alpha_j) + 1) \tag{4.34}$$

$$\alpha'_j - l_j = 0.5(u_j - l_j)(\tanh(\alpha_j) + 1)$$

$$\alpha'_j = l_j + 0.5(u_j - l_j)(\tanh(\alpha_j) + 1)$$

Fonte: Elaborado pela própria autora

Reescrevendo a expressão:

$$\alpha'_j = \frac{2l_j}{2} + \frac{u_j - l_j}{2}(1 + \tanh(\alpha_j))$$

$$\alpha'_j = \frac{2l_j}{2} + \frac{u_j - l_j}{2} + \frac{u_j - l_j}{2}\tanh(\alpha_j) \tag{4.35}$$

$$\alpha'_j = \frac{u_j + l_j}{2} + \frac{u_j - l_j}{2}\tanh(\alpha_j)$$

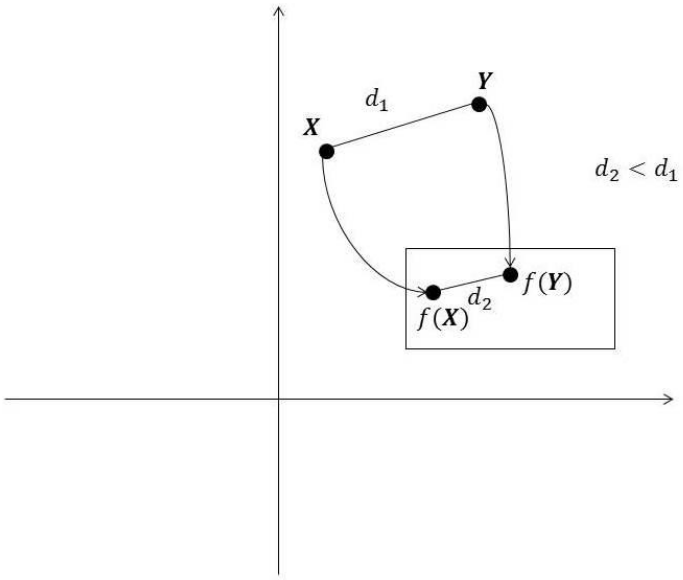
Fonte: Elaborado pela própria autora

Esta é a forma utilizada na implementação em C/C++ do MINPACK realizada por Frédéric Devernay. O MINPACK é uma implementação *open-source* do algoritmo de Levenbeg-Marquardt feita originalmente em FORTRAN por Jorge Moreé, Burt Garbow, e Ken Hillstrom do Laboratório Nacional de Argonne.

Devernay (2007) utiliza $\tanh\left(\left(\alpha_j - \alpha middle\right)/\alpha width\right)$, com $\alpha middle = \left(\frac{u_j+l_j}{2}\right)$ e $\alpha width = \left(\frac{u_j-l_j}{2}\right)$ ao invés de simplesmente $\tanh(\alpha_j)$, para preservar a escala de variáveis e por ser quase a identidade próxima do meio do intervalo.

A ideia da transformação com tangente hiperbólica é fazer com que os pontos do espaço de busca sofram uma contração para ficarem no interior da caixa incluindo as bordas dela. Seja $\Omega \subset \mathbb{R}^n$ um espaço aberto então a aplicação contínua $f$ de $\Omega$ será uma contração se a métrica euclidiana $d$ obedece a seguinte relação $d(f(\boldsymbol{X}), f(\boldsymbol{Y})) < d(\boldsymbol{X}, \boldsymbol{Y}), \forall \boldsymbol{X} \neq \boldsymbol{Y}$, $\boldsymbol{X} = (x_1, x_2, \dots, x_n)$ e $\boldsymbol{Y} = (y_1, y_2, \dots, y_n)$ em $\Omega$. Com o sinal $<$ a contração é dita forte como ocorre no caso da transformação com tangente hiperbólica (LIMA, 1970).

**Figura 12 – Contração**




Fonte: Elaborado pela própria autora

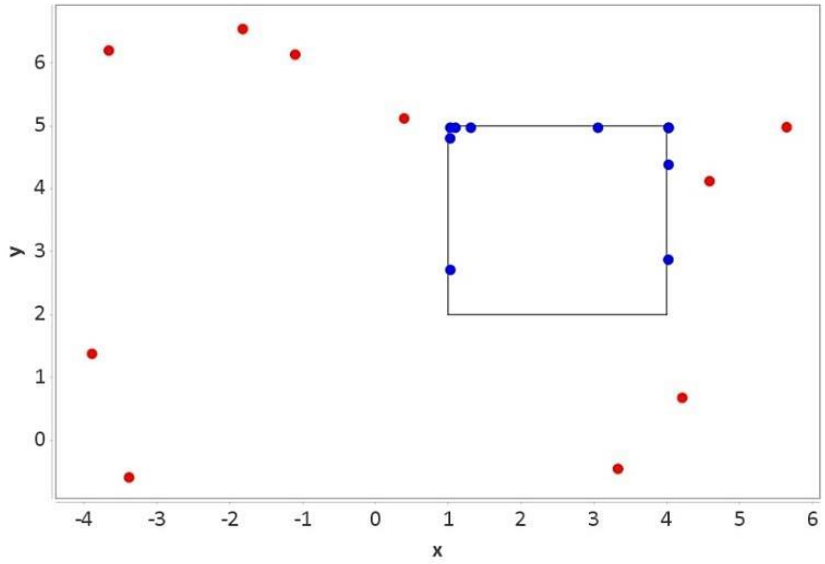
É possível visualizar nas figuras 13 e 14 o que acontece ao usar a fórmula dada por Nash (2014): $\alpha'_j = \frac{u_j+l_j}{2} + \frac{u_j-l_j}{2}\tanh(\alpha_j)$ para restringir o valor de $\alpha_j$ em comparação com a fórmula utilizada por Devernay (2007):

$$\alpha'_j = \frac{u_j+l_j}{2} + \frac{u_j-l_j}{2}\tanh\left(\frac{\left(\alpha_j - \left(\frac{u_j+l_j}{2}\right)\right)}{\left(\frac{u_j-l_j}{2}\right)}\right) \tag{4.36}$$

Fonte: Elaborado pela própria autora

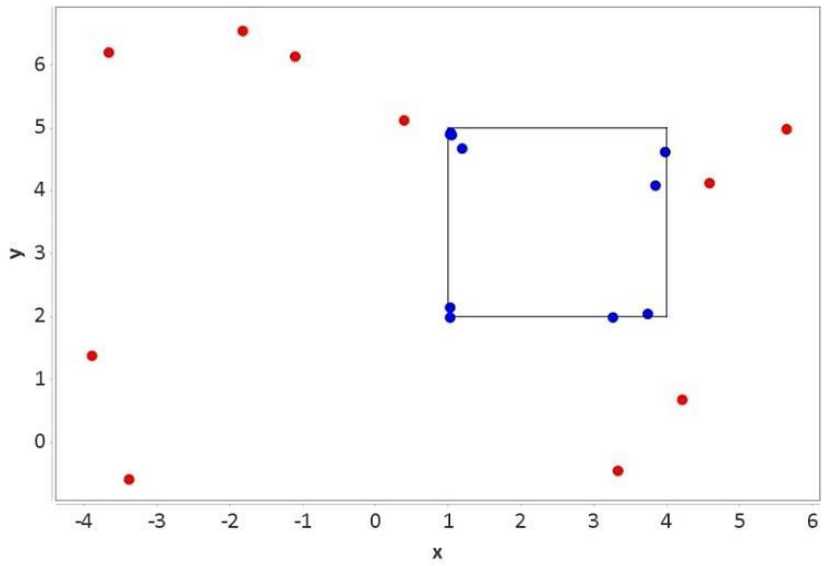
Ambas as fórmulas levam os pontos para dentro da caixa, mas os pontos com a fórmula colocada por Nash (2014) tem maior tendência a irem para as bordas da caixa enquanto ao usar a fórmula utilizada por Devernay (2007) eles ficam melhores ajustados dentro da caixa.

**Figura 13 – Fórmula colocada por Nash e a tendência de irem para as bordas da caixa**




Fonte: Elaborado pela própria autora

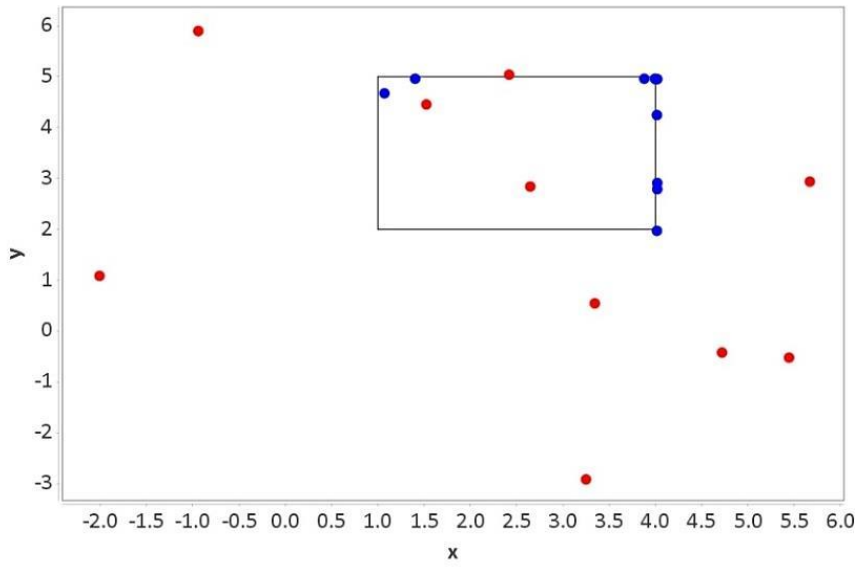
**Figura 14 – Fórmula utilizada por Devernay e o melhor ajuste dos pontos**




Fonte: Elaborado pela própria autora

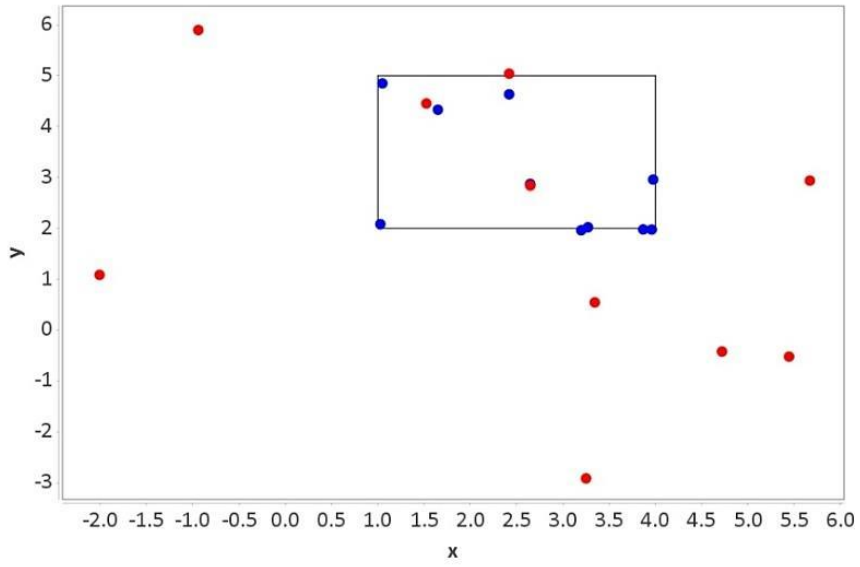
Os pontos que já estão dentro da caixa podem ser mapeados para a mesma posição ou numa outra posição ainda dentro da caixa, como ocorre nas figuras 15 e 16.

**Figura 15 – Fórmula colocada por Nash e mapeamento de pontos dentro da caixa**




Fonte: Elaborado pela própria autora

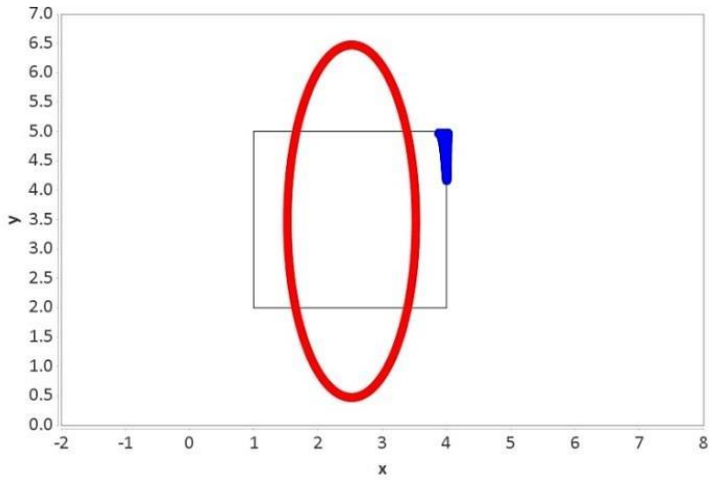
**Figura 16 – Fórmula colocada por Devernay e mapeamento de pontos dentro da caixa**




Fonte: Elaborado pela própria autora

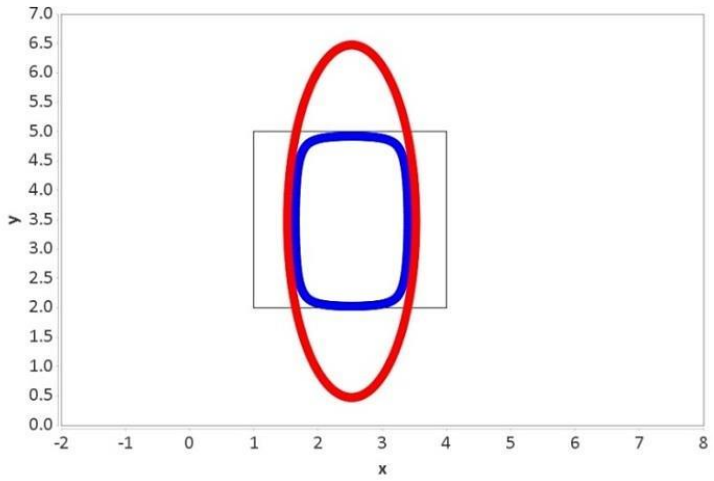
Com a elipse nas figuras 17 e 18 a fórmula dada por Nash (2014) aglutina os pontos da elipse próximos ao canto superior direito da caixa o que não é desejável enquanto que com a fórmula usada por Devernay (2007) faz os pontos da elipse se situarem no interior da caixa, sendo envolvidos pela primeira.

**Figura 17 – Mapeamento dos pontos da elipse com a fórmula colocada por Nash**




Fonte: Elaborado pela própria autora

**Figura 18 – Mapeamento dos pontos da elipse com a fórmula utilizada por Devernay**




Fonte: Elaborado pela própria autora

Devernay (2007) explica como realizar a simulação de restrições do tipo caixa, antes de iniciar qualquer mudança de variáveis é necessário colocar o chute inicial no intervalo, a mudança de variáveis ocorre logo antes de repassar as variáveis para a função que irá realizar o cálculo dos valores de cada um dos resíduos $r_i(\boldsymbol{\alpha}) = y(x_i; \alpha_1, \alpha_2, \dots, \alpha_n) - y_i \quad para \ i = 1{:}\,m$ e depois da otimização ter sido calculada. É necessário fazer um ajuste da matriz Jacobiana devido às derivadas terem de se adequar à nova função, então basta multiplicar cada coluna da matriz Jacobiana já calculada com cada $\alpha'_j$, pelo fator $jacfac_j = 1 - \tanh^2\left(\frac{(\alpha_j - \alpha middle)}{\alpha width}\right), \ j = 1{:}\,n.$

O fator $jacfac_j$ se deve à derivada de $\tanh(x)$ ser $\operatorname{sech}^2(x) = 1 - \tanh^2(x)$. (THOMAS, 2009). Então para exemplificar suponha que uma função $f = x^2 + 1$ com $\frac{df}{dx} = 2x$ se torne agora $f_{new} = u^2 + 1$ com $u = \tanh(x)$:

$$\begin{aligned} &\frac{df_{new}}{du} = 2u \\ &df_{new} = 2u \cdot du \\ &\frac{du}{dx} = 1 - \tanh^2(x) \\ &du = 1 - \tanh^2(x) \cdot dx \\ &df_{new} = 2u \cdot (1 - \tanh^2(x))dx \\ &df_{new} = 2 \tanh(x) \cdot (1 - \tanh^2(x))dx \end{aligned} \tag{4.37}$$

Fonte: Elaborado pela própria autora

Logo, obtém-se:

$$\frac{df_{new}}{dx}(x) = \frac{df}{dx}(\tanh(x)) \times (1 - \tanh^2(x)) \tag{4.38}$$

Fonte: Elaborado pela própria autora

No algoritmo 4.4 segue o pseudocódigo.

**Algoritmo 4.4 Transformação para restrições do tipo caixa**

01: **para** $j \coloneqq 1{:}\, m$
02: $\alpha middle \coloneqq \frac{u_j + l_j}{2}$
03: $\alpha width \coloneqq \frac{u_j - l_j}{2}$
04: $th \coloneqq \tanh\left(\left(\alpha_j - \alpha middle\right)/\alpha width\right)$
05: $\alpha'_j \coloneqq \alpha middle + th \times \alpha width$
06: $jacfac_j \coloneqq 1 - th \times th$

Fonte: Adaptado de (DEVERNAY, 2007)

## 4.6 Exemplo de Ajuste de Curva com Levenberg-Marquardt

Outra situação seria ajustar um conjunto de dados a seguinte curva $f(x) \approx g(x) = \alpha_1 \cos(\alpha_2 x + \alpha_3) + \alpha_4$ usando agora o método de Levenberg-Marquardt sem restrições e com restrições do tipo caixa. O chute inicial é $\boldsymbol{\alpha}_0 = (2, 0.8, 1.5{,}1)$ e os parâmetros encontrados sem restrições foram: $\boldsymbol{\alpha} = (-1.8440, 0.9646, 0.0764, 2.9649)$. A soma dos quadrados dos resíduos é 1.0829.

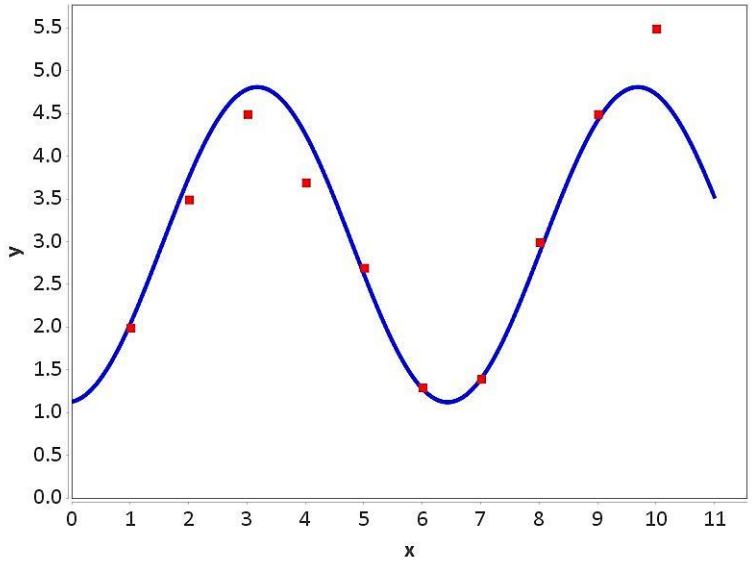
**Tabela 7 – Resíduos com Levenberg-Marquardt sem restrições**

| $x$ | $y$ | $f(x)$ | Resíduo $(f(x) - y)$ |
|---|---|---|---|
| 1 | 2 | 2.0329 | 0.0329 |
| 2 | 3.5 | 3.7415 | 0.2415 |
| 3 | 4.5 | 4.7819 | 0.2819 |
| 4 | 3.7 | 4.2589 | 0.5589 |
| 5 | 2.7 | 2.6225 | -0.0775 |
| 6 | 1.3 | 1.2807 | -0.0193 |

| | | | |
|---|---|---|---|
| 7 | 1.4 | 1.3881 | -0.0119 |
| 8 | 3 | 2.8522 | -0.1478 |
| 9 | 4.5 | 4.4133 | -0.0867 |
| 10 | 5.5 | 4.7281 | -0.7719 |

Fonte: Elaborado pela própria autora

**Figura 19 – Curva obtida pelo Levenberg-Marquardt sem restrições**




Fonte: Elaborado pela própria autora

Agora colocando restrições do tipo caixa, com o mesmo chute inicial os parâmetros encontrados foram: $\boldsymbol{\alpha} = (1.8440, 0.9646, 3.2180, 2.9649)$, elas conduziram a outra solução para o problema com os mesmos resíduos da anterior sem restrições.

$$\begin{aligned} 0 \leq\ & \alpha_1 \leq 2 \\ 0 \leq\ & \alpha_2 \leq 4 \\ 0 \leq\ & \alpha_3 \leq 4 \\ 1 \leq\ & \alpha_4 \leq 3 \end{aligned} \tag{4.39}$$

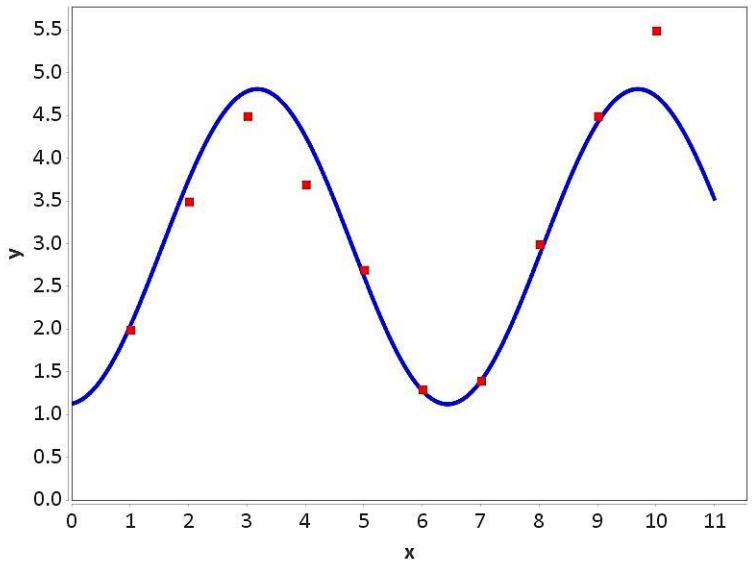
Fonte: elaborado pela própria autora

**Tabela 8 – Resíduos com Levenberg-Marquardt e restrições do tipo caixa**

| $x$ | $y$ | $f(x)$ | Resíduo $(f(x) - y)$ |
|---|---|---|---|
| 1 | 2 | 2.0329 | 0.0329 |
| 2 | 3.5 | 3.7415 | 0.2415 |
| 3 | 4.5 | 4.7819 | 0.2819 |
| 4 | 3.7 | 4.2589 | 0.5589 |
| 5 | 2.7 | 2.6225 | -0.0775 |
| 6 | 1.3 | 1.2807 | -0.0193 |
| 7 | 1.4 | 1.3881 | -0.0119 |
| 8 | 3 | 2.8522 | -0.1478 |
| 9 | 4.5 | 4.4133 | -0.0867 |
| 10 | 5.5 | 4.7281 | -0.7719 |

Fonte: Elaborado pela própria autora

**Figura 20 – Curva obtida usando Levenberg-Marquardt com restrições do tipo caixa**




Fonte: Elaborado pela própria autora

## 4.7 Sistemas de Equações Não Lineares

Em (NOCEDAL, WRIGHT, 2006) num sistema de equações não lineares sem estar relacionado a ajuste de curva os resíduos do MMQNL são representados por funções não lineares $f_1(x_1, x_2, \dots, x_n), f_2(x_1, x_2, \dots, x_n), \dots, f_m(x_1, x_2, \dots, x_n)$:

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \vdots \quad\quad\quad\quad\quad\quad\quad\quad \vdots \quad \vdots \\ f_m(x_1, x_2, \dots, x_n) = 0 \end{cases} \tag{4.40}$$

Fonte: Adaptado de (CUNHA, 2000, p. 81)

O Levenberg-Marquardt será utilizado para resolver o seguinte sistema de equações não lineares, os residuais são dados por $f_1 = x^2 + 3y^2 - 100$, $f_2 = 5x + y^3 - 200$ e $f_3 = x^3 + 2xy - 50$.

$$\begin{cases} x^2 + 3y^2 = 100 \\ 5x + y^3 = 200 \\ x^3 + 2xy = 50 \end{cases} \tag{4.41}$$

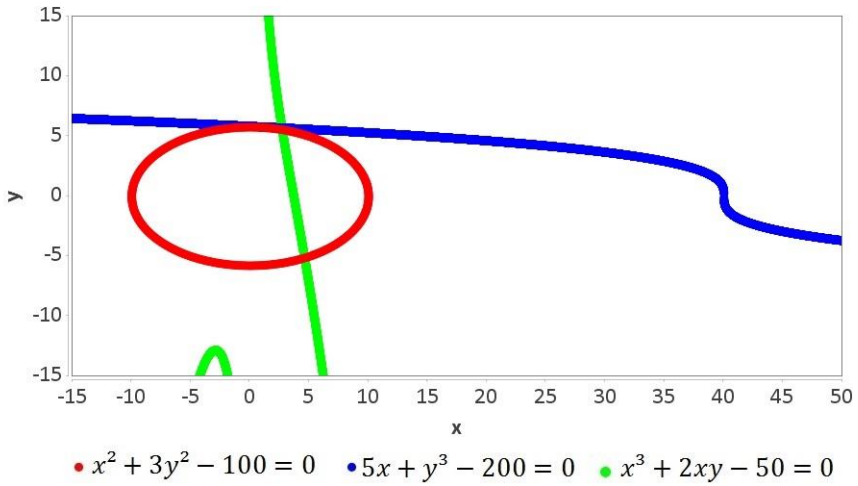
Fonte: Elaborado pela própria autora

Então se deseja:

$$\min((x^2 + 3y^2 - 100)^2 + (5x + y^3 - 200)^2 + (x^3 + 2xy - 50)^2) \tag{4.42}$$

Fonte: Elaborado pela própria autora

**Figura 21 – Sistema de equações não lineares**




Fonte: Elaborado pela própria autora

Os resultados obtidos sem restrições e com restrições do tipo caixa são mostrados nas tabelas 9 e 10.

**Tabela 9 – Resultados sem restrições**

| | | | |
|---|---|---|---|
| $\boldsymbol{X}_0 = (5,4)$ | $x = 2.6710$ e $y = 5.6988$ | $f1 = 4.5616$<br>$f2 = -1.5731$<br>$f3 = -0.5031$ | $\sum_{i=1}^{m} {f_i}^2(\boldsymbol{X}) = 23.5358$ |

Fonte: Elaborado pela própria autora

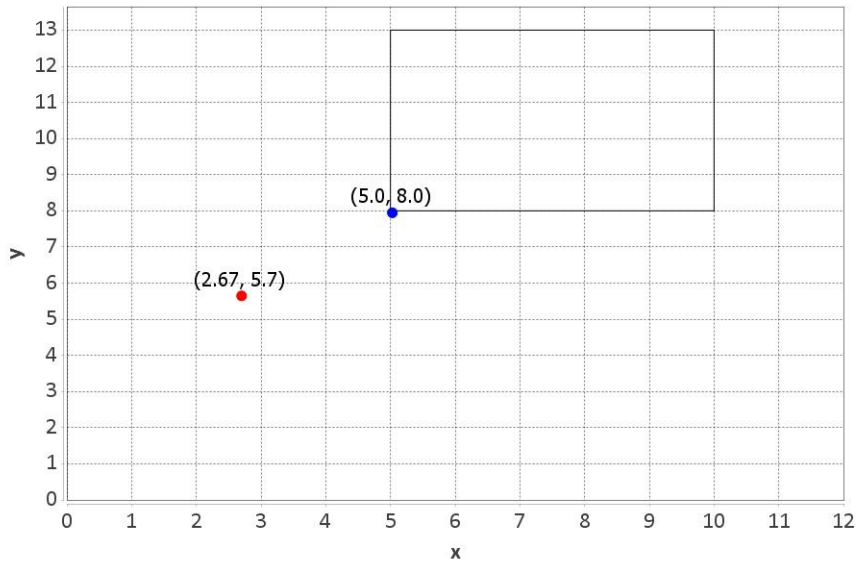
**Tabela 10 – Resultados com restrições do tipo caixa**

| | | |
|---|---|---|
| $\boldsymbol{X}_0 = (5,4)$<br>$0 \leq x \leq 10$<br>$0 \leq y \leq 10$ | $x = 2.6710$ e $y = 5.6988$ | $\sum_{i=1}^{m} {f_i}^2(\boldsymbol{X}) = 23.5358$ |
| $\boldsymbol{X}_0 = (5,4)$<br>$4 \leq x \leq 10$<br>$5 \leq y \leq 13$ | $x = 4.0$ e $y = 5.5593$ | $\sum_{i=1}^{m} {f_i}^2(\boldsymbol{X}) = 3562.2484$ |
| $\boldsymbol{X}_0 = (5,4)$<br>$5 \leq x \leq 10$<br>$8 \leq y \leq 13$ | $x = 5.0$ e $y = 8.0$ | $\sum_{i=1}^{m} {f_i}^2(\boldsymbol{X}) = 151283.0$ |

Fonte: Elaborado pela própria autora

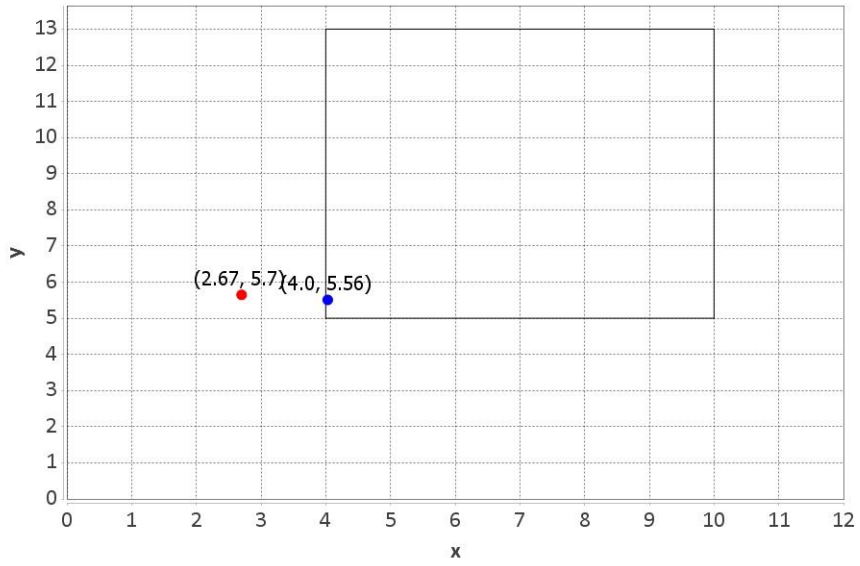
Fazendo uma comparação da solução sem restrições com as soluções com restrições do tipo caixa, pode se notar que a solução na terceira linha da tabela 10 foi para o ponto mais próximo de $(2.6710, 5.6988)$, já na segunda linha isto não ocorreu. Isso está exposto nas figuras 22 e 23, a solução sem restrições $(2.6710, 5.6988)$ está em vermelho, as soluções com restrições do tipo caixa estão em azul.

**Figura 22 – Solução foi para o ponto mais próximo**




Fonte: Elaborado pela própria autora

**Figura 23 – Solução não foi para o ponto mais próximo**



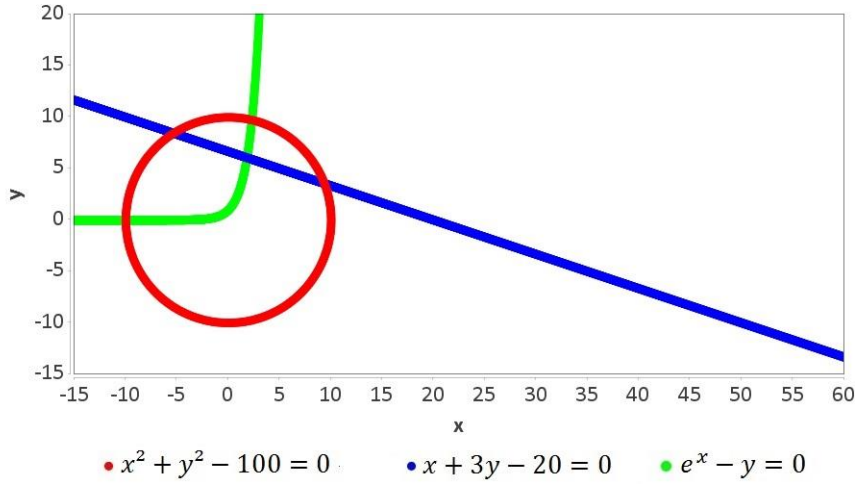

Fonte: Elaborado pela própria autora

Também outro fato que se nota é que implementar as restrições do tipo caixa não é a mesma coisa que calcular os valores de $\boldsymbol{X}$ sem restrições e depois apenas verificar se os valores estão dentro do intervalo das restrições ou não, se não estiverem substituir pelo limite mais próximo. Caso isso fosse válido a segunda linha da tabela 10 teria como solução $x = 4.0$ e $y = 5.6988$ e não $x = 4.0$ e $y = 5.5593$, já que as restrições são $4 \leq x \leq 10$ e $5 \leq y \leq 13$ e a solução sem restrições é $x = 2.6710$ e $y = 5.6988$. Porém a solução $x = 4.0$ e $y = 5.6988$ pioraria a soma dos quadrados dos resíduos de 3562.2484 para 3757.1192.

Observa-se que quase sempre não há um ponto em comum entre as curvas então o Levenberg-Marquardt, assim como outras variantes do MMQNL, tenta encontrar uma solução

de modo que o valor de cada função não linear se aproxime de 0 com a ideia de minimizar a soma dos quadrados dos resíduos, está de maneira bem visível na figura 24.

**Figura 24 – Sem um ponto de interseção entre todas as três curvas**



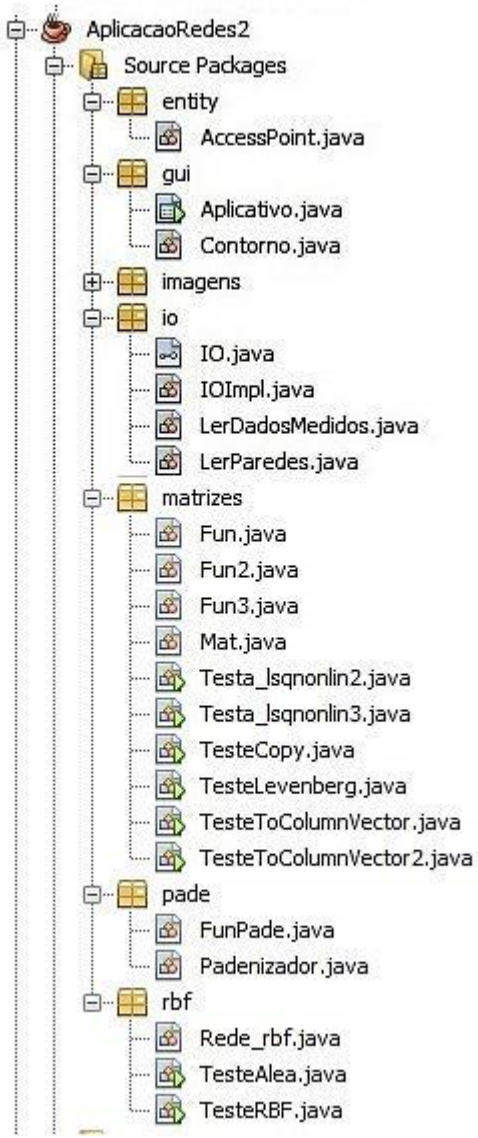

Fonte: Elaborado pela própria autora

# 5 APLICAÇÃO

O MMQNL pode ser aplicado para a determinação dos parâmetros de Padé em um *software* Java para posicionamento ótimo de WLANs. Pereira (2016) deu continuidade aos estágios desenvolvidos por Carneiro (2011) incluindo uma rotina de MQNL com restrições do tipo caixa.

O aplicativo foi desenvolvido na ferramenta NetBeans IDE 8.0.2 e na figura 25 as classes dele.

**Figura 25 – Classes do aplicativo Java**




Fonte: Elaborado pela própria autora

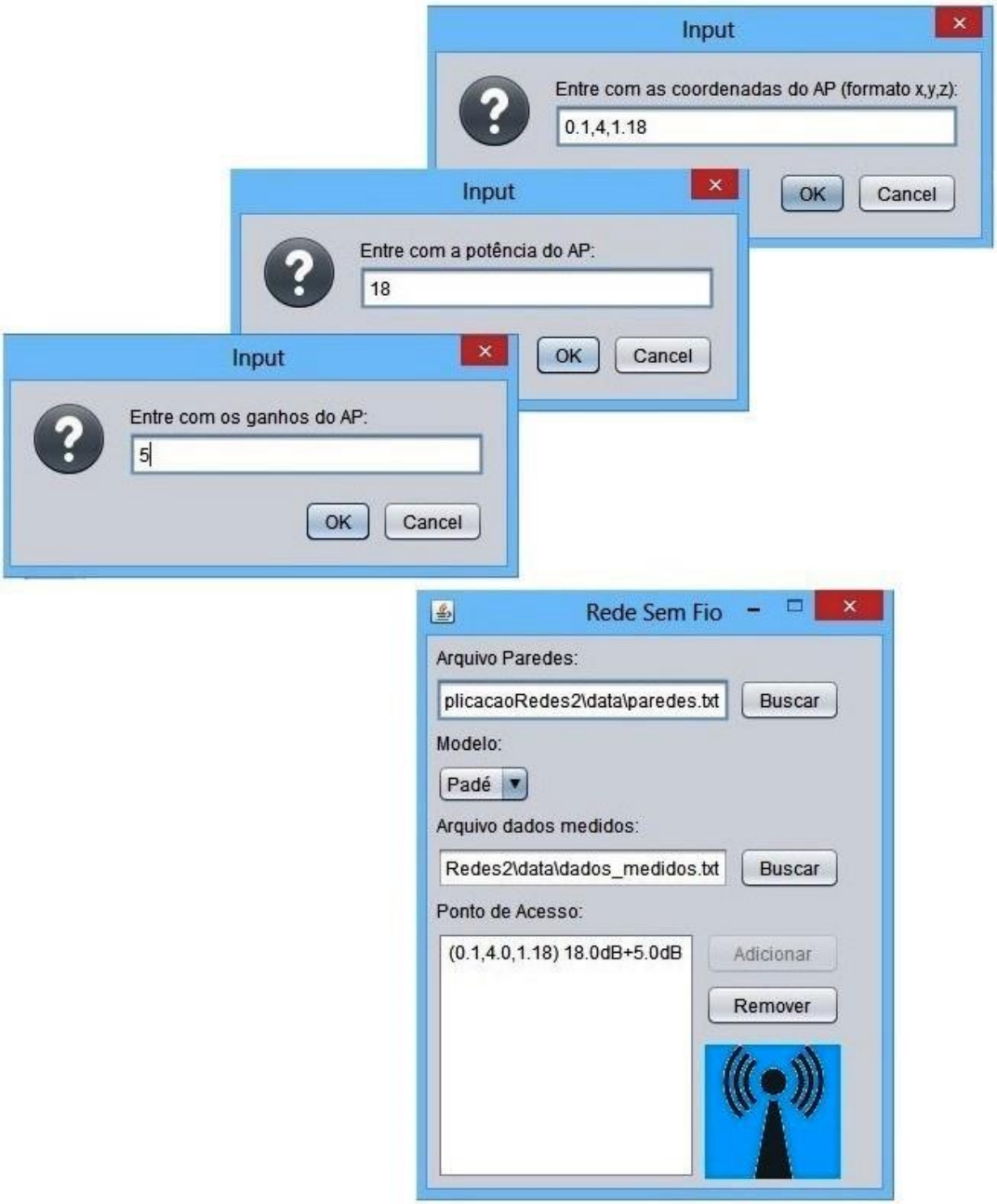
A figura 26 mostra a tela inicial do aplicativo.

**Figura 26 – Tela inicial do aplicativo**


Fonte: Elaborado pela própria autora

Na figura 27 um exemplo da entrada de dados para o programa já tendo sido fornecidos o arquivo .txt que contém as localizações das paredes e o arquivo .txt dos dados medidos que contém a potência recebida em cada local em dB.

**Figura 27 – Entrada de dados**




Fonte: Elaborado pela própria autora

## 5.1 Aproximantes de Padé e Redes Neurais RBF

Os aproximantes de Padé são funções racionais, eles servem para aproximar funções transcendentes como a função exponencial. Às vezes se está interessado no comportamento de uma função transcendente dentro de um intervalo de seu domínio, para isso se usam os aproximantes de Padé. As referências Fraiha (2009) e Fraiha et al. (2007) fazem uso de um aproximante de Padé como segue:

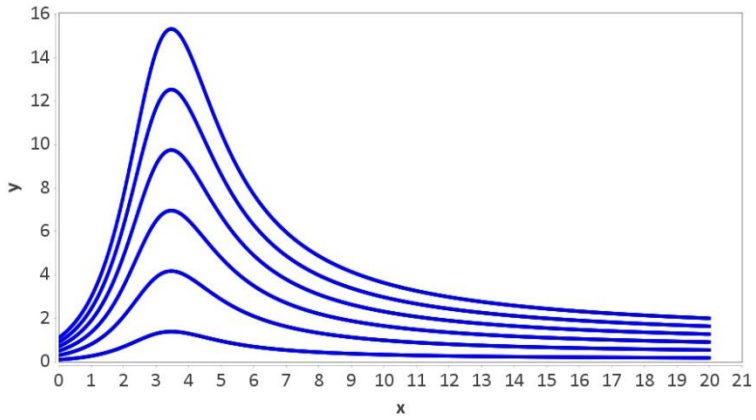
$$f(x;a,b) = \frac{a\left(1+\frac{bx}{2}+\frac{b^2x^2}{12}\right)}{1-\frac{bx}{2}+\frac{b^2x^2}{12}} \quad (5.1)$$

Fonte: (Pereira, Fraiha, Gomes, 2016, p. 01)

O comportamento da não linearidade de $f(x;a,b)$ pode ser visto como uma substituição local da função exponencial, onde $a$ promove a altura de pico até onde se deseja o comportamento da exponencial, enquanto que $b$ promove o deslocamento horizontal da curva. Assim é possível escolher $a$ e $b$ de modo a substituir uma parte de uma função exponencial.

A figura 28 mostra o comportamento do parâmetro $a$ para 6 valores de $a$ no intervalo $[0.1, 1.1]$ e $b$ fixo com valor igual a unidade, no eixo $\boldsymbol{x}$ os valores de $x$ e a função de Padé no eixo $\boldsymbol{y}$.

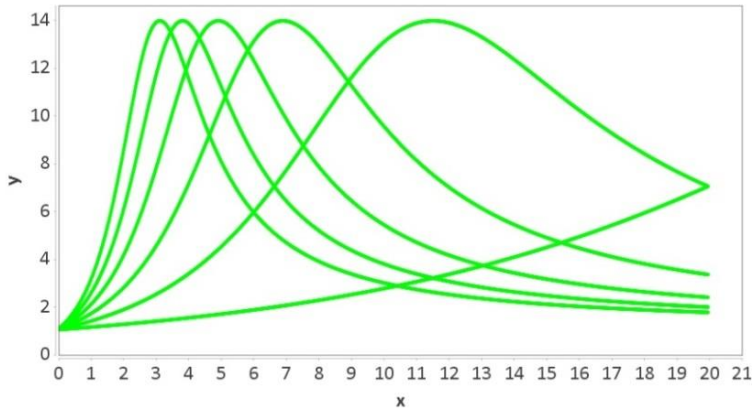
**Figura 28 – Gráfico da função de Padé com $a$ variando e $b$ constante**




Fonte: Elaborado pela própria autora

A figura 29 mostra o comportamento do parâmetro $b$ para 6 valores de $b$ no intervalo $[0.1, 1.1]$ e $a$ fixo com valor igual a unidade, no eixo $\boldsymbol{x}$ os valores de $x$ e a função de Padé no eixo $\boldsymbol{y}$.

**Figura 29 – Gráfico da função de Padé com $b$ variando e $a$ constante**




Fonte: Elaborado pela própria autora

Uma rede neural RBF (*Radial Basis Function*), que é um regressor generalizado (HAYKIN, 2001) é responsável pela expansão dos dados dos dados medidos, necessária para o cálculo das perdas em cada ponto do ambiente.

Depois dos dados terem sido expandidos os parâmetros de Padé (valores de $a$ e $b$ em (5.1)) são calculados por meio de uma rotina de MQNL, os parâmetros não podem assumir qualquer valor, eles não podem ser negativos devido a isto levar a uma pobre visualização dos obstáculos, o ideal é que eles estejam entre um limite inferior e superior, logo a rotina de MQNL deve ser implementada com restrições do tipo caixa.

Com os dados obtidos a probabilidade de recepção é calculada usando-se a distribuição exponencial, de acordo com Fraiha (2009):

$$P(x) = 1 - e^{-\alpha(x-limiar)} \tag{5.2}$$

Fonte: (Pereira, Fraiha, Gomes, 2016, p. 01)

Para conseguir $P$ em percentuais é só multiplicar por 100%, como foi feito em (Pereira, Fraiha, Gomes, 2016) "Onde $limiar$ é o limite mínimo para recepção e $\alpha$ é uma constante que depende da potência recebida na distância de referência." (Pereira, Fraiha, Gomes, 2016, p. 01).

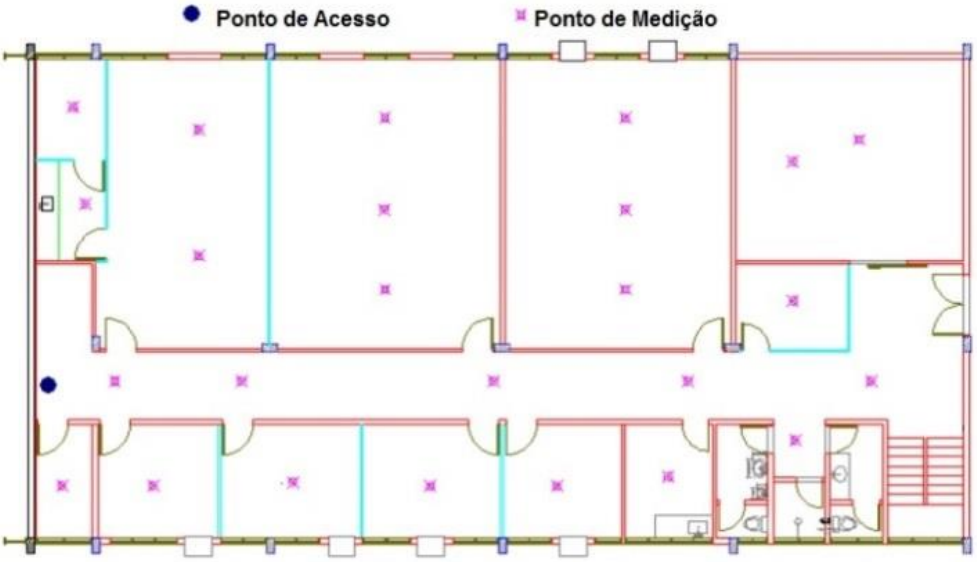
## 5.2 Medições

As medições foram realizadas no térreo de um prédio anexo ao Laboratório de Engenharia Elétrica e de Computação (LEEC) da Universidade Federal do Pará (UFPA). Em 2008, a época das medições, ele estava em fase de acabamento e, portanto estava sem mobílias e pessoas (Fraiha, 2009). "Suas paredes são de tijolo com janelas de vidro e esquadrias de alumínio em toda a sua extensão, além disso, são usadas divisórias para separar algumas salas." (Pereira, Fraiha, Gomes, 2016, p. 01).

A campanha de medição seguiu a metodologia descrita a seguir:

- **Determinação dos pontos de medição e do ponto de acesso -** A figura 30 mostra a planta baixa do andar térreo do prédio, onde estão localizados 25 pontos de medição e o ponto de acesso (PA) em azul, a rede gerada desse PA foi denominada de rede em estudo (Fraiha, 2009);
- **Conexão da Rede em Estudo** - A arquitetura da rede em estudo utiliza o canal 7 e frequência central de 2,442 GHz (Fraiha, 2009);
- **Medida da Potência –** A medição da potência em cada ponto foi feita usando um *notebook* com o *software Network Stumbler*© (Fraiha, 2009).

**Figura 30 – Planta baixa do prédio mostrando os pontos de medição com asteriscos lilás e o ponto de acesso em círculo azul**



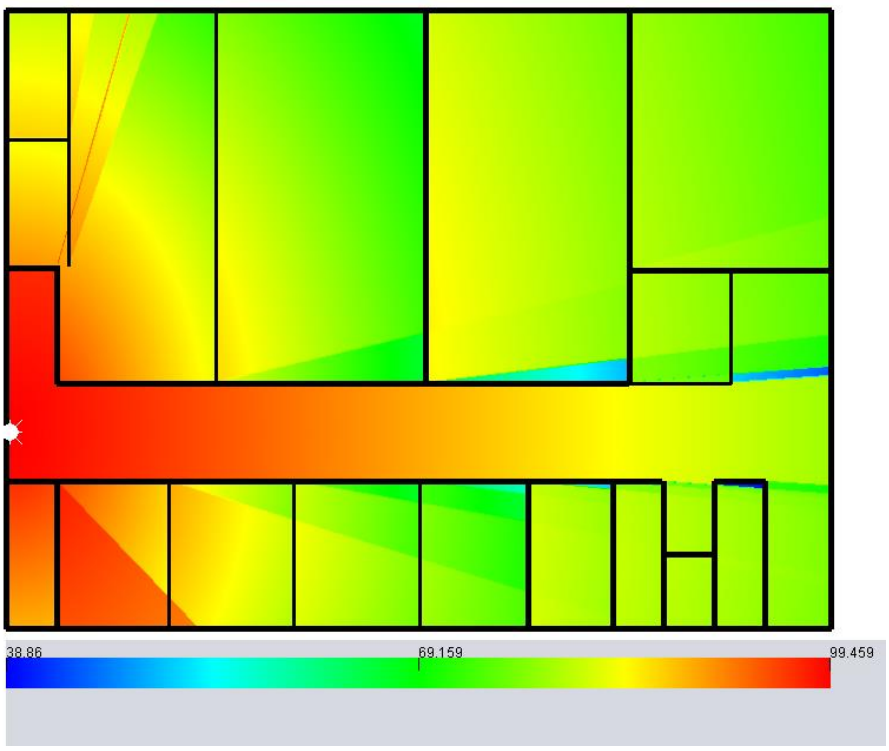

Fonte: (Pereira, Fraiha, Gomes, 2016, p. 01)

# 6 RESULTADOS

Será exposto o uso do MMQNL sem restrições e com restrições do tipo caixa para a determinação dos parâmetros de Padé no *software* Java para o posicionamento ótimo dos pontos de acesso de uma WLAN em um ambiente *indoor*.

As figuras 31, 32 e 33 mostram a probabilidade de recepção do sinal para diferentes localizações dos pontos de acesso, os parâmetros de Padé foram determinados pelo método de Levenberg-Marquardt. O modelo apresenta sensibilidade à presença de paredes e divisórias do prédio, além de ser sensível à distância. Próximo do ponto de acesso o sinal é forte, tendo uma alta probabilidade de recepção, por outro lado longe do PA o sinal mostra-se fraco, cor azul, devido à distância e a presença de paredes. "Em alguns locais do mapa de cores a rede RBF mostrou valores subestimados. Isto ocorre devido à falta de dados nestes locais. As redes RBF aglutinam bons valores próximos aos pontos medidos e subestimam valores distantes." (Pereira, Fraiha, Gomes, 2016, p. 01). Na figura 31 o método de Levenberg-Marquardt com restrições do tipo caixa no intervalo $[0, 4]$ forneceu como resultado os parâmetros $(2.8145, 1.8684)$, o ponto de acesso está localizado em $(0.1, 4, 1.18)$ tendo potência igual a 18 e os ganhos iguais a 5.

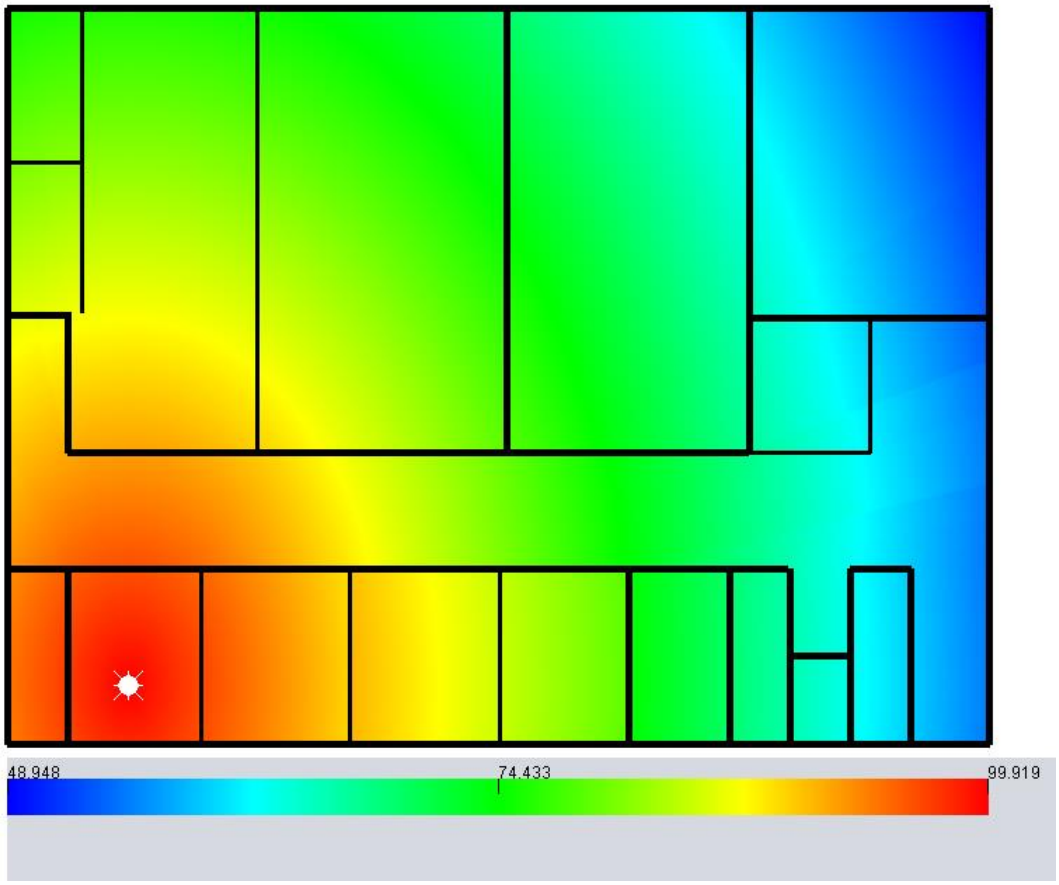
**Figura 31 – Mapa de calor mostrando as diferentes probabilidades de recepção a partir do ponto de acesso juntamente com paredes e divisórias**




Fonte: Elaborado pela própria autora

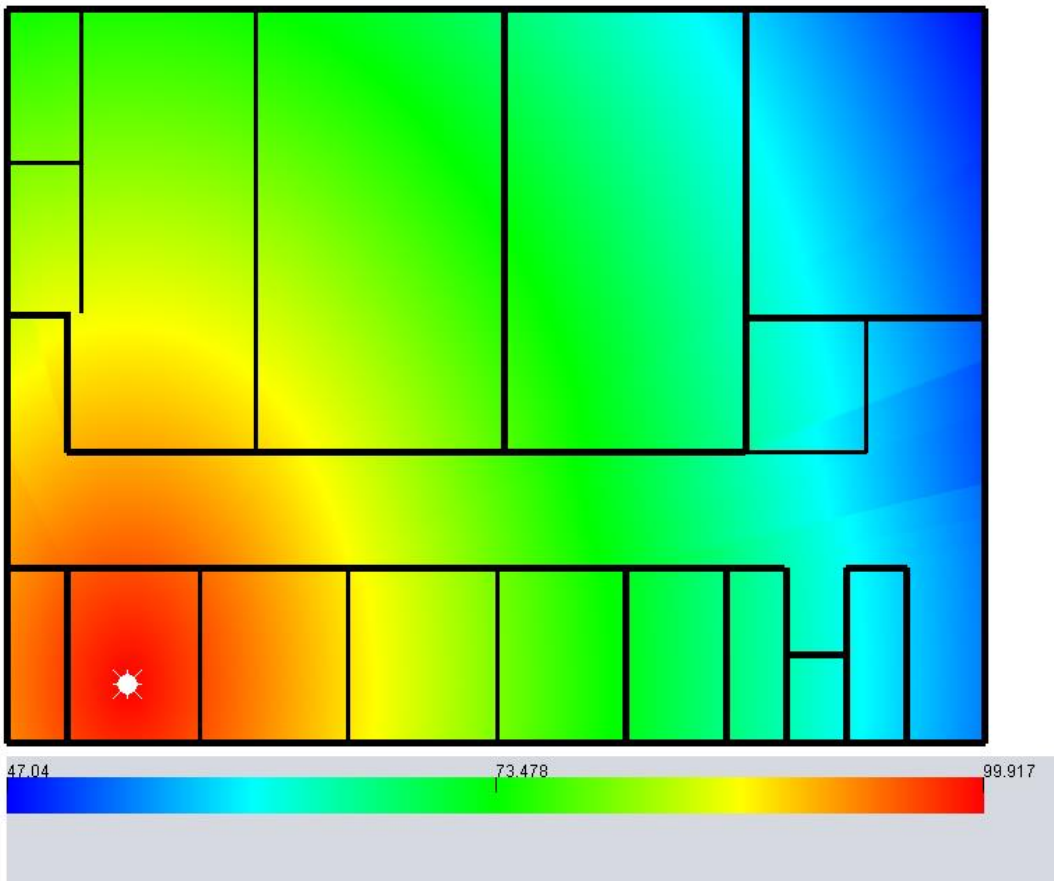
Nas figuras 32 e 33 com o ponto de acesso na localização $(3, 1, 0.5)$ tendo potência igual a 18 e os ganhos iguais a 0, o Levenberg-Marquardt foi utilizado primeiro sem restrições e depois com restrições do tipo caixa no intervalo $[0, 4]$. Sem restrições um dos parâmetros deu valor negativo enquanto que com restrições do tipo caixa os dois deram positivos permitindo uma melhor visualização das paredes e divisórias como obstáculos.

**Figura 32 – Mapa de calor mostrando as diferentes probabilidades de recepção com os parâmetros de Padé determinados por Levenberg-Marquardt sem restrições**




Fonte: Elaborado pela própria autora

**Figura 33 – Mapa de calor mostrando as diferentes probabilidades de recepção com os parâmetros de Padé determinados por Levenberg-Marquardt com restrições do tipo caixa**




Fonte: Elaborado pela própria autora

O trabalho foi bem sucedido com a rotina de MQNL determinando valores satisfatórios para os parâmetros.

# 7 CONSIDERAÇÕES FINAIS

## 7.1 Conclusões

Neste trabalho realizou-se uma implementação em Java do MMQNL para dar suporte ao problema da localização ótima de WLAN. A implementação das restrições do tipo caixa com a tangente hiperbólica é uma forma de forçar as variáveis a se situarem dentro do intervalo pretendido, uma vantagem de se utilizá-la é que pode ser aplicadas a diversas variantes do MMQNL e não está condicionada a uma implementação em particular.

Durante a execução deste trabalho foram encontradas dificuldades relacionadas à implementação das rotinas no Java, porque as rotinas dos capítulos 5 e 6 referentes ao *software* Java estavam implementadas no Matlab então foi necessário transcrevê-las para o Java e para a implementação do MMQNL tomou-se como base o que foi implementado e testado no capítulo 4.

## 7.2 Trabalhos Futuros

Para trabalhos futuros pode-se sugerir o cálculo dos parâmetros de Padé por uma rotina de otimização mais moderna do tipo Algoritmo Genético, *Simulated Annealing* ou Busca Tabu.

## 7.3 Informações Adicionais

Os capítulos 5 e 6 deste trabalho tiveram contribuição do que foi desenvolvido sob uma Bolsa de Iniciação Científica dentro do PIBIC (Programa Institucional de Bolsas de Iniciação Científica) financiado pela FAPESPA (Fundação Amazônia de Amparo a Estudos e Pesquisas) na Universidade Federal do Pará sob a orientação da professora Dra. Simone Fraiha. Foi citado nos capítulos o trabalho resultante da Bolsa apresentado no XXXIV SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES – SBrT2016 realizado em Santarém, Pará no período de 30 de agosto a 02 de setembro com o título “Predição de perda de propagação em WLANs usando aproximação de Padé com redes neurais RBF”.

**REFERÊNCIAS**


BABU, G. J.; FEIGELSON, E. D. **Analytical and Monte Carlo Comparisons of six different Linear Least Squares Fits**, 1992.

CARNEIRO, O. O.; FRAIHA, S. G. C.; GOMES, H. S. **Uso de Rede de Função de Base Radial para Expansão de Dados de um Sinal de Rede sem Fio**. In: Computer on the Beach, Univali, Florianópolis, 2011.

CHAPRA, S. C.; CANALE, R. P. **Numerical Methods for Engineers**. 6 ed. Boston: McGraw-Hill, 2010.

CHONG, E. K. P.; ŻAK, S. H. **An Introduction to Optimization**. 2 ed. Nova York: John Wiley & Sons, 2001.

CUNHA, M. C. C. **Métodos Numéricos.** 2 ed. rev. e aum. 2ª reimpressão. Campinas: Editora Unicamp, 2000.

DEVERNAY, F. **C/C++ Minpack**, 2007. Disponível em: <http://devernay.free.fr/hacks/cminpack/>. Acesso em: 31 de mar. 2017.

DUKKIPATI, R. V. **Numerical Methods**. Nova Delhi: New Age International, 2010.

FRAIHA, S. G. C. et al. **Methodology for Analysis of the Coverage Probability of WLAN using the Padé Approximant**. In: IMOC 2007, Salvador, 2007.

FRAIHA, S. G. C. **Localização Ótima de Pontos de Acesso em Ambientes Indoor em Projetos de Sistemas Wireless.** Tese (Doutorado) – Universidade Federal do Pará, Belém. 2009.

GOROSTIZAGA, J. C. **Funciones Hiperbólicas**, 2007. Disponível em: <http://www.ehu.eus/juancarlos.gorostizaga/apoyo/func_hiperbolic.htm>. Acesso em: 14 de abr. 2017.

HAYKIN, S. **Redes Neurais: Princípios e Prática**. 2 ed. Porto Alegre: Bookman, 2001.

KILHIAN, K. **Como encontrar o foco e a diretriz de uma parábola com régua e compasso**, 2015. Disponível em: <http://obaricentrodamente.blogspot.com.br/2015/11/como-encontrar-o-foco-e-diretriz-de-uma.html>. Acesso em: 24 de abr. 2017.

LIMA, E. L. **Elementos de Topologia Geral.** 3 ed. Rio de Janeiro: LTC, 1970.

MADSEN, K.; NIELSEN, H. B.; TINGLEFF, O. **Methods for Nonlinear Least Squares Problems**. 2 ed. Copenhagen: Informatics and Mathematical Modelling - Technical University of Denmark, 2004.

NASH, J.C. **Nonlinear Parameter Optimization using R tools**. 1 ed. Ottawa: John Wiley & Sons, 2014.

NOCEDAL, J.; WRIGHT, S. J. **Numerical Optimization**. 2 ed. Nova York: Springer, 2006.

PEREIRA, J. O.; FRAIHA, S. G. C.; GOMES, H. S. **Predição de Perda de Propagação em WLANs usando Aproximação de Padé com Redes Neurais RBF**. In: XXXIV SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES – SBrT2016, Santarém, 2016.

RICH, B.; THOMAS, C. **Geometry**. 4 ed. Nova York: McGraw-Hill, 2009. Schaum's Outlines.

RODRIGUES, S. C. A. **Modelo de Regressão Linear e suas Aplicações.** Dissertação (Mestrado) – Universidade da Beira Interior, Covilhã. 2012.

TEAGUE, D. J. **Re-expression and Linear Regression**, 2005.

THOMAS, G. B. **Cálculo**. 11 ed. São Paulo: Addison Wesley, 2009. v. 1.

WEISS, J. **A Catalog of Type II Regression Methods**, 2003. Disponível em: <https://www.unc.edu/courses/2007spring/biol/145/001/docs/lectures/Nov5.html>. Acesso em: 14 de abr. 2017.

## APÊNDICE – Principais partes do código fonte do programa

O apêndice contém alguns trechos do código fonte desenvolvido nos capítulos 3 e 4 do trabalho sem incluir partes do código fonte do aplicativo Java citado nos capítulos 5 e 6, que possui muito mais linhas de código.

```
package matrizes;

import graficos.GraficoFun;
import java.io.FileWriter;

/**
 *
 * @author Jeanne
 */
public class TestaMinqua1 {
   /* Mínimos Quadrados Lineares
    * Y como função de x   Y=a0+a1*X
    * X como função de y   X=b0+b1*Y
    * Reta Bissetriz   Y=c0+c1*X das anteriores
    * v=[ a0 a1 b0 b1 c0 c1 ]
    */

   public static void main(String[] args) {
      double xx[]= { 1, 2, 3, 4 };
      double yy[]= { 10.1, 12.3, 15.2, 16.4};
      int m=xx.length;
      double[] v= Mat.minqua(xx, yy);
      String novaLinha = System.getProperty("line.separator");
      try{
         FileWriter resultados = new FileWriter("arqs_resultados/"
               + "resultados_minqua1.txt");
         Mat.print(v, "Vetor de coeficientes ");


         resultados.write(Mat.toStr(v, "Vetor de coeficientes "));

         double[] f = new double[m];
         for (int i = 0; i < m; i++) {
            f[i] = v[0] + v[1] * xx[i];
         }
         Mat.print(f, "Valores da reta da regressao "
               + "linear de y em funcao de x");
         resultados.write(Mat.toStrLn(f, "Valores da reta da regressao "
               + "linear de y em funcao de x"));

         double resq = 0;
         double[] res = new double[m];
         for (int i = 0; i < m; i++) {
            res[i] = f[i] - yy[i];
            resq += Math.pow(res[i], 2);
         }
         Mat.print(res, "Residuos da regressao linear de y "
               + "em funcao de x");
```

```
        System.out.println("Resq da regressao linear de y "
                + "em funcao de x = " + resq + "\n");

        resultados.write(Mat.toStrLn(res, "Residuos da "
                + "regressao linear de y em funcao de x"));
        resultados.write("Resq da regressao linear de y "
                + "em funcao de x = " + resq + novaLinha + novaLinha);

        resq = 0;
        for (int i = 0; i < m; i++) {
           res[i] = v[2] + v[3] * xx[i] - yy[i];
           resq += Math.pow(res[i], 2);
        }
        System.out.println("Resq da regressao linear de x "
                + "em funcao de y = " + resq + "\n");
        resultados.write("Resq da regressao linear de x "
                + "em funcao de y = " + resq + novaLinha + novaLinha);

        for (int i = 0; i < m; i++) {
           f[i] = v[4] + v[5] * xx[i];
        }
        Mat.print(f, "Valores da reta bissetriz");
        resultados.write(Mat.toStrLn(f, "Valores da reta bissetriz"));

        resq = 0;
        for (int i = 0; i < m; i++) {
           res[i] = f[i] - yy[i];
           resq += Math.pow(res[i], 2);
        }
        System.out.println("Resq da bissetriz = " + resq + "\n");
        resultados.write("Resq da bissetriz = " + resq
                + novaLinha + novaLinha);

        resultados.write("^y(6) = "+(v[0] + v[1]*6)
                +novaLinha+novaLinha);
        System.out.println("^y(6) = "+(v[0] + v[1]*6)+"\n");

        resultados.close();

        double[] t = Mat.linspace(0, 6, 300);
        double[] y = new double[t.length];
        for (int i = 0; i < t.length; i++) {
           y[i] = v[0] + v[1] * t[i];
        }
        GraficoFun graf = new GraficoFun(
                "MQL01",
                xx, yy,
                t, y
        );

        graf.criarGrafico();
        graf.salvarGrafico("MQL01");
    }
    catch(Exception exc) {
        System.out.println(exc);
    }
```

```
   }

}

package matrizes;

import graficos.GraficoFun;
import java.io.FileWriter;

/**
 *
 * @author Jeanne
 */
public class TestaMinqua2 {
   /* Mínimos Quadrados Lineares
    * Y como função de x   Y=a0+a1*X
    * X como função de y   X=b0+b1*Y
    * Reta Bissetriz   Y=c0+c1*X das anteriores
    * v=[ a0 a1 b0 b1 c0 c1 ]
    */

   public static void main(String[] args) {

      try{
         double[] xx = {0.1, 0.2, 0.4, 0.6, 0.9, 1.3, 1.5, 1.7, 1.8};
         double[] yy = {0.75, 1.25, 1.45, 1.25, 0.85, 0.55, 0.35, 0.28,
            0.18};
         int m=xx.length;
         double[] zz = new double[m];
         for(int i=0; i < m; i++) {
            zz[i]=Math.log(yy[i]/xx[i]);
         }
         double[] v = Mat.minqua(xx, zz);

         String novaLinha = System.getProperty("line.separator");

         FileWriter resultados = new FileWriter("arqs_resultados/"
               + "resultados_minqua2.txt");
         Mat.print(v, "Vetor de coeficientes ");
         resultados.write(Mat.toStr(v, "Vetor de coeficientes "));

         double[] f = new double[m];
         for(int i=0; i<m; i++) {
            f[i]=v[0]+v[1]*xx[i];
         }
         Mat.print(f, "z=ln(y/x)");
         resultados.write(Mat.toStrLn(f, "z=ln(y/x)"));

         double resq=0;
         double[] res = new double[m];
         for(int i=0; i<m; i++) {
            res[i] = f[i]-zz[i];
            resq += Math.pow(res[i],2);
         }
         Mat.print(res, "Residuos dos dados transformados");
         System.out.println("Resq do minqua = "+resq+"\n");
```

```
resultados.write(Mat.toStrLn(res,
      "Residuos dos dados transformados"));
resultados.write("Resq do minqua = "+resq+novaLinha+novaLinha);

System.out.println("Coeficientes do modelo nao linear"+"\n\n");
System.out.println(Math.exp(v[0]) + " " + v[1] + "\n");
resultados.write("Coeficientes do modelo nao linear"
      + novaLinha + novaLinha + novaLinha);
resultados.write(Math.exp(v[0]) + " " + v[1]
      + novaLinha + novaLinha);

for(int i=0; i<m; i++) {
   f[i]=Math.exp(v[0])*xx[i]*Math.exp(v[1]*xx[i]);
}
Mat.print(f, "y");
resultados.write(Mat.toStrLn(f, "y"));

resq=0;
for(int i=0; i < m; i++) {
   res[i] = f[i]-yy[i];
   resq += Math.pow(res[i],2);
}
Mat.print(res, "Residuos dos dados sem transformacao");
System.out.println("Resq do minqua sem transformacao = "
      +resq+"\n");
resultados.write(Mat.toStrLn(res, "Residuos dos dados "
      + "sem transformacao"));
resultados.write("Resq do minqua sem transformacao = "+
      resq+novaLinha+novaLinha);

resultados.close();

// Dados transformados
double[] t = Mat.linspace(0, 2, 300);
double[] y = new double[t.length];
double[] z = new double[t.length];
for(int i=0; i < t.length; i++) {
   z[i] = v[0]+v[1]*t[i];
}
GraficoFun graf = new GraficoFun(
   "MQL02_01",
   xx,zz,
   t,z
);
graf.setYLabel("ln(y/x)");

graf.criarGrafico();
graf.salvarGrafico("MQL02_01");

//Dados originais
for(int i=0; i < t.length; i++) {
   y[i]=Math.exp(v[0])*t[i]*Math.exp(v[1]*t[i]);
}
graf = new GraficoFun(
   "MQL02_02",
   xx,yy,
   t,y
```

```
            );

            graf.criarGrafico();
            graf.salvarGrafico("MQL02_02");
        }
        catch(Exception exc) {
            System.out.println(exc);
        }

    }

}


package matrizes;

import funcoes.Fun2;
import graficos.GraficoFun;
import java.io.FileWriter;

/**
 *
 * @author Jeanne
 */
public class Testa_curvefit1 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        try{
            double[] xx = new double[11];
            for(int i=0; i < xx.length; i++)
                xx[i] = 2*i/10.0;
            double[] yy = {0.45, 0.52, 0.46, 0.40, 0.29, 0.21, 0.15, 0.08, 0.04, 0.02,
0.01};

            Fun2 f2 = new Fun2(xx,yy);
            double[] alfa0 = {1,0,1};

            String novaLinha = System.getProperty("line.separator");

            FileWriter resultados = new FileWriter("arqs_resultados/"
                    + "resultados_cuverfit1.txt");

            f2.sol = Mat.gauss_newton(f2, alfa0, resultados);

            Mat.print(f2.sol, "Solucao");
            resultados.write(Mat.toStr(f2.sol, "Solucao"));

            double[] fsol = f2.func(f2.sol);
            Mat.print(fsol, "Funcao na solucao");
            resultados.write(Mat.toStrLn(fsol, "Funcao na solucao"));

            double[] res = f2.res(f2.sol);
            Mat.print(res, "Residuos na solucao");
```

```
            resultados.write(Mat.toStrLn(res, "Residuos na solucao"));

            System.out.println("Residuo quadratico = "+f2.resq());
            resultados.write("Residuo quadratico = "+f2.resq()+novaLinha);
            resultados.close();

            double[] t = Mat.linspace(-0.2, 2, 300);
            GraficoFun graf = new GraficoFun(
               "MQNL01_01",
               xx,yy,
               t,f2.func(f2.sol, t)
            );

            graf.criarGrafico();
            graf.salvarGrafico("MQNL01_01");
        }
        catch(Exception exc) {
            System.out.println(exc);
        }




    }
}

package matrizes;

import funcoes.Fun3;
import graficos.GraficoFun;
import java.io.FileWriter;
import java.io.IOException;

/**
 *
 * @author Jeanne
 */
public class Testa_curvefit2 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        try{
            double[] xx = new double[10];
            for(int i=0; i < xx.length; i++)
                xx[i] = i+1;
            double[] yy = {2, 3.5, 4.5, 3.7, 2.7, 1.3, 1.4, 3, 4.5, 5.5};

            Fun3 f3 = new Fun3(xx,yy);
            double[] alfa0 = {2, 0.8, 1.5, 1};
            String novaLinha = System.getProperty("line.separator");

            FileWriter resultados = new FileWriter("arqs_resultados/"
                    + "resultados_cuverfit2.txt");
```

```
            f3.sol=Mat.levenberg_marquardt(f3, alfa0, resultados);

            double[] fsol = f3.func(f3.sol);
            double[] res = f3.res(f3.sol);
            Mat.print(f3.sol, "Solucao");
            Mat.print(fsol, "Funcao na solucao");
            Mat.print(res, "Residuos na solucao");
            System.out.println("Residuo quadratico = "+f3.resq());

            resultados.write(Mat.toStr(f3.sol, "Solucao"));
            resultados.write(Mat.toStrLn(fsol, "Funcao na solucao"));
            resultados.write(Mat.toStrLn(res, "Residuos na solucao"));
            resultados.write("Residuo quadratico = "+f3.resq()+novaLinha);

            double[] t = Mat.linspace(0, 11, 300);
            GraficoFun graf = new GraficoFun(
                  "MQNL01_01",
                  xx,yy,
                  t,f3.func(f3.sol, t)
            );

            graf.criarGrafico();
            graf.salvarGrafico("MQNL02_01");

            double[] l = {0,0,0,1};
            double[] u = {3,4,4,3};

            f3.sol=Mat.levenberg_marquardt(f3, alfa0, l, u, resultados);
            fsol = f3.func(f3.sol);
            res = f3.res(f3.sol);
            Mat.print(f3.sol, "Solucao");
            Mat.print(fsol, "Funcao na solucao");
            Mat.print(res, "Residuos na solucao");
            System.out.println("Residuo quadratico = "+f3.resq());

            resultados.write(Mat.toStr(f3.sol, "Solucao"));
            resultados.write(Mat.toStrLn(fsol, "Funcao na solucao"));
            resultados.write(Mat.toStrLn(res, "Residuos na solucao"));
            resultados.write("Residuo quadratico = "+f3.resq()+novaLinha);

            t = Mat.linspace(0, 11, 300);
            graf = new GraficoFun(
                  "MQNL02_02",
                  xx,yy,
                  t,f3.func(f3.sol, t)
            );

            graf.criarGrafico();
            graf.salvarGrafico("MQNL02_02");

            resultados.close();
         }catch(IOException exc) {
            System.out.println(exc);
         }

      }
}
```

```
package matrizes;

import funcoes.Fun4;
import java.io.FileWriter;

/**
 *
 * @author Jeanne
 */
public class Testa_lsqnonlin1 {

   /**
    * @param args the command line arguments
    */
   public static void main(String[] args) {

      try {
         double[] x0 = {5,4};
         double[] l = {0,0};
         double[] u = {10,10};

         String novaLinha = System.getProperty("line.separator");

         FileWriter resultados = new FileWriter("arqs_resultados/"
               + "resultados_lsqnonlin1.txt");

         Fun4 f4 = new Fun4();
         f4.sol = Mat.levenberg_marquardt(f4, x0, resultados);
         Mat.print(f4.sol, "Solucao");
         resultados.write(Mat.toStr(f4.sol, "Solucao"));

         double[] res = f4.res(f4.sol);
         Mat.print(res, "Residuos");
         System.out.println("Residuo quadratico = " + f4.resq());
         resultados.write(Mat.toStrLn(res, "Residuos"));
         resultados.write("Residuo quadratico = " + f4.resq()+novaLinha);

         f4.sol = Mat.levenberg_marquardt(f4, x0, l, u, resultados);
         Mat.print(f4.sol, "Solucao");
         System.out.println("Residuo quadratico = " + f4.resq());
         resultados.write(Mat.toStr(f4.sol, "Solucao"));
         resultados.write("Residuo quadratico = " + f4.resq()+novaLinha);

         l[0]=4;
         l[1]=5;
         u[0]=10;
         u[1]=13;
         f4.sol = Mat.levenberg_marquardt(f4, x0, l, u, resultados);
         Mat.print(f4.sol, "Solucao");
         resultados.write(Mat.toStr(f4.sol, "Solucao"));


         System.out.println("Residuo quadratico = " + f4.resq());
         resultados.write("Residuo quadratico = " + f4.resq()+novaLinha);
```

```
            f4.sol = new double[]{4, 5.6988};
            Mat.print(f4.sol, "Solucao que piora o residuo");
            System.out.println("Residuo quadratico = " + f4.resq());
            resultados.write(Mat.toStr(f4.sol, "Solucao que "
                  + "piora o residuo"));
            resultados.write("Residuo quadratico = " + f4.resq()+novaLinha);

            l[0]=5;
            l[1]=8;
            f4.sol = Mat.levenberg_marquardt(f4, x0, l, u, resultados);
            Mat.print(f4.sol, "Solucao");
            System.out.println("Residuo quadratico = " + f4.resq());
            resultados.write(Mat.toStr(f4.sol, "Solucao"));
            resultados.write("Residuo quadratico = " + f4.resq() + novaLinha);

            resultados.close();
         }catch(Exception exc) {
            System.out.println(exc);
         }


      }

}


package matrizes;

import funcoes.Fun5;
import java.io.FileWriter;

/**
 *
 * @author Jeanne
 */
public class Testa_lsqnonlin2 {

   /**
    * @param args the command line arguments
    */
   public static void main(String[] args) {

      try {
         double[] x0 = {1,2};
         double[] l = {0,0};
         double[] u = {4,5};
         String novaLinha = System.getProperty("line.separator");

         FileWriter resultados = new FileWriter("arqs_resultados/"
               + "resultados_lsqnonlin2.txt");

         Fun5 f5 = new Fun5();
         f5.sol = Mat.levenberg_marquardt(f5, x0, resultados);
         Mat.print(f5.sol, "Solucao");
         resultados.write(Mat.toStr(f5.sol, "Solucao"));

         double[] res = f5.res(f5.sol);
```

```
            Mat.print(res, "Residuos");
            System.out.println("Residuo quadratico = " + f5.resq());
            resultados.write(Mat.toStrLn(res, "Residuos"));
            resultados.write("Residuo quadratico = " + f5.resq() + novaLinha);

            f5.sol = Mat.levenberg_marquardt(f5, x0, l, u, resultados);
            Mat.print(f5.sol, "Solucao");
            System.out.println("Residuo quadratico = "+f5.resq());
            resultados.write(Mat.toStr(f5.sol, "Solucao"));
            resultados.write("Residuo quadratico = " + f5.resq() + novaLinha);

            resultados.close();
        }catch(Exception exc) {
            System.out.println(exc);
        }
    }

}


package matrizes;

import funcoes.Fun6;
import java.io.File;
import java.io.FileWriter;

/**
 *
 * @author Jeanne
 */
public class Testa_lsqnonlin3 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        try {
            double[] x0 = {1,2,3}; // Alterei de x0 = {1,2,3}
            double[] l = {0,0,0};
            double[] u = {4,5,2};
            String novaLinha = System.getProperty("line.separator");

            FileWriter resultados = new FileWriter("arqs_resultados"
                    +"/"+"resultados_lsqnonlin3.txt");

            Fun6 f6 = new Fun6();
            f6.sol = Mat.levenberg_marquardt(f6, x0, resultados);
            Mat.print(f6.sol, "Solucao");
            resultados.write(Mat.toStr(f6.sol, "Solucao"));

            double[] res = f6.res(f6.sol);
            Mat.print(res, "Residuos");
            System.out.println("Residuo quadratico = " + f6.resq());
            resultados.write(Mat.toStrLn(res, "Residuos"));
            resultados.write("Residuo quadratico = " + f6.resq() + novaLinha);
```

```
            f6.sol = Mat.levenberg_marquardt(f6, x0, l, u, resultados);
            Mat.print(f6.sol, "Solucao");
            System.out.println("Residuo quadratico = " + f6.resq());
            resultados.write(Mat.toStr(f6.sol, "Solucao"));
            resultados.write("Residuo quadratico = " + f6.resq() + novaLinha);

            resultados.close();
        }catch(Exception exc) {
            System.out.println(exc);
        }

    }

}

package matrizes;

import funcoes.*;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;


/**
 * Classe contendo rotinas e operações referentes ao uso de Matrizes.
 *
 * @author Herminio
 *
 * Última Modificação Jeanne de Oliveira Pereira 19/04/2017 13:36
 */
public class Mat {
/**
    * Método que resolve um problema de mínimos quadrados não
    * lineares utilizando Gauss-Newton.
    * @param f Função contendo os resíduos cuja soma
    * quadrática será minimizada.
    * @param alfa0 Chute inicial.
    * @return Vetor contendo a solução do problema.
    */
   public static double[] gauss_newton(FunMQNL f, double[]alfa0) {
       int n=alfa0.length;
       System.out.println("Número de parametros  dados = "+n);
       int KMAX=  500 ;
       double TOL=1e-5;
       double erro=1e10;
       Mat.print(alfa0, "\n Vetor inicial dado");
       double[] alfa=new double[n];
       Mat.copy(alfa0, n, alfa);

       boolean found=false;
       double [][] J  = f.jacdisc(alfa);
       double []   fa = f.res(alfa);
       double [][] A =  Mat.prod(Mat.transp(J), J);
       double []   g= Mat.prod(Mat.transp(J), fa);
       double []   gm = new double [n] ;

       int k = 0;
```

```
      while(k < KMAX && !found) {
         k++;
         for(int j=0; j<n; j++) gm[j]=-g[j];
         double[] h = Mat.solve_LU(A, gm);
         alfa = Mat.sum(alfa, h);
         J = f.jacdisc(alfa);
         fa = f.res(alfa);
         A =  Mat.prod(Mat.transp(J), J);
         g= Mat.prod(Mat.transp(J), fa);

         int[] ind = new int[1];
         erro = Mat.maxabs(h, ind);
         if(erro < TOL) {
            found = true;
         }
      }

      System.out.println("Iteracoes da lsqnonlin = "+k);
      System.out.println("Erro final = "+erro+"\n");
      return alfa;
   } // gauss_newton

   /**
    * Método que resolve um problema de mínimos quadrados não
    * lineares utilizando Gauss-Newton.
    * @param f Função contendo os resíduos cuja soma
    * quadrática será minimizada.
    * @param alfa0 Chute inicial.
    * @param arqres Arquivo .txt contendo os resultados
    * fornecidos pelo método.
    * @return Vetor contendo a solução do problema.
    * @throws java.io.IOException
    */
   public static double[] gauss_newton(FunMQNL f, double[]alfa0,
         FileWriter arqres) throws IOException {
      int n=alfa0.length;
      String novaLinha = System.getProperty("line.separator");
      System.out.println("Numero de parametros  dados = "+n);
      arqres.write("Numero de parametros  dados = "+n+novaLinha);
      int KMAX=  500 ;
      double TOL=1e-5;
      double erro=1e10;
      Mat.print(alfa0, "\n Vetor inicial dado");
      arqres.write(Mat.toStr(alfa0, novaLinha+" Vetor inicial dado"));
      double[] alfa=new double[n];
      Mat.copy(alfa0, n, alfa);

      boolean found=false;
      double [][] J  = f.jacdisc(alfa);
      double []   fa = f.res(alfa);
      double [][] A =  Mat.prod(Mat.transp(J), J);
      double []   g= Mat.prod(Mat.transp(J), fa);
      double []   gm = new double [n] ;

      int k = 0;
      while(k < KMAX && !found) {
         k++;
```

```
        for(int j=0; j<n; j++) gm[j]=-g[j];
        double[] h = Mat.solve_LU(A, gm);
        alfa = Mat.sum(alfa, h);
        J = f.jacdisc(alfa);
        fa = f.res(alfa);
        A =  Mat.prod(Mat.transp(J), J);
        g= Mat.prod(Mat.transp(J), fa);

        int[] ind = new int[1];
        erro = Mat.maxabs(h, ind);
        if(erro < TOL) {
           found = true;
        }
     }

     System.out.println("Iteracoes da lsqnonlin = "+k);
     System.out.println("Erro final = "+erro+"\n");
     arqres.write("Iteracoes da lsqnonlin = "+k+novaLinha);
     arqres.write("Erro final = "+erro+novaLinha+novaLinha);
     return alfa;
  } // gauss_newton

  /**
   * Método que resolve um problema de mínimos quadrados não
   * lineares utilizando Levenberg-Marquardt.
   * @param f Função contendo os resíduos cuja soma
   * quadrÃitica será minimizada.
   * @param alfa0 Chute inicial.
   * @return Vetor contendo a solução do problema.
   */
  public static double[] levenberg_marquardt(FunMQNL f, double[]alfa0) {
     int n=alfa0.length;
     System.out.println("Numero de parametros  dados = "+n);
     int KMAX=  500 ;
     int k=0;
     double erro=1e10, v=2, tau=1e-3;
     final double  EPS1=1e-5;
     final double  EPS2=1e-5;
     Mat.print(alfa0, "\n Vetor inicial dado");
     double[] alfa = new double[n];
     Mat.copy(alfa0, n, alfa);

     double [] alfanew;
     double [][] J  = f.jacdisc(alfa);
     double []   fa = f.res(alfa);
     double [][] A =  Mat.prod(Mat.transp(J), J);
     double []   fan ;
     double []   g= Mat.prod(Mat.transp(J), fa);
     double []   gm = new double [n] ;
     boolean found = Mat.norminf(g)<EPS1;
     double[] diag = new double[n];
     for(int i=0; i < n; i++) {
        diag[i]=A[i][i];
     }
     double mi=tau*Mat.max(diag),ganho;

     while (!found && k<KMAX){
```

```
            k++;
            for(int j=0; j<n; j++) A[j][j] = A[j][j] + mi;
            for(int j=0;j<n;j++)   gm[j]=-g[j];
            double[] h = Mat.solve_cholesky(A, gm);
            erro= Mat.normdois(h);
//          System.out.println("erro = "+erro);
            if( erro <= EPS2*(Mat.normdois(alfa)+EPS2) ){
               found=true;
            }
            else
            {
               alfanew=Mat.sum(alfa, h);
               fa=f.res(alfa);
               fan=f.res(alfanew);
               ganho=0;
               for(int j=0; j<n;j++) ganho+=h[j]*(mi*h[j]-g[j]);
               double ro=(Mat.prod(fa, fa)- Mat.prod(fan,fan))/ganho;
               if( ro > 0){
                  Mat.copy(alfanew, n, alfa);
                  J= f.jacdisc(alfa);
                  A= Mat.prod(Mat.transp(J), J);
                  g= Mat.prod(Mat.transp(J), fan);
                  erro=Mat.norminf(g);
                  found=erro<=EPS1;
                  mi=mi*Math.max(0.3333333333333,1-Math.pow(2*ro-1,3)); v=2;
               }
               else
               {
                  mi=mi*v; v=2*v;
               }
            }
         }

         System.out.println("Iteracoes da lsqnonlin = "+k);
         System.out.println("Erro final = "+erro+"\n");
         return alfa;
      }// levenberg_marquardt

      /**
       * Método que resolve um problema de mínimos quadrados não
       * lineares utilizando Levenberg-Marquardt com restrições
       * em caixa.
       * @param f Função contendo os resíduos cuja soma
       * quadrática será minimizada.
       * @param alfa0 Chute inicial.
       * @param l Vetor contendo o limite inferior.
       * @param u Vetor contendo o limite superior.
       * @return Vetor contendo a solução do problema.
       */
      public static double[] levenberg_marquardt(FunMQNL f, double[]alfa0,
            double[] l, double[] u) {
         int n=alfa0.length;
         System.out.println("Numero de parametros  dados = "+n);
         int KMAX=  500 ;
         int k=0;
         double erro=1e10, v=2, tau=1;
         final double  EPS1=1e-5;
```

```
      final double  EPS2=1e-5;
      Mat.print(alfa0, "\n Vetor inicial dado");
      System.out.println("l = " + Arrays.toString(l));
      System.out.println("u = " + Arrays.toString(u));
      double[] alfa = new double[n];
      Mat.copy(alfa0,n,alfa);

      double [] alfanew;
      double[] alfaconstrained = new double[alfa.length];
      double[] jacfac = new double[alfa.length];

      for(int i = 0; i < n; i++) {
         if(u[i] <= l[i])
            throw new RuntimeException("Os valores de u devem "
               + "ser maiores do que l");
      }

      for(int i = 0; i < n; i++) {
         if(alfa[i] < l[i])
            alfa[i] = l[i];
         else if(alfa[i] > u[i])
            alfa[i] = u[i];
      }

      for (int i = 0; i < jacfac.length; i++) {
         double alfamiddle = (l[i]+u[i])/2;
         double alfawidth = (u[i]-l[i])/2;
         double th =  Math.tanh((alfa[i]-alfamiddle)/alfawidth);
         alfaconstrained[i] = alfamiddle + th * alfawidth;
         jacfac[i] = 1 - th * th;
      }

      double [][] J  = f.jacdisc(alfaconstrained);
      for(int i = 0; i < J.length; i++)
         for(int j = 0; j < jacfac.length; j++)
            J[i][j] *= jacfac[j];
      double []   fa = f.res(alfaconstrained);
      double [][] A =  Mat.prod(Mat.transp(J), J);
      double []   fan ;
      double []   g= Mat.prod(Mat.transp(J), fa);
      double []   gm = new double [n] ;
      boolean found = Mat.norminf(g)<EPS1;
      double[] diag = new double[n];
      for(int i=0; i < n; i++) {
         diag[i]=A[i][i];
      }
      double mi=tau*Mat.max(diag),ganho;

      while (!found && k<KMAX){
         k++;
         for(int j=0; j<n; j++) A[j][j] = A[j][j] + mi;
         for(int j=0;j<n;j++)   gm[j]=-g[j];
         double[] h = Mat.solve_cholesky(A, gm);
         erro= Mat.normdois(h);
//       System.out.println("erro = "+erro);
         if( erro <= EPS2*(Mat.normdois(alfa)+EPS2) ){
            found=true;
```

```
        }
        else
        {
            alfanew=Mat.sum(alfa, h);

            for (int i = 0; i < jacfac.length; i++) {
                double alfamiddle = (l[i]+u[i])/2;
                double alfawidth = (u[i]-l[i])/2;
                double th =  Math.tanh((alfa[i]-alfamiddle)/alfawidth);
                alfaconstrained[i] = alfamiddle + th * alfawidth;
            }

            fa=f.res(alfaconstrained);

            for (int i = 0; i < jacfac.length; i++) {
                double alfamiddle = (l[i]+u[i])/2;
                double alfawidth = (u[i]-l[i])/2;
                double th =  Math.tanh((alfanew[i]-alfamiddle)/alfawidth);
                alfaconstrained[i] = alfamiddle + th * alfawidth;
                jacfac[i] = 1 - th * th;
            }
            fan=f.res(alfaconstrained);

            ganho=0;
            for(int j=0; j<n;j++) ganho+=h[j]*(mi*h[j]-g[j]);
            double ro=( Mat.prod(fa, fa)- Mat.prod(fan,fan))/ganho;
            if( ro > 0){
                Mat.copy(alfanew, n, alfa);
                J= f.jacdisc(alfaconstrained);
                for(int i = 0; i < J.length; i++)
                    for(int j = 0; j < jacfac.length; j++)
                      J[i][j] *= jacfac[j];
                A= Mat.prod(Mat.transp(J), J);
                g= Mat.prod(Mat.transp(J), fan);
                erro=Mat.norminf(g);
                found=erro<=EPS1;
                mi=mi*Math.max(0.3333333333333,1-Math.pow(2*ro-1,3)); v=2;
            }
            else
            {
                mi=mi*v; v=2*v;
            }
        }

    }

    System.out.println("Iteracoes da lsqnonlin = "+k);
    System.out.println("Erro final = "+erro+"\n");

    for (int i = 0; i < jacfac.length; i++) {
        double alfamiddle = (l[i]+u[i])/2;
        double alfawidth = (u[i]-l[i])/2;
        double th =  Math.tanh((alfa[i]-alfamiddle)/alfawidth);
        alfa[i] = alfamiddle + th * alfawidth;
    }

    alfa = tiny(alfa,l,u);
```

```
      return alfa;
   }// levenberg_marquardt

   /**
    * Método que resolve um problema de mínimos quadrados não
    * lineares utilizando Gauss-Newton.
    * @param f Função contendo os resíduos cuja soma
    * quadrática será minimizada.
    * @param alfa0 Chute inicial.
    * @param arqres Arquivo .txt contendo os resultados
    * fornecidos pelo mÃ©todo.
    * @return Vetor contendo a solução do problema.
    * @throws java.io.IOException
    */
   public static double[] levenberg_marquardt(FunMQNL f, double[]alfa0,
         FileWriter arqres) throws IOException {
      int n=alfa0.length;
      String novaLinha = System.getProperty("line.separator");
      System.out.println("Numero de parametros  dados = "+n);
      arqres.write("Numero de parametros  dados = "+n+novaLinha);
      int KMAX=  500 ;
      int k=0;
      double erro=1e10, v=2, tau=1e-3;
      final double  EPS1=1e-5;
      final double  EPS2=1e-5;
      arqres.write(Mat.toStr(alfa0, novaLinha+" Vetor inicial dado"));
      Mat.print(alfa0, "\n Vetor inicial dado");
      double[] alfa = new double[n];
      Mat.copy(alfa0, n, alfa);

      double [] alfanew;
      double [][] J  = f.jacdisc(alfa);
      double []   fa = f.res(alfa);
      double [][] A =  Mat.prod(Mat.transp(J), J);
      double []   fan ;
      double []   g= Mat.prod(Mat.transp(J), fa);
      double []   gm = new double [n] ;
      boolean found = Mat.norminf(g)<EPS1;
      double[] diag = new double[n];
      for(int i=0; i < n; i++) {
         diag[i]=A[i][i];
      }
      double mi=tau*Mat.max(diag),ganho;

      while (!found && k<KMAX){
         k++;
         for(int j=0; j<n; j++) A[j][j] = A[j][j] + mi;
         for(int j=0;j<n;j++)   gm[j]=-g[j];
         double[] h = Mat.solve_cholesky(A, gm);
         erro= Mat.normdois(h);
//       System.out.println("erro = "+erro);
         if( erro <= EPS2*(Mat.normdois(alfa)+EPS2) ){
            found=true;
         }
         else
         {
            alfanew=Mat.sum(alfa, h);
```

```
            fa=f.res(alfa);
            fan=f.res(alfanew);
            ganho=0;
            for(int j=0; j<n;j++) ganho+=h[j]*(mi*h[j]-g[j]);
            double ro=(Mat.prod(fa, fa)- Mat.prod(fan,fan))/ganho;
            if( ro > 0){
               Mat.copy(alfanew, n, alfa);
               J= f.jacdisc(alfa);
               A= Mat.prod(Mat.transp(J), J);
               g= Mat.prod(Mat.transp(J), fan);
               erro=Mat.norminf(g);
               found=erro<=EPS1;
               mi=mi*Math.max(0.3333333333333,1-Math.pow(2*ro-1,3)); v=2;
            }
            else
            {
               mi=mi*v; v=2*v;
            }
         }
      }

      System.out.println("Iteracoes da lsqnonlin = "+k);
      System.out.println("Erro final = "+erro+"\n");
      arqres.write("Iteracoes da lsqnonlin = "+k+novaLinha);
      arqres.write("Erro final = "+erro+novaLinha+novaLinha);
      return alfa;
   }// levenberg_marquardt

   /**
   * Método que resolve um problema de mínimos quadrados não
   * lineares utilizando Levenberg-Marquardt com restrições
   * em caixa.
   * @param f Função contendo os resíduos cuja soma
   * quadrática será minimizada.
   * @param alfa0 Chute inicial.
   * @param l Vetor contendo o limite inferior.
   * @param u Vetor contendo o limite superior.
   * @param arqres Arquivo .txt contendo os resultados
   * fornecidos pelo método.
   * @return Vetor contendo a solução do problema.
   * @throws java.io.IOException
   */
   public static double[] levenberg_marquardt(FunMQNL f, double[]alfa0,
         double[] l, double[] u, FileWriter arqres) throws IOException {
      int n=alfa0.length;
      String novaLinha = System.getProperty("line.separator");
      System.out.println("Numero de parametros  dados = "+n);
      arqres.write("Numero de parametros  dados = "+n+novaLinha);
      int KMAX=  500 ;
      int k=0;
      double erro=1e10, v=2, tau=1;
      final double  EPS1=1e-5;
      final double  EPS2=1e-5;
      Mat.print(alfa0, "\n Vetor inicial dado");
      System.out.println("l = " + Arrays.toString(l));
      System.out.println("u = " + Arrays.toString(u));
      arqres.write(Mat.toStr(alfa0, novaLinha+" Vetor inicial dado"));
```

```
        arqres.write((("l = " + Arrays.toString(l) + novaLinha));
        arqres.write((("u = " + Arrays.toString(u) + novaLinha));

        double[] alfa = new double[n];
        Mat.copy(alfa0,n,alfa);

        double [] alfanew;
        double[] alfaconstrained = new double[alfa.length];
        double[] jacfac = new double[alfa.length];

        for(int i = 0; i < n; i++) {
           if(u[i] <= l[i])
              throw new RuntimeException("Os valores de u devem "
                 + "ser maiores do que l");
        }

        for(int i = 0; i < n; i++) {
           if(alfa[i] < l[i])
              alfa[i] = l[i];
           else if(alfa[i] > u[i])
              alfa[i] = u[i];
        }

        for (int i = 0; i < jacfac.length; i++) {
           double alfamiddle = (l[i]+u[i])/2;
           double alfawidth = (u[i]-l[i])/2;
           double th =  Math.tanh((alfa[i]-alfamiddle)/alfawidth);
           alfaconstrained[i] = alfamiddle + th * alfawidth;
           jacfac[i] = 1 - th * th;
        }

        double [][] J  = f.jacdisc(alfaconstrained);
        for(int i = 0; i < J.length; i++)
           for(int j = 0; j < jacfac.length; j++)
              J[i][j] *= jacfac[j];
        double []   fa = f.res(alfaconstrained);
        double [][] A =  Mat.prod(Mat.transp(J), J);
        double []   fan ;
        double []   g= Mat.prod(Mat.transp(J), fa);
        double []   gm = new double [n] ;
        boolean found = Mat.norminf(g)<EPS1;
        double[] diag = new double[n];
        for(int i=0; i < n; i++) {
           diag[i]=A[i][i];
        }
        double mi=tau*Mat.max(diag),ganho;

        while (!found && k<KMAX){
           k++;
           for(int j=0; j<n; j++) A[j][j] = A[j][j] + mi;
           for(int j=0;j<n;j++)   gm[j]=-g[j];
           double[] h = Mat.solve_cholesky(A, gm);
           erro= Mat.normdois(h);
//         System.out.println("erro = "+erro);
           if( erro <= EPS2*(Mat.normdois(alfa)+EPS2) ){
              found=true;
           }
```

```
        else
        {
           alfanew=Mat.sum(alfa, h);

           for (int i = 0; i < jacfac.length; i++) {
              double alfamiddle = (l[i]+u[i])/2;
              double alfawidth = (u[i]-l[i])/2;
              double th =  Math.tanh((alfa[i]-alfamiddle)/alfawidth);
              alfaconstrained[i] = alfamiddle + th * alfawidth;
           }

           fa=f.res(alfaconstrained);

           for (int i = 0; i < jacfac.length; i++) {
              double alfamiddle = (l[i]+u[i])/2;
              double alfawidth = (u[i]-l[i])/2;
              double th =  Math.tanh((alfanew[i]-alfamiddle)/alfawidth);
              alfaconstrained[i] = alfamiddle + th * alfawidth;
              jacfac[i] = 1 - th * th;
           }
           fan=f.res(alfaconstrained);

           ganho=0;
           for(int j=0; j<n;j++) ganho+=h[j]*(mi*h[j]-g[j]);
           double ro=( Mat.prod(fa, fa)- Mat.prod(fan,fan))/ganho;
           if( ro > 0){
              Mat.copy(alfanew, n, alfa);
              J= f.jacdisc(alfaconstrained);
              for(int i = 0; i < J.length; i++)
                  for(int j = 0; j < jacfac.length; j++)
                    J[i][j] *= jacfac[j];
              A= Mat.prod(Mat.transp(J), J);
              g= Mat.prod(Mat.transp(J), fan);
              erro=Mat.norminf(g);
              found=erro<=EPS1;
              mi=mi*Math.max(0.3333333333333,1-Math.pow(2*ro-1,3)); v=2;
           }
           else
           {
              mi=mi*v; v=2*v;
           }
        }

     }

     System.out.println("Iteracoes da lsqnonlin = "+k);
     System.out.println("Erro final = "+erro+"\n");
     arqres.write("Iteracoes da lsqnonlin = "+k+novaLinha);
     arqres.write("Erro final = "+erro+novaLinha+novaLinha);

     for (int i = 0; i < jacfac.length; i++) {
        double alfamiddle = (l[i]+u[i])/2;
        double alfawidth = (u[i]-l[i])/2;
        double th =  Math.tanh((alfa[i]-alfamiddle)/alfawidth);
        alfa[i] = alfamiddle + th * alfawidth;
     }
```

```
      alfa = tiny(alfa,l,u);
      return alfa;
   }// levenberg_marquardt

   public static double[] tiny(double[] x, double[] l, double[] u) {
      int n=x.length;
      double EPS=1e-5;
      double[] y = new double[n];
      Mat.copy(x, n, y);

      for(int i = 0; i < n; i++) {
         if(Math.abs(x[i]-l[i]) < EPS) y[i] = l[i];
         if(Math.abs(x[i]-u[i]) < EPS) y[i] = u[i];
      }

      return y;
   }

    public static double tiny(double x) {
      double EPS=1e-5;
      if(Math.abs(x-Math.round(x)) < EPS) x = Math.round(x);

      return x;
   }

   /* Mínimos Quadrados Lineares
    * Y como função de x   Y=a0+a1*X
    * X como função de y   X=b0+b1*Y
    * Reta Bissetriz   Y=c0+c1*X
    */
   public static double[] minqua(double []x, double []y){
      double sx=0,sxx=0,sy=0,syy=0,sxy=0,db;
      double a0,a1,b0,b1,c0,c1,d;
      double theta1,theta2,theta,x0,y0;
      int n= x.length;
      for(int i=0;i<n;i++){
         sx+=x[i];
         sy+=y[i];
         sxx+=x[i]*x[i];
         syy+=y[i]*y[i];
         sxy+=x[i]*y[i];
      }
      d=n*sxx-sx*sx;
      db=n*sxy-sx*sy;
      a1=db/d;
      a0=sy/n-a1*sx/n;
      d=n*syy-sy*sy;
      b1=db/d;
      b0=sx/n-b1*sy/n;
      theta1=Math.atan(a1);
      theta2=Math.atan(1/b1);
      theta=(theta1+theta2)/2;
      c1=Math.tan(theta);
      y0=sy/n; x0=sx/n;
      c0=y0-c1*x0;
      double []v = new double[6];
      v[0]=a0;
```

```
    v[1]=a1;
    v[2]=b0;
    v[3]=b1;
    v[4]=c0;
    v[5]=c1;
    return v;
  }
```