Universidade Estadual de Campinas
Instituto de Matemática, Estatística e
Computação Científica - IMECC

Sobre o desempenho de métodos Quase-Newton e aplicações

Carlos Alberto Sassi

carlosasassi@hotmail.com

Dissertação de Mestrado Profissional Matemática

Orientadora: **Profa. Dra. Maria Aparecida Diniz Ehrhardt**

Outubro de 2010
Campinas - SP

Sobre o desempenho de métodos Quase-Newton e aplicações

Este exemplar corresponde à redação final da dissertação devidamente corrigida e defendida por **Carlos Alberto Sassi** e aprovada pela comissão julgadora.

Campinas, Outubro de 2010.

Prof. (a). Dr (a). MARIA APARECIDA DINIZ EHRHARDT

Banca Examinadora:

Prof^a. Dr^a. Maria Aparecida Diniz Ehrhardt (orientadora)

Prof. Dr. Marcos Eduardo R. do Valle Mesquita (UEL - Londrina - Paraná)

Prof^a. Dr^a. Vera L. da Rocha Lopes (Unicamp - IMECC)

Prof. Dr. Mario Cesar Zambaldi (UFSC - Santa Catarina)

Prof^a. Dr^a. Sandra Augusta Santos (Unicamp - IMECC)

Dissertação apresentada ao Instituto de Matemática, Estatística e Computação Científica, **UNICAMP**, como requisito para obtenção de Título de **Mestre em Matemática Profissional**.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Sassi, Carlos Alberto
Sa79s Sobre o desempenho de métodos quase Newton e aplicações/Carlos Alberto Sassi-- Campinas, [S.P. : s.n.], 2010.

Orientador : Maria Aparecida Diniz Ehrhardt
Dissertação (mestrado profissional) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Computação Científica.

1.Newton-Raphson, Método. 2.Sistemas não lineares. 3.Shamanski, Método. 4.Algoritmos. I. Diniz-Ehrhardt, Maria Aparecida. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Computação Científica. III. Título.

Título em inglês: Performance of quasi-Newton methods

Palavras-chave em inglês (Keywords): 1. Method, Newton-Raphson. 2. Nonlinear systems. 3. Shamanski Method. 4. Algorithms.

Área de concentração: Matemática Aplicada

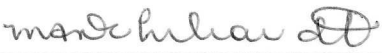
Titulação: Mestre em Matemática Aplicada

Banca examinadora: Profa. Dra. Maria Aparecida Diniz Ehrhardt. (IMECC-UNICAMP)
Profa. Dra. Vera Lúcia da Rocha Lopes (IMECC - UNICAMP)
Prof. Dr. Marcos Eduardo R. Valle Mesquita (IME - UEL)

Data da defesa: 04/10/2010

Programa de Pós-Graduação: Mestrado Profissional em Matemática

Dissertação de Mestrado Profissional defendida em 04 de outubro de 2010 e aprovada pela Banca Examinadora composta pelos Profs. Drs.



Prof. (a). Dr (a). MARIA APARECIDA DINIZ EHRHARDT

Prof. (a). Dr (a). VERA LUCIA DA ROCHA LOPES

Prof. (a). Dr (a). MARCOS EDUARDO RIBEIRO DO VALLE MESQUITA

Dedico este trabalho ao meu filho, Carlos Alberto Sassi Junior que sempre me apoiou em todos os momentos difícies da minha trajetória e à minha mulher Marlene que sempre esteve ao meu lado nestes momentos, e pela colaboração na correção do presente texto em relação a língua mãe.

Agradecimentos

Gostaria de deixar aqui registrado meus sinceros agradecimentos a todos os mestres que direta ou indiretamente contribuíram para a consecução do objetivo deste trabalho, e em especial à Profa. Dra. Maria Aparecida Diniz Ehrhardt minha orientadora em todos os detalhes que este envolveu, principalmente aos computacionais.

Gostaria também de agradecer aos professores: Edmundo Capelas de Oliveira, José Plínio de Oliveira Santos, Sandra Augusta Santos, Sueli I. Rodrigues Costa, Vera Lucia da Rocha Lopes, bem como aos colegas Agnaldo Ferrari, Cristiano Torrezan, e João Paulo Bressan.

Resumo

Iniciamos este trabalho com o estudo de equações não lineares, transcendentais de uma única variável, com o objetivo principal de abordar sistemas de equações não lineares, analisar os métodos, algoritmos e realizar testes computacionais, embasados na plataforma MatLab "The Language of Technical computer - R2008a - version 7.6.0.324". Os algoritmos tratados se referem ao método de Newton, métodos Quase-Newton, método Secante e aplicações, com enfoque na H-equação de Chandrasekhar.

Estudamos aspectos de convergência de cada um destes métodos que puderam ser analisados na prática, a partir dos experimentos numéricos realizados.

Abstract

This work begins with the study of nonlinear and transcendental equations, with only one variable, which has the main purpose to study systems of nonlinear equations, methods and algoritms, in order to accomplish computational tests using MatLab Codes "The Language of Technical computer - R2008a - version 7.6.0.324". These algoritms were concerned to Newton's method, Quasi-Newton method, Secant method, and the main application was the Chandrasekhar H-Equation.

Convergence studies for these methods were analysed with the applied numerical methods.

Sumário

Introdução

O procedimento usado em geral por cientistas e engenheiros para estudar um dado fenômeno da natureza consiste em encontrar um modelo matemático que descreve os aspectos mais importantes do fenômeno e posteriormente recorrer a ferramentas matemáticas para analisar o modelo obtido.

Em grande parte dos casos os modelos são descritos por equações não lineares, e por vezes não é possível obter uma solução analítica para o problema. Nesse caso se torna necessário recorrer a métodos numéricos para se chegar à solução pretendida.

Se para resolver uma equação linear $ax + b = 0$ é trivial, o mesmo já não ocorre se considerarmos uma equação mais complicada.

Passamos a considerar, por exemplo, as equações

(a) $x^4 - 4x^3 - x + 5 = 0$ e

(b) $2e^x - x.sin(x + 3) = 0$.

Ambas são equações não lineares que, no primeiro caso, apesar de existir uma fórmula resolvente geral, é muito complicada. Já no segundo caso se faz necessário recorrer a métodos numéricos.

É portanto mister que para solucionar este(s) tipo(s) de equações, utilizemos métodos numéricos iterativos.

O objetivo deste trabalho é discutir os métodos, algoritmos, e análise envolvida na resolução de sistemas de equações não lineares.

Um sistema de equações não lineares, (também chamado somente de "equações não lineares") está estruturado como segue:

Dada $F : R^n \to R^n$,

determinar $\mathbf{x}^* \in R^n : F(\mathbf{x}^*) = 0 \in R^n$,

onde R^n denota o espaço Euclidiano n-dimensional.

O cenário típico é a solução numérica de problemas não lineares, partindo-se de um ponto inicial $\mathbf{x}^0$, que está próximo da solução $\mathbf{x}^*$. Iremos discutir os métodos básicos e suprir detalhes nos algoritmos que são normalmente considerados para resolução destes problemas.

O projeto foi desenvolvido com a utilização de literatura especializada, vide referências [1] [3],[4],[5],[9].

Foi apresentada uma série de exemplos de sistemas de equações não lineares, que foram apresentados em pequenas dimensões na primeira fase, e posteriormente sistemas não lineares envolvendo grandes dimensões, como o problema **Tridiagonal de Broyden**.

Os experimentos no presente trabalho foram realizados com MATLAB "The language of Technical computer - R2008a - version 7.6.0.324"[9]. [1]

Os programas utilizados foram: **nsol.m** e **brsol.m**, escritos por C.T. Kelley [9]. Os mesmos podem ser obtidos a partir de math.ncsu.edu (152.1.30.3) no diretório **pub/kelley/matlab**, ou também podem ser obtidos de **SIAM's World Wide Web** server,[2] na página http://www.siam.org/books/kelley/kelley.html, e o programa Newton - **newton.m** (vide anexo C).

O presente trabalho foi desenvolvido em capítulos, da seguinte forma:

Capítulo I - Métodos para equações não lineares. Foram aplicados os testes envolvendo equações de uma única variável, que é a introdução para visualizar os procedimentos, os métodos, os algoritmos, e a análise envolvida na solução computacional dos problemas não lineares, através do MatLab Code que servirá de base para o desenvolvimento dos sistemas de multivariáveis. Este capítulo abordou o método de Newton e o método Secante e suas convergências.

Capítulo II - Método de Newton para sistema de equações não lineares. Foram abordadas as resoluções de sistemas de equações não lineares, com aplicação do método de Newton e do método de Newton Modificado, bem como uma análise de convergência.

Capítulo III - Métodos Quase-Newton para equações não lineares. Foram aplicados métodos desse tipo a sistemas não lineares, com destaque para o método de Shamanski,

[1] MATLAB é marca registrada de The MathWorks, Inc.
[2] SIAM - Society for Industrial and Applied Mathematics

testado com vários parâmetros: m=5; m=10; e m=20. Observou-se porém que, para **m=10**, obteve-se os melhores resultados. Foram também aplicados para os mesmos sistemas de equações o método de Broyden, e neste caso, observou-se que o mesmo funcionou relativamente bem para alguns tipos de equações, porém, em outros não, principalmente para os tridiagonais (trid) com dimensões grandes, e para o sistema trigonométrico exponencial (triexp), onde o método não convergiu.

Capítulo IV - H-equação Chandrasekhar. Foram aplicados os mesmos MatLab Code, para a solução da integral de resolução da transferência radioativa, H-equação Chandrasekhar, com discretização do intervalo de integração [0, 1], com os seguintes parâmetros: N =100, c=0,9 e N=100, c=0,9999. Finalizou-se com a análise dos resultados obtidos nos testes computacionais, onde observou-se que a par dos custos envolvidos, o método de Newton nos parece o mais eficiente, pois em alguns dos sistemas de equações, os métodos quase-Newton não obtiveram convergência. Onde ocorreram convegência os tempos computacionais foram bastante próximos.

Capítulo 1

Métodos para equações não lineares

1.1 Introdução

Nesse capítulo abordamos a solução de problemas de equações não lineares para o caso escalar, isto é, estamos interessados em determinar as raízes de uma única equação não linear; posteriormente métodos numéricos para soluções de sistemas de equações não lineares serão abordados.

Existe uma grande quantidade de métodos de resolução para este tipo de problema. Entretanto, dois merecem destaque e são aplicados com maior frequência: **Método de Newton** e o **Secante** [3].

Estes métodos garantem uma boa aproximação para a solução da equação e são relativamente simples, na forma em que são aplicados, pois estas características lhes asseguram grande popularidade.

A razão do estudo de caso de uma variável separadamente é que nos permite compreender os princípios de construções, aproximações lineares, algoritmos, que servirão de bases para algoritmos para problemas envolvendo os casos de várias variáveis.

A formulação inicial do problema é a seguinte: definida uma função não linear $f : \Re \to \Re$, procuramos os valores $\mathbf{x}_*$, tais que $f(\mathbf{x}_*) = 0$, ou ainda, graficamente,

procuramos os pontos onde a curva $f(x)$ intercepta o eixo dos x, como indica a figura (1.1). Estes pontos são as raízes da equação $f(x) = 0$, ou ainda, os zeros de f.

Os métodos são iterativos, de forma que estabelecemos uma função de iteração, que aplicada repetidas vezes, a partir de uma estimativa inicial $\mathbf{x}_0$ - (iteração inicial), produz uma sequência de aproximações que convergem ou não para a solução do problema.

No caso de uma única equação não linear, $f(x) = 0$, um resultado básico do Cálculo Diferencial e Integral é:

Teorema 1.1 *Teorema do valor intermediário: Se $f(x)$ é uma função contínua no intervalo $[a, b]$, e se $f(a).f(b) < 0$, então existe pelo menos uma raiz de $f(x)$ no intervalo considerado $[a, b]$.*

O teorema do valor intermediário nos garante a existência de pelo menos uma raiz, porém, a unicidade é garantida pelo teorema que segue:

Teorema 1.2 *Sob as mesmas hipóteses do teorema anterior e, sendo f diferenciável, tal que a função $f'(x)$ não muda de sinal no intervalo considerado, então existe uma única raiz de $f(x) = 0$ no intervalo [a, b].*

1.2 O que é possível

Consideremos o problema de determinar os zeros de cada uma das funções não lineares em uma única variável [4]:
$f_1(x) = x^4 - 12x^3 + 47x^2 - 60x$
$f_2(x) = x^4 - 12x^3 + 47x^2 - 60x + 24$
$f_3(x) = x^4 - 12x^3 + 47x^2 - 60x + 24.1$

Seria bom que tivéssemos um programa de computador que implementasse a resolução dos problemas citados, informando a partir de um dado $\mathbf{x}_0$, ponto inicial, que: "Os zeros de $f_1(x)$ são $x = 0$ *ou* 3 *ou* 4 *ou* 5"; e que os "zeros de $f_2(x)$ são $x = 1$ ou $x \simeq 0.888$"; enquanto que "$f_3(x)$ não converge para solução real em um número pré-fixado de iterações".

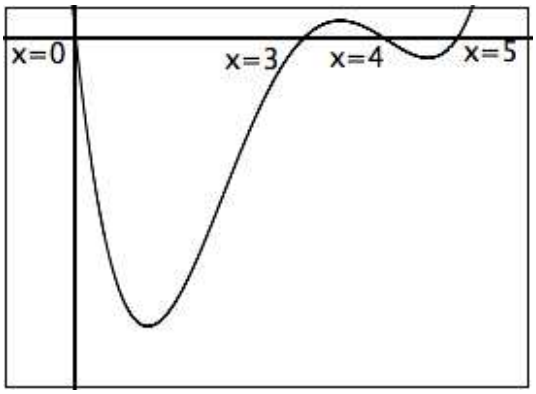


Figura 1.1: gráfico da função $f_1(x)$

A determinação da existência e unicidade da solução de equações não lineares não é nossa primeira preocupação, neste trabalho. É também aparente que podemos encontrar ou não, soluções aproximadas para a maioria dos problemas não lineares. Observamos que, pelo resultado clássico devido a Galois, não temos uma fórmula analítica capaz de determinar os zeros de polinômios de grau $n \geq 5$. Assim, devemos desenvolver métodos que tentam determinar uma solução aproximada de problemas não lineares.

1.3 Método de Newton - equação não linear de uma variável

Seja f a função da figura (1.2), que é diferenciável em um determinado intervalo real. Desde que a tangente ao gráfico de f no ponto $(x_1; f(x_1))$ representa a melhor aproximação linear de f, próximo ao ponto considerado, parece razoável esperar que, se x_1 está próximo da raiz c da equação $f(x) = 0$ então, o ponto x_2, onde a tangente em x_1 intercepta o eixo-x, está também próximo da raiz c, vide figura (1.2). A tangente agora no ponto x_2 produz o próximo valor x_3 e assim iterativamente, de forma a obtermos uma solução tão próxima de c, raiz da equação, quanto desejarmos.

Dado uma função $f : \Re \to \Re$ e $\mathbf{x}_0 \in \Re$, o método de Newton consiste em definir uma sequência x_k através da equação

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \qquad k = 0, 1, 2.... \tag{1.1}$$

desde que $f'(x_k) \neq 0$ para $k = 0, 1, 2, ...$, a partir de um dado x_0. A expressão (1.1) é obtida a partir da raiz da equação da reta tangente no ponto considerado.

Este processo de iteração é conhecido como **Método de Newton** (ou **Método de Newton-Raphson)**, para soluções aproximadas de equações não lineares $f(x) = 0$. Devemos ressaltar que existem situações em que o método falha, como veremos ainda neste capítulo.

As figuras (1.2), (1.3), (1.4) ilustram a conexão entre as sucessivas aproximações $x_k \ e \ x_{k+1}$.

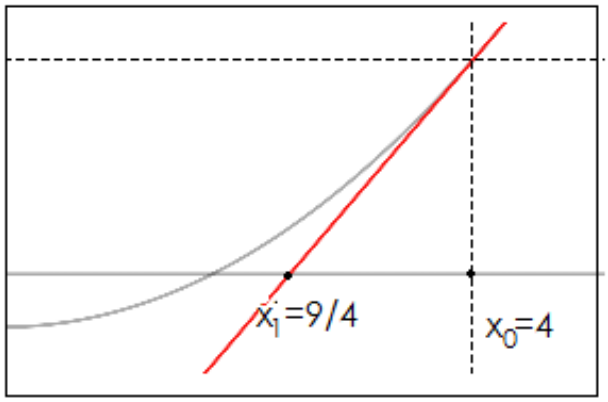


Figura 1.2: Aproximações iniciais sucessivas

Equação da reta tangente a $f(x)$ em $(x_k, f(x_k))$:

$y - f(x_k) = f'(x_k)(x - x_k)$.

$0 - f(x_k) = f'(x_k)(x_{k+1} - x_k)$.

Isolando x_{k+1}, nós temos a equação dada por (1.1).

Exemplo 1.1 *Aproximar* $\sqrt{3}$ *pela aplicação do método de Newton-Raphson à função* $f(x) = x^2 - 3$, *escolhendo* $x_0 = 2$ *como a primeira aproximação [10].*

Desde que $f'(x) = 2x$, temos:

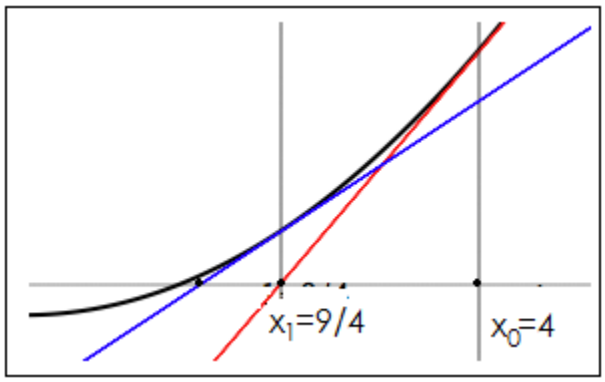


Figura 1.3: Obtenção de uma nova aproximação

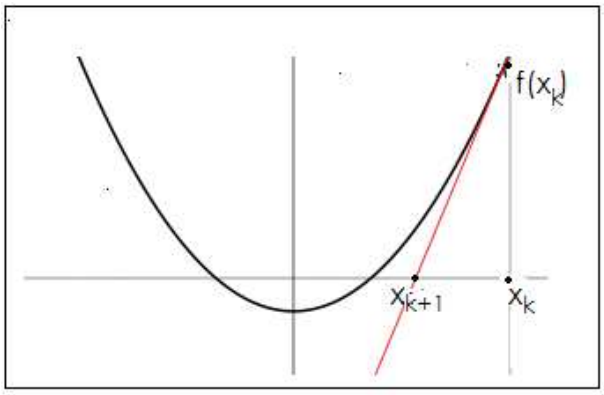


Figura 1.4: Evolução de um ponto (x_k) a um ponto x_{k+1}

$$x_{k+1} = x_k - \frac{(x_k)^2 - 3}{2x_k} = \frac{(x_k)^2 + 3}{2x_k} = \frac{1}{2}(x_k + \frac{3}{x_k})$$

Tabela 1.1: Tabela de valores computados

k	x_k	$\frac{3}{x_k}$	$x_{k+1} = \frac{1}{2}(x_k + \frac{3}{x_k})$
1	2,0000000	1,5000000	1,7500000
2	1,7500000	1,7142857	1,7314280
3	1,7321428	1,7319588	1,7320508

Desde que $(1,7320508)^2 = 2,9999999$ na calculadora usada para executar estes cálculos, o método conseguiu obter uma boa aproximação de solução em apenas 3 iterações.

Exemplo 1.2 *Determinar os zeros da função* $f(x) = cos(x) + x$.*[12]*

Neste caso vamos utilizar um programa do MatLab. (vide Anexo C).

Newton-Raphson Method for solving transcendental equations

Elementos de entrada:

Função $f(x) = cos(x) + x = 0$

Aproximação inicial $x_0 = 1$

Número máximo de iterações = 10.

Critério de parada $|f(x)|_\infty \leq 10^{-6}$

Elementos de saída:

Raiz $= -7.390851.10^{-1}$

Número de iterações = 7

Critério de parada atingido: $|f(x)| \leq 10^{-6}$

Utilizamos um software gráfico Grapher - plataforma iMac para efeito de comparação com o resultado obtido - (vide figuras (1.5) e (1.6)).

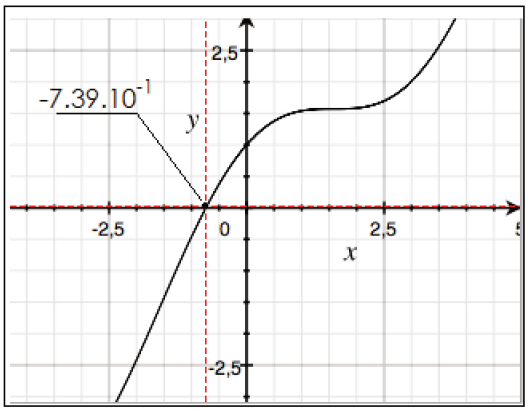


Figura 1.5: y = cos(x) + x

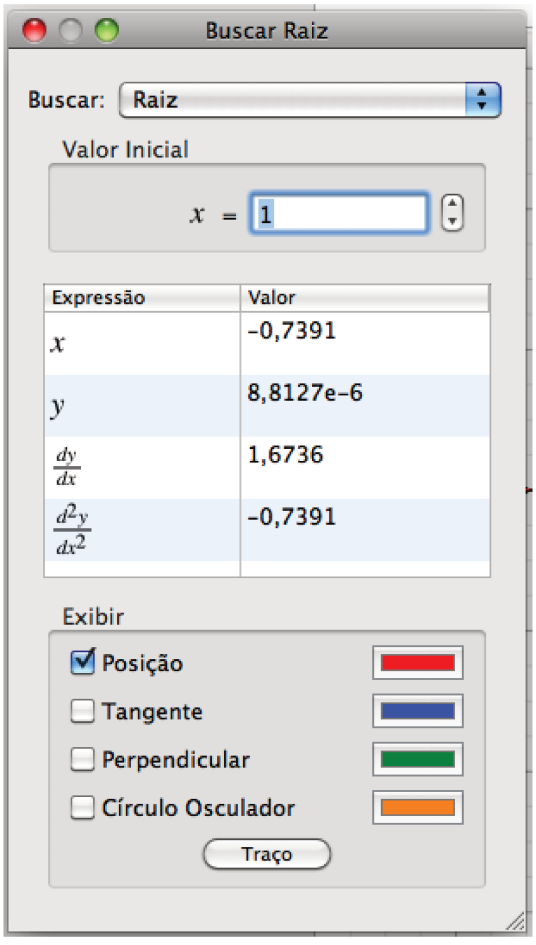


Figura 1.6: Software Grapher-determinação da raiz

Seguem alguns exemplos onde o método falha [8]:

i) Claramente o método falha se acontecer que $f'(x_k) = 0$, ou seja, o coeficiente angular m da reta tangente é zero, com $f(x_k) \neq 0$ - (vide figura (1.7));

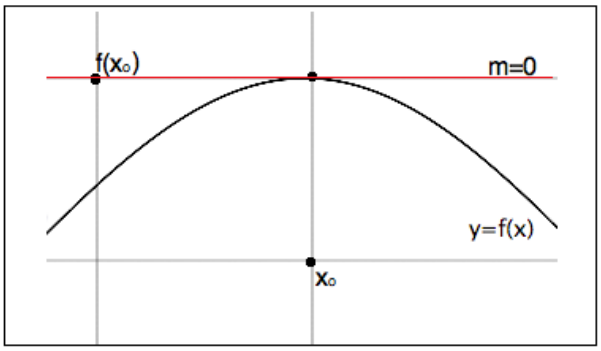


Figura 1.7: Coeficiente angular m da reta tangente em $(x_0; f(x_0))$ é zero

ii) Um outro caso onde o método de Newton falha.

Vejamos a aplicação do método de Newton para resolver a equação:

$$x^3 - x + \frac{\sqrt{2}}{2} = 0 \quad \text{(vide figura 1.8)}$$

Consideremos o ponto inicial $x_0 = 0$, (isto é, nossa iteração inicial $x_0 = 0$).

Como a derivada da função f é $f'(x) = 3x^2 - 1$, temos $f'(0) = -1$, e $f(0) = \frac{\sqrt{2}}{2}$.

Dado que $x_1 = x_0 - \frac{1}{f'(x_0)}.f(x_0)$, ou ainda $x_1 = x_0 - \frac{x_0^3 - x_0 + \frac{\sqrt{2}}{2}}{3x_0^2 - 1} = 0 + \frac{\frac{\sqrt{2}}{2}}{1} = \frac{\sqrt{2}}{2}$,

temos que $x_1 = \frac{\sqrt{2}}{2}$; $f'(x_1) = \frac{1}{2}$

e então temos:

$$x_2 = \frac{\sqrt{2}}{2} - 2.(\frac{\sqrt{2}}{4} - \frac{\sqrt{2}}{2} + \frac{\sqrt{2}}{2}) = 0, \text{ ou seja, } x_2 = x_0 = 0.$$

Desta forma retornamos ao ponto de partida, ou seja $x_0 = 0$, e desta forma, os valores de x_k ficarão variando entre 0 e $\frac{\sqrt{2}}{2}$, de forma a não convergir para qualquer raiz. Ou seja o método de Newton produz a sequência $(0, \frac{\sqrt{2}}{2}, 0, \frac{\sqrt{2}}{2}, ...)$.

No entanto, se aplicarmos o método de Newton a partir de $x_0 = -1$, obtemos os seguintes resultados:

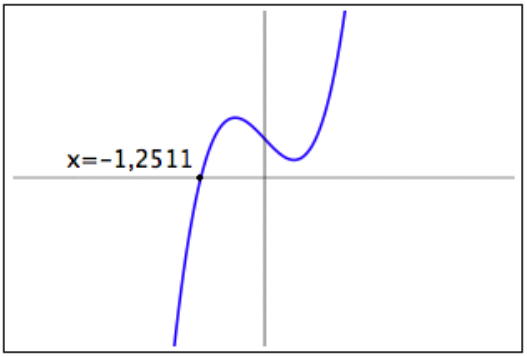


Figura 1.8: Gráfico f(x)=$x^3 - x + \frac{\sqrt{2}}{2}$

Newton-Raphson Method for solving transcendental equations (vide anexo C).

A raiz = -1.251079

Numero de iterações = 4

Resolução da equação, utilizando o Software Grapher (Mac) - vide figura (1.9).

Observe que nos dois exemplos onde o método falha, concluimos que a aproximação da raiz e a própria escolha deste ponto inicial é de suma importância pois, o mesmo pode acarretar convergência ou não à solução.

1.4 Método Secante

Diferentemente do método de Newton que utiliza a reta tangente no ponto considerado para gerar a sequência que converge para a solução da equação $f(x) = 0$, o *método da secante* utiliza a reta secante entre dois pontos da função, $(x_k; f(x_k))$ e $(x_{k-1}; f(x_{k-1}))$ que é uma aproximação para a reta tangente [3] (vide figura 1.10).

Neste caso são necessários dois pontos iniciais x_0 e x_1.

Para estabelecermos a relação de recorrência, utilizamos a semelhança de triângulos, e temos:

$$\frac{f(x_0)}{x_0 - x_2} = \frac{f(x_1)}{x_1 - x_2}$$

Resolvendo para a incógnita x_2, temos:

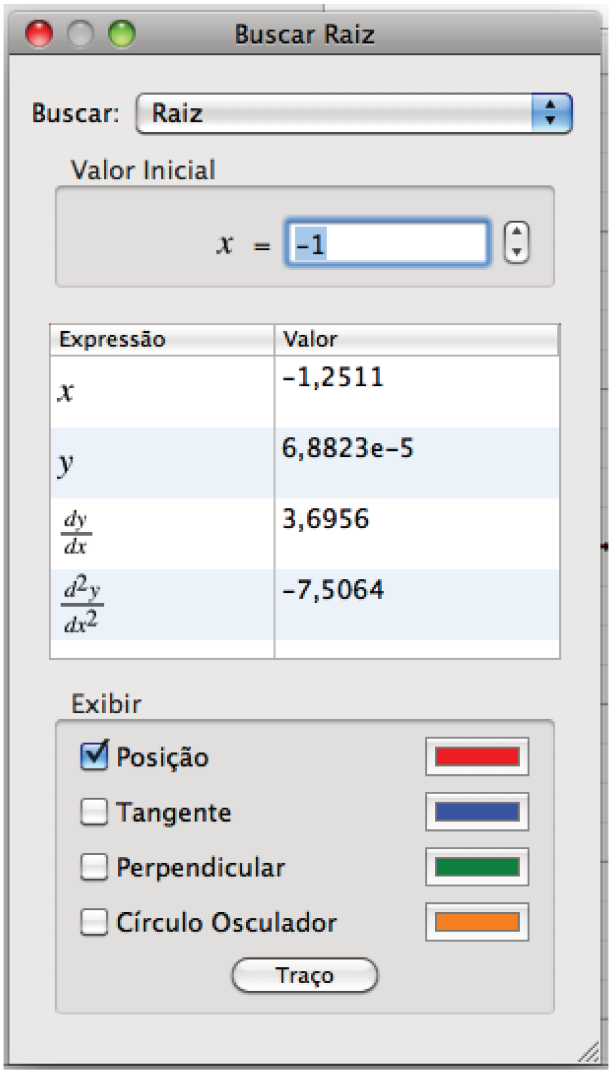


Figura 1.9: Software Grapher - determinação da raiz - equação figura (1.8)

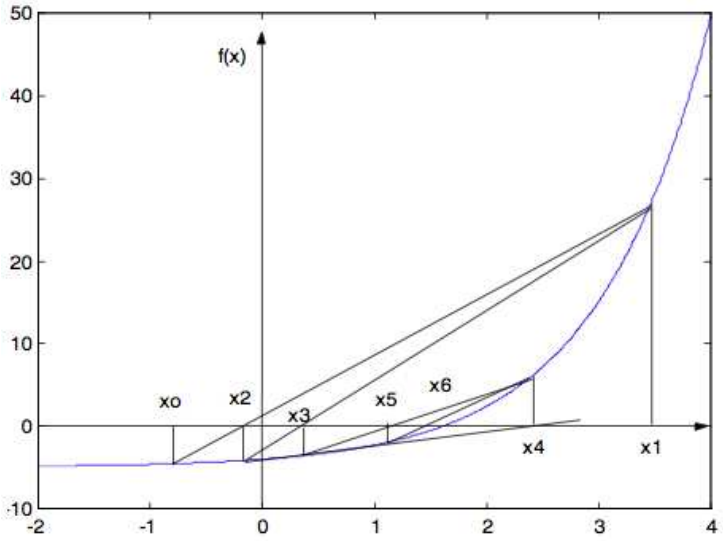


Figura 1.10: Método secante

$$x_2 = \frac{x_0 f(x_1) - x_1 f(x_0)}{f(x_1) - f(x_0)}$$

Podemos chegar a mesma expressão substituindo na iteração de Newton $f'(x_0)$ por $\frac{f(x_1) - f(x_0)}{x_1 - x_0}$, é:

Generalizando, a expressão para o termo geral é: dados x_0, x_1

$$x_{k+1} = \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}. \qquad \text{k} = 2, 3, \ldots$$

Observe que, no método secante não é necessário o cálculo de valores das derivadas de $f(x)$, como era no método de Newton; usamos os próprios valores de $f(x_k)$ e $f(x_{k-1})$, que de qualquer forma seriam calculados. Isto também representa um ganho quando as derivadas são de difíceis expressões pois esses cálculos também envolvem custos computacionais.

Exemplo 1.3 *O método da secante é usado para encontrar a raiz da equação* $cos(x) + x = 0$ *[10]. Tomamos como valores iniciais* $x_0 = 1$ *e* $x_1 = -0.5$ *e tolerância de* 10^{-6}.

Tabela 1.2: Tabela de valores computados

iter.	x_{k-1}	x_k	$f(x_{k-1})$	$f(x_k)$	$\|(x^k - x^{k-1})\|$	$\|\|f\|\|$	x_{k+1}
1	1.000000	-0.500000	1.540302	0.377582	1.500000	1.162719	-0.987111
2	-0.500000	-0.987111	0.377582	-0.436008	0.487111	0.813591	-0.726065
3	-0.987111	-0.726065	-0.436008	0.021727	0.261045	0.457735	-0.738451
4	-0.726065	-0.738456	0.021727	0.001052	0.012391	0.020675	-0.739086
5	-0.738456	-0.739086	0.001052	0.000003	0.000630	0.001055	-0.739085
6	-0.739086	-0.739085	0.000003	-0.000000	0.000001	0.000003	-0.739085
7	-0.739085	-0.739085	-0.000000	-0.000000	0.000000	0.000000	-0.739085

Pelo método da secante, com cálculos passo a passo, obtivemos a aproximação da solução para o exemplo em questão, isto é, $x = -0.73908513$, em 7 iterações. Aqui foi utilizado um programa do MatLab que implementa o método da secante (vide anexo B). Realizamos comparação entre os resultados obtidos pelos métodos: Newton-Raphson e da Secante, implementados nos programas do MatLab, aplicados à resolução da equação $f(x) = cos(x) + x = 0$.

Observamos que ambos os métodos obtiveram, em apenas 7 iterações, uma boa aproximação para a solução do problema em questão. Destacamos que a escolha dos pontos iniciais é fundamental para o desempenho desses métodos iterativos. No caso do método de Newton, um ponto inicial mais próximo da solução, como o valor -0.5 usado na inicialização do método da secante, poderia gerar melhores resultados em relação ao número de iterações.

A partir de um valor inicial próximo da raiz procurada, se a função possui derivadas contínuas, e f' não se anula em x_*, temos a garantia de convergência, tanto no método de Newton, quanto no método da Secante. Quanto à evolução do erro desses métodos, podemos citar o resultado que pode ser encontrado em bibliografia especializada [2].

1.5 Convergência

Dado um processo iterativo que produz uma sequência de pontos $x_1, x_2, ...$, a partir de um ponto inicial x_0, nós estamos interessados em determinar se esta sequência converge para a solução $\mathbf{x}_*$, e ainda quão rapidamente. Se nós assumirmos que esta converge, então as seguintes definições caracterizam as propriedades que serão necessárias.

1.5.1 Algumas Definições:

Definição 1.1 *Seja* $x_* \in \Re$, $x_k \in \Re$, $k = 0, 1, 2, ...$

Então, a sequência: $\{x_k\}$=$\{x_0, x_1, x_2, ...\}$ *é dita convergente a* x_*, *se:*

$$\lim_{k\to\infty} |x_k - x_*| = 0.$$

Definição 1.2 *Se, em adição, existir uma constante* $c \in [0, 1]$, *e um inteiro* $k_* \geq 0$, *tal que para todo* $k \geq k_*$,

$$|x_{k+1} - x_*| \leq c\,|x_k - x_*|,$$

então, $\{x_k\}$ é dita *q-linearmente convergente* a x_*.

Definição 1.3 *Se para alguma sequência* $\{c_k\}$ *que convirja a zero,*

$$|x_{k+1} - x_*| \leq c_k\,|x_k - x_*|,$$

então, $\{x_k\}$ é dita ser *q-superlinearmente convergente* a x_*.

Definição 1.4 *Se existirem constantes* $p > 1$, $c \geq 0$ *e* $k_* \geq 0$, *tais que* $\{x_k\}$ *converge para* x_* *e para todo* $k \geq k_*$,

$$|x_{k+1} - x_*| \leq c\,|x_k - x_*|^p,$$

então, $\{x_k\}$ *é dita convergente a* x_*, *com "q-ordem" pelo menos "p". Se* $p = 2$ *ou* $p = 3$, *a convergência é dita q-quadrática ou q-cúbica, respectivamente.*

Definição 1.5 *Uma função g é dita Lipschitz contínua com constante* γ *em um conjunto X, denotado por* $g \in Lip_\gamma(X)$, *se, para todo* $x, y \in X$;

$$|g(x) - g(y)| \leq \gamma|x - y|.$$

1.5.2 Convergência dos métodos de Newton e Secante

Teorema 1.3 *Seja $f : D \to \Re$, D um dado intervalo aberto, e seja ainda $f' \in Lip_\gamma(D)$. Assuma que para um dado $\rho > 0$, $|f'(x)| \geq \rho$, para todo $x \in D$. Se $f(x) = 0$ tem uma solução $x_* \in D$, então há um $\eta > 0$, tal que:*

Se $|x_0 - x_*| \leq \eta$, então a sequência $\{x_n\}$, gerada por:
$x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}, \qquad para \;\; n = 0, 1, 2, ...$
está bem definida e converge para x_*. Mais ainda, para $n = 0, 1, 2, ...$

$$|x_{n+1} - x_*| \leq \frac{\gamma}{2\rho}.|x_n - x_*|^2$$

.

(Convergência q-quadrática).

Para prova do teorema, vide [4].

Sob as mesmas hipóteses do teorema 1.3, a sequência $\{x_n\}$ gerada pelo método das secantes converge a $\mathbf{x}_*$ e, para n suficientemente grande, $\dfrac{|x^{n+1} - x^*|}{|x^n - x^*|^p} = c$, para alguma constante c não nula e $p \simeq 1.618$ (convergência q-superlinear).

Capítulo 2

Método de Newton para sistemas de equações não lineares

2.1 Introdução

O problema básico a ser estudado neste capítulo é a resolução de sistemas de equações não lineares; a formulação dos mesmos é: Dada uma função $F : \Re^n \rightarrow \Re^n$, determinar $\mathbf{x}^* \in \Re^n$, tal que $F(\mathbf{x}^*) = 0$, onde F é uma função contínua, para que o problema esteja bem definido e diferenciável, para que os métodos que vamos abordar sejam aplicáveis.

Representamos:

$$F(x) = \begin{pmatrix} f_1(x_1, x_2, ...x_n) \\ f_2(x_1, x_2, ...x_n) \\ ..... \\ f_n(x_1, x_2, ...x_n) \end{pmatrix} \tag{2.1}$$

Vamos analisar, inicialmente, os exemplos a seguir:

Exemplo 2.1 *Sejam as funções:* $\begin{cases} f_1(\boldsymbol{x}) = x_1^2 + x_2^2 - 2 \\ f_2(\boldsymbol{x}) = x_1^2 - \dfrac{x_2^2}{9} - 1, \end{cases}$ *onde* $\boldsymbol{x} = (x_1, x_2) \in \Re^2$

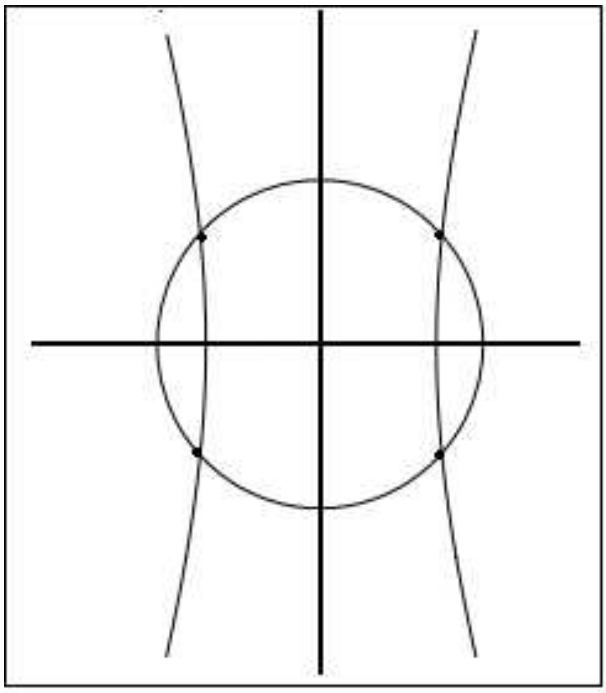


Figura 2.1: Gráfico das funções f_1 *e* f_2 do exemplo 2.1

O sistema de equações $\begin{cases} f_1(x) = 0 \\ f_2(x) = 0 \end{cases}$ admite quatro soluções, que são os pontos onde as curvas $f_1(x)$ e $f_2(x)$ se interceptam como mostra a figura (2.1).

Exemplo 2.2 *Sejam as funções :* $\begin{cases} f_1(x) = x_1^2 + x_2 - 0.2 \\ f_2(x) = x_2^2 - x_1 + 1 \end{cases}$

Neste exemplo observamos que o sistema $\begin{cases} f_1(x) = 0 \\ f_2(x) = 0 \end{cases}$ não tem solução, pois não existem pontos onde as curvas se interceptam, como mostra a figura (2.2).

Para uma dimensão qualquer, a análise de existência e unicidade de solução de um sistema não linear pode se tornar uma questão complexa. Em nosso trabalho, vamos nos concentrar no estudo de métodos numéricos para resolução de equações não lineares, sem entrar em detalhes sobre a teoria da existência e unicidade de solução.

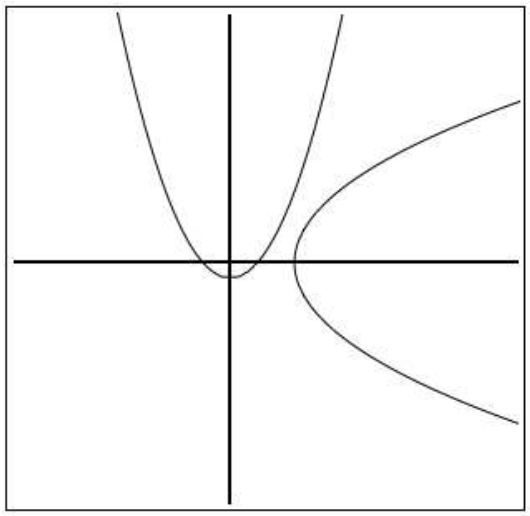


Figura 2.2: Gráfico das funções f_1 e f_2 do exemplo 2.2

2.2 Método de Newton para sistemas de equações não lineares

Seja o conjunto de n equações não lineares em n variáveis:

$$\begin{cases} f_1(x_1, x_2, ...x_n) = 0 \\ f_2(x_1, x_2, ...x_n) = 0 \\ ..... \\ f_n(x_1, x_2, ...x_n) = 0 \end{cases} \tag{2.2}$$

Na forma vetorial, temos: $F(x) = 0$, onde $\mathbf{x} \in \Re^n$ *e* $F(x)$ é definida como:

$$F(x) = \begin{pmatrix} f_1(x_1, x_2, ...x_n) \\ f_2(x_1, x_2, ...x_n) \\ ..... \\ f_n(x_1, x_2, ...x_n) \end{pmatrix}$$

Similarmente ao caso de uma variável, a *matriz Jacobiana*, das derivadas parciais de 1^a ordem de F, é dada por:

$$J(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} \cdots\cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} \cdots\cdots & \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \cdots\cdots & & \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \frac{\partial f_n(\mathbf{x})}{\partial x_2} \cdots\cdots & \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{pmatrix}$$

O método de Newton consiste em fazer uma aproximação linear $L_k(\mathbf{x}) \simeq F(\mathbf{x}^k) + J(\mathbf{x}^k).(\mathbf{x} - \mathbf{x}^k)$, da função F no ponto $\mathbf{x}^k$ dado, representada por planos ou hiperplanos. A partir da utilização destas aproximações lineares para $k = 0, 1, 2, ...$, seguem-se as iterações, gerando uma sequência que, sob certas hipóteses, converge para a solução.

Como em qualquer método iterativo, precisamos estabelecer critérios de parada para aceitar um ponto $\mathbf{x}^k$ como uma boa aproximação da solução exata $\mathbf{x}^*$, ou ainda para detectarmos a convergência lenta do processo. Uma vez que, na solução exata $\mathbf{x}^*$, temos: $F(\mathbf{x}^*) = 0$, um critério de parada consiste em verificar se todas as componentes de $F(\mathbf{x}^k)$ têm módulo pequeno. Como $F(\mathbf{x}^k)$ é um vetor no $\Re^n$, verificamos se a norma de $F(\mathbf{x}^k)$ é menor que um dado ϵ, isto é: $||F(x^k)|| < \epsilon$, para $\epsilon > 0$ suficientemente pequeno.

Outro critério de parada consiste em verificar se a norma entre duas iterações sucessivas está próxima de zero, isto é, escolhemos $\mathbf{x}^{k+1}$ como uma boa aproximação para a solução exata $\mathbf{x}^*$, se temos $||\mathbf{x}^{k+1} - \mathbf{x}^k|| < \epsilon$; esse critério é denominado de erro absoluto. Pode-se alternativamente usar o erro relativo, que também consiste em um critério de parada, isto é: $\dfrac{||\mathbf{x}^{k+1} - \mathbf{x}^k||}{||\mathbf{x}^{k+1}||} < \epsilon$.

O método mais amplamente utilizado e conhecido para resolver sistemas de equações não lineares e transcendentais é o método de Newton (também conhecido como método de **Newton - Puro**).

Cada iteração k do método de Newton consiste na obtenção de um zero da aproximação linear $L_k(\mathbf{x})$, o qual será a nova estimativa x^{k+1}. Assim o método requer basicamente, a cada iteração:

i) A avaliação da matriz Jacobiana no ponto em questão, $J(\mathbf{x}^k)$.

ii) A resolução do sistema linear $J(\mathbf{x}^k).s^k = -F(\mathbf{x}^k)$.

iii) A atualização do novo valor $\mathbf{x}^{k+1} = \mathbf{x}^k + s^k$.

Exemplo 2.3 *Resolver o sistema de equações a seguir:*

$$\begin{cases} f_1(x) = ln(x_1^2 + 2x_2^2 + 1) - 0.5 = 0. \\ f_2(x) = x_2 - x_1^2 + 0.2 = 0 \end{cases} \tag{2.3}$$

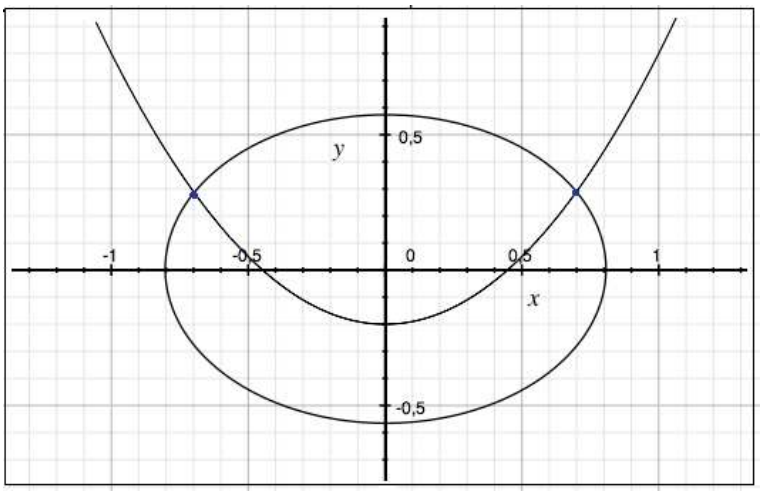


Figura 2.3: Gráfico das funções f_1 e f_2 do exemplo 2.3

Neste exemplo vamos resolver o sistema de duas formas:

i) Executando inicialmente, passo a passo, todas as operações envolvidas nas iterações.

ii) Resolvendo o sistema usando um programa MatLab envolvendo operações com matrizes, onde pode-se ressaltar que a operação de divisão tem duas formas a saber [6]:

Divisão à esquerda - Esta divisão à esquerda é usada para resolver a equação matricial $A\mathbf{x} = \mathbf{b}$, onde A é matriz não singular. Nesta equação $\mathbf{x}$ e $\mathbf{b}$ são vetores

coluna. Esta equação pode ser resolvida pela multiplicação à esquerda, dos dois lados, pela inversa de A.

$$A^{-1}A\mathbf{x} = A^{-1}\mathbf{b}$$

O lado esquerdo desta equação é $\mathbf{x}$, desde que:

$$A^{-1}A\mathbf{x} = I\mathbf{x} = \mathbf{x}$$

Então a solução de $A\mathbf{x} = \mathbf{b}$ é: $\mathbf{x} = A^{-1}\mathbf{b}$

No MatLab, a última equação pode ser escrita usando o caracter de divisão à esquerda, isto é:

$$\mathbf{x} = A \setminus \mathbf{b}$$

O método pelo qual o MatLab calcula $\mathbf{x}$ é diferente. No primeiro caso o MatLab inicialmente calcula a inversa A^{-1} e a usa para multiplicar por $\mathbf{b}$. No segundo caso (divisão à esquerda) a solução é obtida numericamente com um método baseado na eliminação de Gauss. O método da divisão à esquerda é recomendado para resolver conjuntos de sistemas de equações lineares, pois o método da inversa pode ser menos preciso que a eliminação de Gauss, quando aplicado em matrizes de grandes dimensões [7].

Divisão à direita - A divisão à direita é usada para resolver a equação matricial $\mathbf{x}C = \mathbf{d}$. Nesta equação $\mathbf{x}$ e $\mathbf{d}$ são vetores linha. Esta equação pode ser resolvida pela multiplicação pela direita ambos os lados pela inversa de C:

$$\mathbf{x}.CC^{-1} = \mathbf{d}C^{-1}$$

portanto $\mathbf{x} = \mathbf{d}.C^{-1}$.

No MatLab esta última equação pode ser escrita usando o caracter de divisão à direita.

$$\mathbf{x} = \mathbf{d}/C$$

Resolução passo a passo

Vamos utilizar como primeira aproximação $\mathbf{x}^0 = (1, 1)^t$, de forma a obter a solução positiva do sistema do exemplo (2.3), com critério de parada : $||F(\mathbf{x}^k)||_\infty < 10^{-6}$.

Avaliação da matriz Jacobiana:

$$J(x_1,x_2)=\begin{pmatrix} \dfrac{\partial f_1(x_1,x_2)}{\partial x_1} & \dfrac{\partial f_1(x_1,x_2)}{\partial x_2} \\ \dfrac{\partial f_2(x_1,x_2)}{\partial x_1} & \dfrac{\partial f_2(x_1,x_2)}{\partial x_2} \end{pmatrix}$$

$$J(x_1,x_2)=\begin{pmatrix} \dfrac{2x_1}{x_1^2+2x_2^2+1} & \dfrac{4x_2}{x_1^2+2x_2^2+1} \\ -2x_1 & 1 \end{pmatrix}$$

Primeira iteração:

i) Avaliar a Jacobiana no ponto considerado , isto é: $J(x^0)$, e a função, $F(x^0)$

ii) Resolver o sistema $J(x^0).(s^0) = -F(x^0)$

iii) Determinar o novo ponto $x^1 = x^0 + s^0$, e assim sucessivamente.

Assim, para o ponto inicial $x^0 = [1; 1]$, temos:

$$\text{J(1,1)}=\begin{pmatrix} \dfrac{2}{1^2+2.1^2+1} & \dfrac{4.1}{1^2+2.1^2+1} \\ -2.1 & 1 \end{pmatrix}$$

$$J(1,1) = \begin{pmatrix} \dfrac{1}{2} & 1 \\ -2 & 1 \end{pmatrix}$$

$f_1(1,1) = ln(1^2 + 2.1^2 + 1) - 0.5 = ln(4) - 0.5 = 0.8863$

$f_2(1,1) = 1 - 1^2 + 0.2 = 0.2$

Resolução do sistema: $J(1,1).(s^0) = -F(x^0)$

$$\begin{pmatrix} \dfrac{1}{2} & 1 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} s^{0_1} \\ s^{0_2} \end{pmatrix} = \begin{pmatrix} -0.8863 \\ -0.2000 \end{pmatrix}$$

$\begin{pmatrix} s^{0_1} \\ s^{0_2} \end{pmatrix} = \begin{pmatrix} -0.2745 \\ -0.7491 \end{pmatrix}$ e assim sucessivamente, obtendo-se:

$x^0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ $x^1 = \begin{pmatrix} 0.7255 \\ 0.2510 \end{pmatrix}$ $x^2 = \begin{pmatrix} 0.6982 \\ 0.2868 \end{pmatrix}$ $x^3 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$ $x^4 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$,

$||F(x^4)|| = 1.51110^{-10}$

Resolução do sistema utilizando programa do MatLab (vide anexo C)

Este programa usa a divisão à esquerda. Introduzimos uma nova variável $G = -F(x)$ para efeito de simplificação.

Dados de entrada do programa:

$[x, k] = newton(0.000001)$

aproximação inicial [1; 1]

Tabela 2.1: 1^a Iteração

item	Valor
$x_1^0 =$	1
$x_2^0 =$	1
f_1	0.8863
f_2	0.2000
$G =$	$\begin{pmatrix} -0.8863 \\ -0.2000 \end{pmatrix}$
$J =$	$\begin{pmatrix} 0.5000 & 1.000 \\ -2.000 & 1.000 \end{pmatrix}$
$s = J \setminus G$	$\begin{pmatrix} -0.2745 \\ -0.7490 \end{pmatrix}$
$x^1 =$	$\begin{pmatrix} 0.7255 \\ 0.2510 \end{pmatrix}$

Tabela 2.2: 2^a Iteração

item	Valor
$x_1^1 =$	0.7255
$x_2^1 =$	0.2510
f_1	0.0022
f_2	-0.0754
$G =$	$\begin{pmatrix} -0.0022 \\ 0.0754 \end{pmatrix}$
$J =$	$\begin{pmatrix} 0.8782 & 0.6076 \\ -1.4510 & 1.0000 \end{pmatrix}$
$s = J \setminus G$	$\begin{pmatrix} -0.0272 \\ 0.0358 \end{pmatrix}$
$x^2 =$	$\begin{pmatrix} 0.6982 \\ 0.2668 \end{pmatrix}$

Tabela 2.3: 3^a Iteração

item	Valor
$x_1^2 =$	0.6982
$x_2^2 =$	0.2868
f_1	0.0020
f_2	-0.0007
$G =$	$\begin{pmatrix} -0.0020 \\ 0.0007 \end{pmatrix}$
$J =$	$\begin{pmatrix} 0.8453 & 0.6944 \\ -1.3965 & 1.0000 \end{pmatrix}$
$s = J \setminus G$	$\begin{pmatrix} -0.0014 \\ -0.0012 \end{pmatrix}$
$x^3 =$	$\begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$

Tabela 2.4: 4^a Iteração

item	Valor
$x_1^3 =$	0.6968
$x_2^3 =$	0.2856
f_1	$0.0892.10^{-5}$
f_2	$-0.1925.10^{-5}$
$G =$	$\begin{pmatrix} -0.0892.10^{-5} \\ 0.1925.10^{-5} \end{pmatrix}$
$J =$	$\begin{pmatrix} 0.8453 & 0.6929 \\ -1.3937 & 1.0000 \end{pmatrix}$
$s = J \setminus G$	$\begin{pmatrix} -0.1229.10^{-5} \\ 0.0212.10^{-5} \end{pmatrix}$
$x^4 =$	$\begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$

critério de parada atingido $||F(x^4)|| = 1.51110^{-10}$

$$x^0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad x^1 = \begin{pmatrix} 0.7255 \\ 0.2510 \end{pmatrix} \quad x^2 = \begin{pmatrix} 0.6982 \\ 0.2868 \end{pmatrix} \quad x^3 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix} \quad x^4 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$$

$k = 4$ iterações.

Com o programa Matlab, obtivemos também o valor $x^4 = [0.6968, 0.2856]$ após 4 iterações, com critério de parada satisfeito $||F|| = 1.51110^{-10}$.

Devemos, no entanto, observar que a avaliação da matriz Jacobiana, e a resolução do sistema linear $J(x^k)s^k = -F(x^k)$, para todo $k = 0, 1, 2, ...$, torna a iteração do método computacionalmente cara. Alternativas para contornar estas dificuldades serão discutidas no final deste capítulo e no próximo. Por outro lado, a grande vantagem do método de Newton é a sua velocidade de convergência, como veremos a seguir.

Convergência

No teorema seguinte vemos que a convergência local no Método de Newton é q-quadrática; para demonstração vide [4].

Teorema 2.1 *Seja $F : \Re^n \to \Re^n$ uma função contínua e derivável em um conjunto convexo aberto $D \subset \Re^n$. Suponha que existe $\boldsymbol{x}^* \in \Re^n$ e r, $\beta > 0$, tais que a vizinhança*

$\vartheta(\boldsymbol{x}^*, r) \subset D$, $F(\boldsymbol{x}^*) = 0$, $J(\boldsymbol{x}^*)^{-1}$ existe com $||J(\boldsymbol{x})^*)^{-1}|| \leq \beta$, e $J \in Lip_\gamma(\vartheta(\boldsymbol{x}^*, r))$. Então, existe $\epsilon > 0$ tal que, para todo $x^0 \in \vartheta(x^*, \epsilon)$, a sequência $x^1, x^2, ...$, gerada por

$$x^{k+1} = x^k - J(x^k)^{-1}.F(x^k), \quad para\ k = 0, 1, 2, ...$$

está bem definida, converge para $\boldsymbol{x}^*$ e satisfaz:

$$||x^{k+1} - x^*|| \leq \beta\gamma||x^k - x^*||^2 \quad para\ k = 0, 1, 2, ...$$

(convergência q-quadrática)

2.3 Método de Newton Modificado

No sentido de que o método de Newton (Puro) tem custos significativos quanto ao cálculo das matrizes Jacobianas a cada iteração, a introdução do método de Newton modificado se deu com o objetivo de eliminar esta necessidade, reduzindo assim os custos computacionais envolvidos, porém com a desvantagem do decréscimo da velocidade de convergência.

Consideremos o sistema de equações não lineares (2.2) e a resolução deste através do **Método de Newton Modificado**, conhecido também como **Método de Newton Estacionário**, que consiste em obter a solução do sistema com uma única avaliação da matriz Jacobiana, isto é, somente avaliamos esta no ponto inicial, ao contrário do método de Newton, que avalia a Jacobiana em todas as iterações.

Em resumo o método de Newton Modificado consiste em:

i) Avaliar a Jacobiana no ponto inicial x^0: $J(x^0)$.

ii) Resolver o sistema $J(x^0).(s^0) = -F(x^0)$.

iii) Determinar o novo ponto $x^1 = x^0 + s^0$, e assim sucessivamente, tendo calculado a Jacobiana somente no ponto inicial.

Vamos resolver o sistema de equações que anteriormente fora executado com o método de Newton-Puro (Exemplo 2.3),

Exemplo 2.4 *Resolver o sistema:*

$f_1(x) = ln(x_1^2 + 2x_2^2 + 1) - 0.5 = 0$

$f_2(x) = x_2 - x_1^2 + 0.2 = 0$

(vide figura 2.3)

Similarmente ao método de Newton, resolvemos o sistema $J_0\mathbf{x} = G$, passo a passo, com todas operações envolvidas.

1^a Iteração

$$x^0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$f1 = 0.8863$

$f2 = 0.2000$

$$G = \begin{pmatrix} -0.8863 \\ -0.2000 \end{pmatrix}$$

$$L = \begin{pmatrix} -0.2500 & 1.0000 \\ 1.0000 & 0.0000 \end{pmatrix}$$

$$U = \begin{pmatrix} -2.0000 & 1.0000 \\ 0.0000 & 1.2500 \end{pmatrix}$$

$$y = L \setminus G = \begin{pmatrix} -0.2000 \\ -0.9363 \end{pmatrix}$$

$s = U \setminus y$

$$s^0 = \begin{pmatrix} -0.2745 \\ -0.7490 \end{pmatrix}$$

$x^1 = x^0 + s^0$

$$x^1 = \begin{pmatrix} 0.7255 \\ 0.2510 \end{pmatrix}$$

E assim sucessivamente para as próximas iterações, mantendo a Matriz Jacobiana constante, calculada no ponto x^0, vide resultados a seguir.

Tabela 2.5: 1^a Iteração

item	Valor
$x_1^0 =$	1
$x_2^0 =$	1
f_1	0.8863
f_2	0.2000
$G =$	$\begin{pmatrix} -0.8863 \\ -0.2000 \end{pmatrix}$
$y = L \setminus G$	$\begin{pmatrix} -0.2000 \\ -0.9363 \end{pmatrix}$
$s = U \setminus y$	$\begin{pmatrix} -0.2745 \\ -0.7490 \end{pmatrix}$
$x^1 =$	$\begin{pmatrix} 0.7255 \\ 0.2510 \end{pmatrix}$

Tabela 2.6: 2^a Iteração

item	Valor
$x_1^1 =$	0.7255
$x_2^1 =$	0.2510
f_1	0.0022
f_2	-0.0754
$G =$	$\begin{pmatrix} -0.0022 \\ 0.0754 \end{pmatrix}$
$y = L \setminus G$	$\begin{pmatrix} 0.0754 \\ 0.1674 \end{pmatrix}$
$s = U \setminus y$	$\begin{pmatrix} -0.0310 \\ 0.0133 \end{pmatrix}$
$x^2 =$	$\begin{pmatrix} 0.6944 \\ 0.2643 \end{pmatrix}$

Tabela 2.7: 3^a Iteração

item	Valor
$x_1^2 =$	0.6944
$x_2^2 =$	0.2643
f_1	-0.0163
f_2	-0.0180
$G =$	$\begin{pmatrix} 0.0163 \\ 0.0180 \end{pmatrix}$
$y = L \setminus G$	$\begin{pmatrix} 0.0180 \\ 0.0208 \end{pmatrix}$
$s = U \setminus y$	$\begin{pmatrix} -0.0006 \\ 0.0166 \end{pmatrix}$
$x^3 =$	$\begin{pmatrix} 0.6938 \\ 0.2809 \end{pmatrix}$

Tabela 2.8: 4^a Iteração

item	Valor
$x_1^3 =$	0.6938
$x_2^3 =$	0.2809
f_1	-0.0057
f_2	-0.0004
$G =$	$\begin{pmatrix} 0.0057 \\ 0.0004 \end{pmatrix}$
$y = L \setminus G$	$\begin{pmatrix} 0.0004 \\ 0.0059 \end{pmatrix}$
$s = U \setminus y$	$\begin{pmatrix} 0.0021 \\ 0.0046 \end{pmatrix}$
$x^4 =$	$\begin{pmatrix} 0.6959 \\ 0.2856 \end{pmatrix}$

Tabela 2.9: 5^a Iteração

item	Valor
$x_1^4 =$	0.6959
$x_2^4 =$	0.2856
f_1	-0.0007
f_2	0.0013
$G =$	$\begin{pmatrix} 0.0007 \\ -0.0013 \end{pmatrix}$
$y = L \setminus G$	$\begin{pmatrix} -0.0013 \\ 0.0004 \end{pmatrix}$
$s = U \setminus y$	$\begin{pmatrix} 0.8077.10^{-3} \\ 0.3102.10^{-3} \end{pmatrix}$
$x^5 =$	$\begin{pmatrix} 0.6967 \\ 0.2860 \end{pmatrix}$

Tabela 2.10: 6^a Iteração

item	Valor
$x_1^5 =$	0.6967
$x_2^5 =$	0.2860
f_1	$0.1836.10^{-3}$
f_2	$0.4905.10^{-3}$
$G =$	$\begin{pmatrix} -0.1836.10^{-3} \\ -0.4905.10^{-3} \end{pmatrix}$
$y = L \setminus G$	$\begin{pmatrix} -0.4905.10^{-3} \\ -0.3062.10^{-3} \end{pmatrix}$
$s = U \setminus y$	$\begin{pmatrix} 0.1227.10^{-3} \\ -0.2449.10^{-3} \end{pmatrix}$
$x^6 =$	$\begin{pmatrix} 0.6968 \\ 0.2857 \end{pmatrix}$

Tabela 2.11: 7^a Iteração

item	Valor
$x_1^6 =$	0.6968
$x_2^6 =$	0.2857
f_1	$0.11175.10^{-3}$
f_2	$0.0744.10^{-3}$
$G =$	$\begin{pmatrix} -0.1175.10^{-3} \\ -0.0744.10^{-3} \end{pmatrix}$
$y = L \setminus G$	$\begin{pmatrix} -0.0745.10^{-3} \\ -0.1361.10^{-3} \end{pmatrix}$
$s = U \setminus y$	$\begin{pmatrix} -0.0172.10^{-3} \\ -0.1089.10^{-3} \end{pmatrix}$
$x^7 =$	$\begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$

Tabela 2.12: 8^a Iteração

item	Valor
$x_1^7 =$	0.6968
$x_2^7 =$	0.2856
f_1	$0.2748.10^{-4}$
f_2	$-0.1044.10^{-4}$
$G =$	$\begin{pmatrix} -0.2748.10^{-4} \\ 0.1044.10^{-4} \end{pmatrix}$
$y = L \setminus G$	$\begin{pmatrix} 0.1044.10^{-4} \\ -0.2487.10^{-3} \end{pmatrix}$
$s = U \setminus y$	$\begin{pmatrix} -0.1517.10^{-4} \\ -0.1990.10^{-4} \end{pmatrix}$
$x^8 =$	$\begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$

Tabela 2.13: 9^a Iteração

item	Valor
$x_1^8 =$	0.6968
$x_2^8 =$	0.2856
f_1	$0.0873.10^{-5}$
f_2	$-0.9196.10^{-5}$
$G =$	$\begin{pmatrix} -0.0873.10^{-5} \\ 0.9196.10^{-5} \end{pmatrix}$
$y = L \setminus G$	$\begin{pmatrix} 0.9196.10^{-5} \\ 0.1426.10^{-5} \end{pmatrix}$
$s = U \setminus y$	$\begin{pmatrix} -0.4028.10^{-5} \\ 0.1141.10^{-5} \end{pmatrix}$
$x^9 =$	$\begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$

Tabela 2.14: 10^a Iteração

item	Valor
$x_1^9 =$	0.6968
$x_2^9 =$	0.2856
f_1	$-0.1741.10^{-5}$
f_2	$-0.2442.10^{-5}$
$G =$	$\begin{pmatrix} 0.1721.10^{-5} \\ 0.2442.10^{-5} \end{pmatrix}$
$y = L \setminus G$	$\begin{pmatrix} 0.2442.10^{-5} \\ 0.2352.10^{-5} \end{pmatrix}$
$s = U \setminus y$	$\begin{pmatrix} -0.0280.10^{-5} \\ 0.1881.10^{-5} \end{pmatrix}$
$x^{10} =$	$\begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$

Critério de parada satisfeito $||F|| = 0.6746.10^{-6}$

Resolução do sistema utilizando o programa **"Newtonmodificado"**, programa este que executa todo processo iterativo utilizando o cálculo da Jacobiana sómente no ponto inicial. (vide anexo D). Resolução:

$[x, k]$=newtonmodificado $(1.d-6)$

entre com a aproximação inicial [1; 1]

$$x^0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad x^1 = \begin{pmatrix} 0.7255 \\ 0.2510 \end{pmatrix} \qquad x^2 = \begin{pmatrix} 0.6945 \\ 0.2643 \end{pmatrix}$$

$$x^3 = \begin{pmatrix} 0.6938 \\ 0.2809 \end{pmatrix} \qquad x^4 = \begin{pmatrix} 0.6960 \\ 0.2856 \end{pmatrix} \qquad x^5 = \begin{pmatrix} 0.6967 \\ 0.2860 \end{pmatrix}$$

$$x^6 = \begin{pmatrix} 0.6969 \\ 0.2857 \end{pmatrix} \qquad x^7 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix} \qquad x^8 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$$

$$x^9 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix} \qquad x^{10} = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$$

$k = 10$ iterações.

Com este programa do Matlab, obtivemos o valor $x^{10} = [0.6968, 0.2856]$ após 10 iterações, com critério de parada satisfeito $||F|| = 0.6746.10^{-6}$.

Dos resultados obtidos, comparando-os com os do método de Newton, observa-se que o método de Newton modificado é muito mais lento, sendo que no primeiro caso foram necessárias 4 iterações, enquanto que no Newton Modificado foram necessárias 10 iterações.

Em relação à convergência deste método: verificamos que, sob as mesmas hipóteses do teorema 2.1, a seqüência gerada pelo método de Newton Modificado converge q-linearmente a x^*, ou seja, $\exists \;\; K_c > 0$, tal que:

$$||x^{k+1} - x^*|| \leq K_c||x^0 - x^*||||x^k - x^*||.$$

A demonstração pode ser encontrada em [9].

2.4 Método de Newton Discreto

Em muitas situações, é difícil e/ou computacionalmente muito caro obter a Jacobiana da função F, ainda que seja em um único ponto, como é o caso que ocorre no método de Newton modificado. Podemos então lançar mão de aproximações para as derivadas parciais de F, substituindo a Jacobiana analítica J(x) por uma aproximação obtida a partir de diferenças finitas.

Se f é uma função de uma variável, podemos aproximar a derivada $f'(x)$ por:

$$f'(x) \simeq (f(x+h) - f(x))/h,$$

onde h é um valor positivo, próximo de zero. Esta aproximação é conhecida como diferença avançada, e podemos pensar que o método das secantes, estudado no Capítulo I, tem uma aproximação desse tipo como base.

Outras aproximações para a derivada de uma função em R, bem como a análise dos erros cometidos nestas aproximações, podem ser encontradas em [2].

No caso de várias variáveis, objeto deste capítulo, podemos usar a mesma idéia para aproximar a componente (i, j) da Jacobiana J(x) por:

$$J_{i,j}(x) \simeq (f_i(x + he_j) - f_i(x))/h \tag{2.4}$$

onde e_j denota o j-ésimo vetor da base canônica de $\Re^n$. A versão do método de Newton em que a Jacobiana em cada iteração k, $J(x^k)$, é aproximada, usando-se a expressão (2.4) calculada em x^k, é conhecida como método de Newton Discreto.

Se o passo h da discretização é escolhido apropriadamente, podemos até mesmo preservar a convergência quadrática do método de Newton. O seguinte teorema resume as propriedades de convegência do método de Newton Discreto:

Teorema 2.2 *Suponha que F e x^* satisfazem as mesmas hipóteses do Teorema 2.1. Então existem ϵ, $h > 0$ tais que, se $\{h_k\}$ é uma seqüência real com $0 < |h_k| \leq h$ e $x^0 \in N(x^*, \epsilon)$, a seqüência $\{x^k\}$ gerada por:*

$(A_k)_j = (F(x^k + h_k e_j) - F(x^k))/h_k, \quad j = 1, ..., n,$

$x^{k+1} = x^k - A_k^{-1} F(x^k), \quad$ *k=0,1,2,.....*

está bem definida e converge q-linearmente para x^*.

Se $\lim_{k\to 0} h_k = 0$, *então a convergência é q-superlinear.*
Se existe alguma constante c_1 *tal que:*

$$|h_k| \leq c_1||x^k - x^*||,$$

ou equivalentemente uma constante c_2 *tal que:*

$$|h_k| \leq c_2||F(x^k)||,$$

então a convergência é q-quadrática.

A demonstração pode ser encontrada em [4].

Capítulo 3

Métodos Quase-Newton para equações não lineares

3.1 Introdução

A partir da idéia de que, para a resolução de sistemas não lineares como (2.2), o método de Newton é bom, porém com alto custo computacional, parece natural a introdução de métodos "quase bons", porém de custo relativamente baixo.

A maioria dos métodos quase-Newton foi estabelecida com este objetivo.

Para ser (quase) tão bom quanto o método de Newton, estes métodos devem ser parecidos com o de Newton sob vários pontos de vista.

O formato dos métodos quase-Newton é:

$$B^k.s^k = -F(x^k)$$
$$x^{k+1} = x^k + s^k$$

onde B_k é uma substituta da matriz Jacobiana $J(x^k)$.

A redução de custos operacionais tem, como contrapartida, a redução na velocidade de convergência. O método quase-Newton mais simples é o chamado de Newton Modificado (Newton Estacionário), visto anteriormente, onde fixamos $B_k = J(x^0)$.

3.2 Método de Shamanski

O método de Newton modificado tem como objetivo reduzir o custo computacional por iteração, em geral, gera uma sequência de pontos $\{x^k\}$ que converge bem mais lentamente à solução do que quando aplicamos o método de Newton. Buscando um compromisso, entre menor custo computacional e maior velocidade de convegência, surge o método de Newton estacionário com recomeços a cada **m** iterações, sendo **m** inteiro e constante para todo o processo, também chamado de método de **Shamanski**; ou seja, fixado um inteiro **m**, fixamos a matriz substituta da Jacobiana a cada **m** iterações.

Existem estudos para encontrar **m** ótimo, com relação à grande eficiência e baixo custo. Mas estes resultados referem-se a casos de problemas específicos; vide literatura especializada [11].

Concluída uma aproximação x^k e definido um inteiro m de iterações sucessivas com ${B_j}^k = J(x^k)$, para $j = 1, ..., m-1$, podemos descrever a transição de x^k para x^{k+1}, por:

$$y^1 = x^k - J(x^k)^{-1}F(x^k)$$
$$y_{j+1} = y_j - J(x^k)^{-1}.F(y^j), \ \ para \ \ 1 \leq j \leq m-1$$
$$x^{k+1} = y_m$$

Observar que para m=1, temos o método de Newton (Puro), e para $m = \infty$, o método de Newton Modificado. Sob as mesmas hipóteses do teorema 2.1, do capítulo anterior pode-se mostrar que a seqüência $\{x^k\}$ gerada pelo método de Shamanski converge q-superlinearmente a x^*.

Tomemos novamente:

$$\begin{cases} f_1(\mathbf{x}) = ln(x_1^2 + 2x_2^2 + 1) - 0.5 \\ f_2(\mathbf{x}) = x_2 - x_1^2 + 0.2 \end{cases}$$

Para aplicar o método de Shamanski, passamos a resolver o sistema:

$$\begin{cases} f_1(x) = 0 \\ f_2(x) = 0 \end{cases}$$

Como pelo método de Newton Modificado foram necessárias 10 iterações, escolhemos, para efeito de exemplo, $m = 3$.

Usamos como critério de parada $||F(x^k)|| < tol$, com $tol = 10^{-6}$.

Tabela 3.1: 1^a Iteração: - Calculamos $J(x^0)$

item	Valor
$x_1^0 =$	1.0
$x_2^0 =$	1.0
f_1	0.8863
f_2	0.2000
G	$\begin{pmatrix} -0.8863 \\ -0.2000 \end{pmatrix}$
J	$\begin{pmatrix} 0.5000 & 1.000 \\ -2.000 & 1.000 \end{pmatrix}$
$s = J \setminus G$	$\begin{pmatrix} -0.2745 \\ -0.7490 \end{pmatrix}$
x^1	$\begin{pmatrix} 0.7255 \\ 0.2510 \end{pmatrix}$

$||F(x^1)|| = 0.07535998 > tol$

Tabela 3.2: 2^a Iteração - Mantida $J(x^0)$

item	Valor
$x_1^1 =$	0.7255
$x_2^1 =$	0.2510
f_1	0.0021
f_2	0.0753
G	$\begin{pmatrix} -0.0021 \\ -0.0753 \end{pmatrix}$
J	$\begin{pmatrix} 0.5000 & 1.0000 \\ -2.0000 & 1.0000 \end{pmatrix}$
$s = J \setminus G$	$\begin{pmatrix} -0.0031 \\ -0.0133 \end{pmatrix}$
x^2	$\begin{pmatrix} 0.6944 \\ 0.2643 \end{pmatrix}$

$||F(x^2)|| = 0.886229436 > tol$

Tabela 3.3: 3^a Iteração - Mantida $J(x^0)$

item	Valor
$x_1^2 =$	0.6944
$x_2^2 =$	0.2643
f_1	-0.0163
f_2	-0.0179
G	$\begin{pmatrix} 0.0163 \\ 0.0179 \end{pmatrix}$
J	$\begin{pmatrix} 0.5000 & 1.0000 \\ -2.0000 & 1.0000 \end{pmatrix}$
$s = J \setminus G$	$\begin{pmatrix} -0.0006 \\ 0.0166 \end{pmatrix}$
x^3	$\begin{pmatrix} 0.6938 \\ 0.2809 \end{pmatrix}$

$||F(x^3)|| = 0.00581116 > tol$

Tabela 3.4: 4^a Iteração: - Calculamos $J(x^3)$

item	Valor
$x_1^3 =$	0.6938
$x_2^3 =$	0.2809
f_1	-0.0057
f_2	-0.0004
G	$\begin{pmatrix} 0.0057 \\ 0.0004 \end{pmatrix}$
J	$\begin{pmatrix} 0.8464 & 0.6856 \\ -1.3876 & 1.0000 \end{pmatrix}$
$s = J \setminus G$	$\begin{pmatrix} 0.0030 \\ 0.0046 \end{pmatrix}$
x^4	$\begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$

$||F(x^4)|| = 0.00575174 > tol$

Tabela 3.5: 5^a Iteração: - Mantida $J(x^3)$

item	Valor
$x_1^4 =$	0.6968
$x_2^4 =$	0.2856
f_1	0.000015
f_2	-0.000009
G	$\begin{pmatrix} -0.000015 \\ 0.000009 \end{pmatrix}$
J	$\begin{pmatrix} 0.8464 & 0.6856 \\ -1,387 & 1.0000 \end{pmatrix}$
$s = J \setminus G$	$\begin{pmatrix} -0.1195 \\ 0.0732 \end{pmatrix}$
x^5	$\begin{pmatrix} 0.6968 \\ 0.2855 \end{pmatrix}$

$||F(x^5)|| = 0.000015143 > tol$

Tabela 3.6: 6^a Iteração: - Mantida $J(x^3)$

item	Valor
$x_1^5 =$	0.6968
$x_2^5 =$	0.2855
f_1	-0.00000004
f_2	0.00000006
G	$\begin{pmatrix} -0.00000004 \\ -0.00000006 \end{pmatrix}$
J	$\begin{pmatrix} 0.8464 & 0.6856 \\ -1.3876 & 1.0000 \end{pmatrix}$
$s = J \setminus G$	$\begin{pmatrix} 0.00000005 \\ 0.00000002 \end{pmatrix}$
x^6	$\begin{pmatrix} 0.6968 \\ 0.2855 \end{pmatrix}$

$||F(x^6)|| = 0.000000069 < tol$ - critério de parada satisfeito

Utilizamos a rotina **nsol.m** do MatLab, escrita por C.T. Kelley [9], novembro de 1993, para Newton Discreto, Shamanski e Newton Modificado, conforme os parâmetros [maxit, isham, rsham] (para detalhes vide Anexo A), sendo para o Método de Newton Discreto: maxit = número máximo de iterações, isham = 1 e rsham = 0; para Shamanski: maxit = número máximo de iterações, isham = 3 (no exemplo) e rsham = 1; e para Newton modificado (Chord): maxit = número máximo de iterações, isham = -1 e rsham = 1. Em todos os casos, a matriz Jacobiana é aproximada por diferenças finitas.

Considerando novamente o exemplo 2.3, vamos mostrar, a seguir, os resultados obtidos com **nsol.m** executando os métodos de Shamanski e Newton Discreto. Mostramos também resultados obtidos com o método de Newton Puro implementado no programa citado no capítulo 2, seção 2.2.

Shamanski

function [f] = fcarlos(x)

$f_1(x) = ln(x_1^2 + 2 * x_2^2 + 1) - .5$

$f_2(x) = x_2 - x_1^2 + .2$

f=f';

» clear all

Entre com os parâmetos iniciais: [40,3,1]

Entre com o erro aceitável: [1.d-6,1.d-6]

$\mathbf{x}^0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ $\mathbf{x}^1 = \begin{pmatrix} 0,7255 \\ 0,2510 \end{pmatrix}$ $\mathbf{x}^2 = \begin{pmatrix} 0,6945 \\ 0,2643 \end{pmatrix}$ $\mathbf{x}^3 = \begin{pmatrix} 0,6938 \\ 0,2809 \end{pmatrix}$ $\mathbf{x}^4 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$

$\mathbf{x}^5 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$

Com o programa Matlab rotina nsol, obtivemos o valor $\mathbf{x}^5 = [0.6968, 0.2856]$ após 5 iterações, com critério de parada satisfeito $||f(\mathbf{x}^5)|| = 7.27010^{-8}$.

Newton Discreto

function [f] = fcarlos(x)

$f_1(x) = ln(x_1^2 + 2 * x_2^2 + 1) - .5$

$f_2(x) = x_2 - x_1^2 + .2$

f=f'

» clear all

Entre com os parâmetos iniciais: [40,1,0]

Entre com o erro aceitável: [1.d-6,1.d-6]

$\mathbf{x}^0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ $\mathbf{x}^1 = \begin{pmatrix} 0.7255 \\ 0.2510 \end{pmatrix}$ $\mathbf{x}^2 = \begin{pmatrix} 0.6982 \\ 0.2867 \end{pmatrix}$ $\mathbf{x}^3 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$ $\mathbf{x}^4 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$

Com o programa Matlab rotina nsol, obtivemos o valor $\mathbf{x}^4 = [0.6968, 0.2856]$ após 4 iterações, com critério de parada satisfeito $||f(\mathbf{x}^4)|| = 1.60310^{-12}$.

.

Newton-Puro

» clear all

fcarlos Newton-Puro

» [x,k]=newton(1.d-6,1.d-6)

entre com a aproximação inicial [1,1]

$\mathbf{x}^0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ $\mathbf{x}^1 = \begin{pmatrix} 0.7255 \\ 0.2510 \end{pmatrix}$ $\mathbf{x}^2 = \begin{pmatrix} 0.6982 \\ 0.2867 \end{pmatrix}$ $\mathbf{x}^3 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$ $\mathbf{x}^4 = \begin{pmatrix} 0.6968 \\ 0.2856 \end{pmatrix}$

Com o programa Matlab rotina Newton-Puro, obtivemos o valor $\mathbf{x}^4 = [0.6968, 0.2856]$ após 4 iterações, com critério de parada satisfeito $||f(\mathbf{x}^4)|| = 1.51110^{-12}$.

Alguns exemplos da literatura foram testados usando o método de Newton-Puro e a rotina **nsol** para os métodos de Newton Discreto e de Shamanski. Os resultados obtidos estão listados na tabela 3.7:

Problema 3.1 *fdennys1:*

$$\begin{cases} f_1(x) = x_1^2 + x_2^2 - 2 \\ f_2(x) = e^{x_1 - 1} + x_2^3 - 2 \end{cases}$$

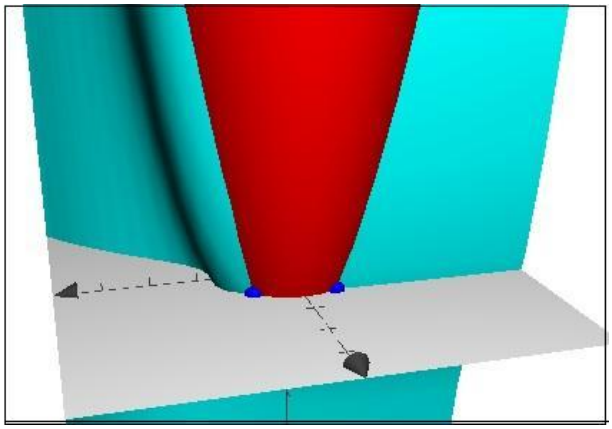


Figura 3.1: Gráfico 3D - função dennys1

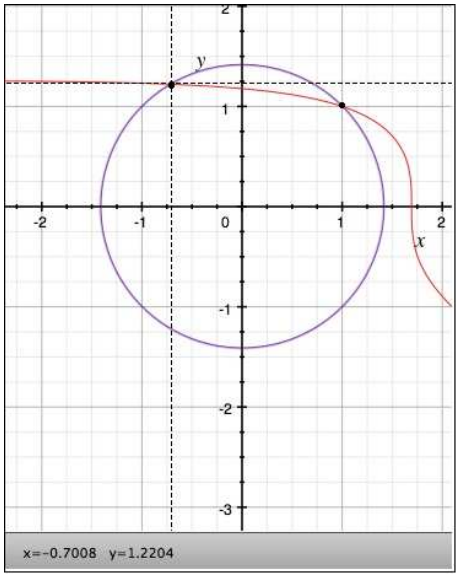


Figura 3.2: Pontos de interseção função dennys1

Problema 3.2 *fdennys2:*

$$\begin{cases} f_1(x) = x_1 + x_2 - 3 \\ f_2(x) = x_1^2 - x_2^2 - 9 \end{cases}$$

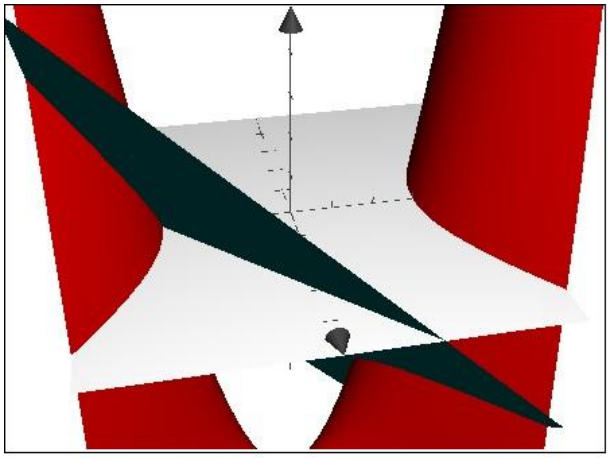


Figura 3.3: Gráfico 3D - função dennys2

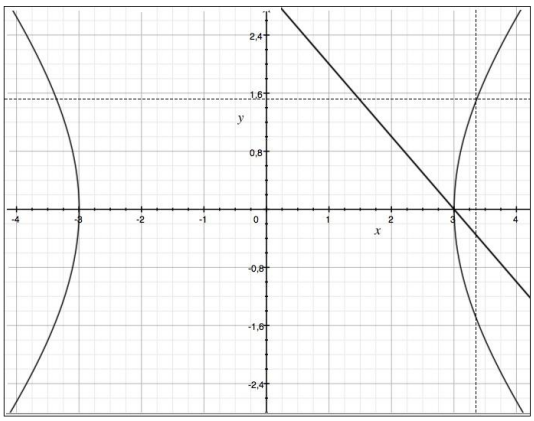


Figura 3.4: Ponto de intersecção função dennys2

Problema 3.3 *fconte1:*

$$\begin{cases} f_1(x) = x_1 + e^{x_1 - 1} + (x_2 + 3)^2 - 27 \\ f_2(x) = \dfrac{e^{x_2 - 2}}{x_1} + x_3^2 - 10 \\ f_3(x) = x_3 + sin(x_2 - 2) + x_2^2 - 7 \end{cases}$$

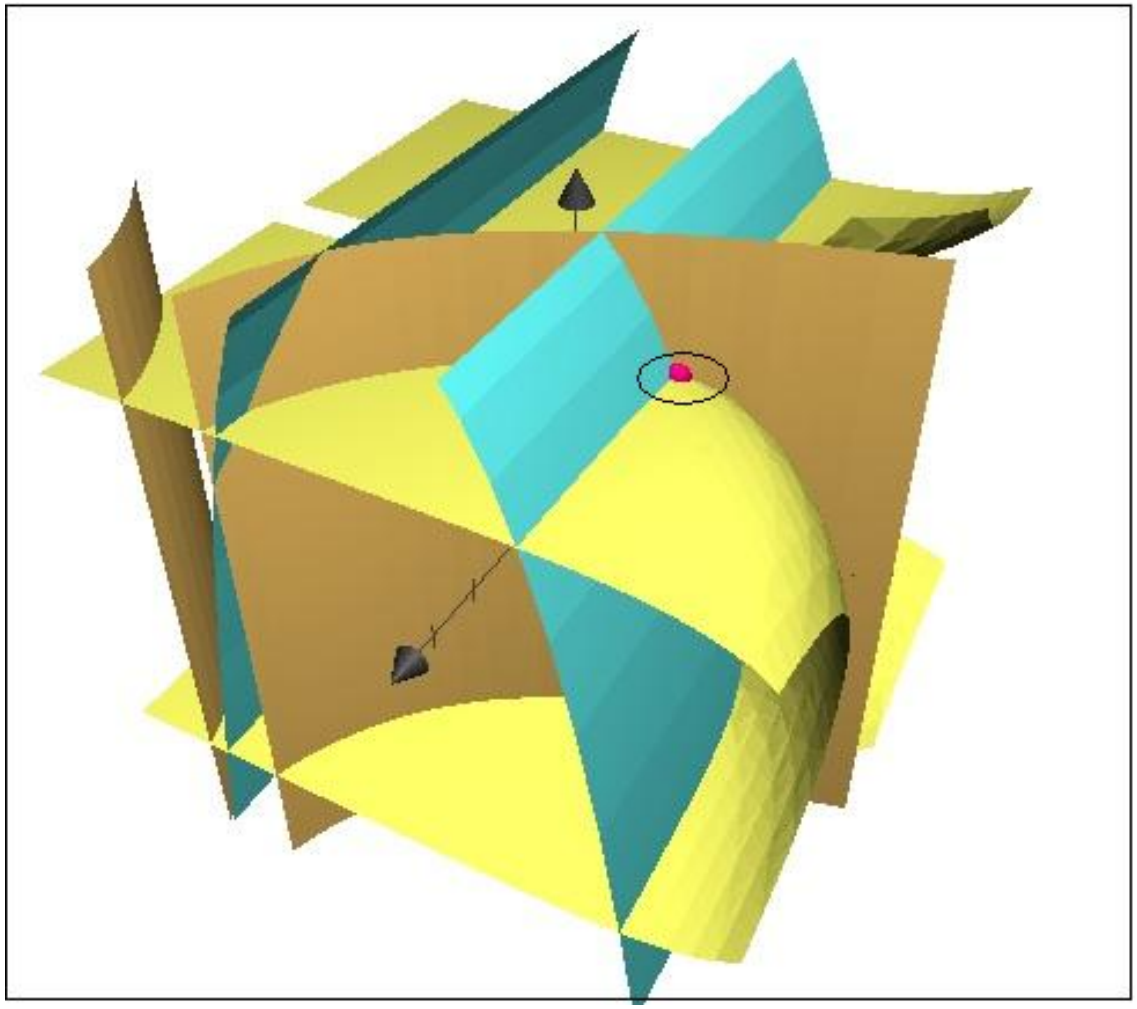


Figura 3.5: Gráfico 3D e ponto de intersecção função conte1

Problema 3.4 *fconte2:*

$$\begin{cases} f_1(x) = x_1 + log_{10}(x_1) - x_2^2 \\ f_2(x) = 2x_1^2 - x_1.x_2 - 5x_1 + 1 \end{cases}$$

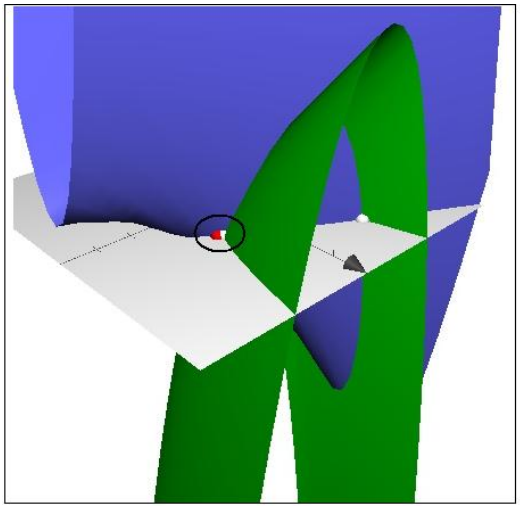


Figura 3.6: Gráfico 3D - função conte2

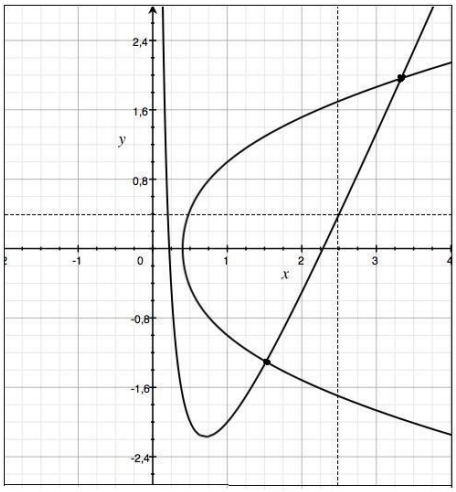


Figura 3.7: Pontos de intersecção função conte2

Problema 3.5 *ffausett:*

$$\begin{cases} f_1(x) = x_1^2 + x_2^2 + x_3^2 - 1 \\ f_2(x) = x_1^2 + x_3^2 - 0.25 \\ f_3(x) = x_1^2 + x_2^2 + 4x_3 \end{cases}$$

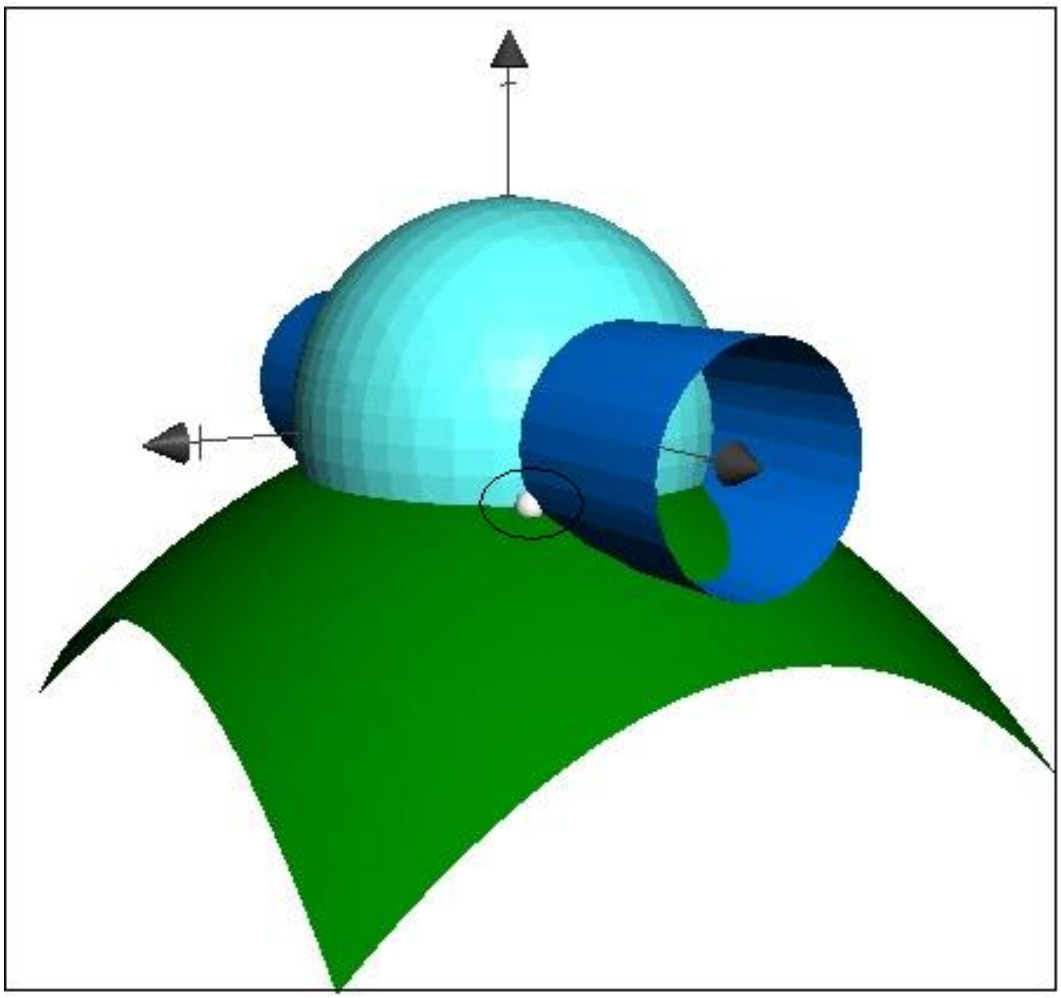


Figura 3.8: Gráfico 3D - função fausett

A tabela 3.7 é composta de seis colunas indicando: na 1^a a função estudada, na 2^a a rotina aplicada, na 3^a o número de iterações (K), na 4^a o tempo computacional medido em (ms). Nas colunas 5^a e 6^a a norma de F e de s ($s = ||\mathbf{x}^{k+1} - \mathbf{x}^k||_\infty$), critérios de parada. Para o método de Shamanski foram utilizados os parâmetros $m = 5, m = 10, m = 20$

Tabela 3.7: Valores computados

Função	Rotina	K	tempo(ms)	$\|\|F\|\|$	$\|\|s\|\|$
fedennys1	Newton-Puro	5	41,289	$0,6342.10^{-6}$	$0,4523.10^{-5}$
	Newton Discreto	5	127,408	$0,2426.10^{-6}$	-
	Shamanski (m=5)	11	118,075	$0,2020.10^{-6}$	-
	Shamanski (m=10)	14	30,606	$0,3163.10^{-7}$	-
	Shamanski (m=20)	22	104,585	$0,5543.10^{-8}$	-
fdennys2	Newton-Puro	-	-	diverge	-
	Newton Discreto	2	118,319	$0,7742.10^{-7}$	-
	Shamanski (m=5)	6	224,749	$0,6662.10^{-7}$	-
	Shamanski (m=10)	11	26,114	$0,2038.10^{-8}$	-
	Shamanski (m=20)	20	79,822	$0,2002.10^{-7}$	-
fconte1	Newton-Puro	6	48,989	$0,2014.10^{-9}$	$0,1227.10^{-5}$
	Newton Discreto	6	50,732	$0,2002.10^{-7}$	-
	Shamanski (m=5	12	90,651	$0,2150.10^{--6}$	-
	Shamanski (m=10)	19	15,037	$0,2120.10^{-6}$	-
	Shamanski (m=20)	25	121,574	$0,3866.10^{-6}$	-
fconte2	Newton-Puro	4	42,896	$0,4121.10^{-12}$	$0,5761.10^{-7}$
	Newton Discreto	4	91,552	$0,3972.10^{-8}$	-
	Shamanski (m=5)	7	122,429	$0,1151.10^{-7}$	-
	Shamanski (m=10)	11	28,883	$0,3442.10^{-8}$	-
	Shamanski (m=20)	14	103,115	$0,1354.10^{-6}$	-
ffausett	Newton-Puro	4	44,809	$0,6321.10^{-7}$	$7,95.10^{-4}$
	Newton Discreto	4	23,129	$0,6321.10^{-6}$	-
	Shamanski (m=5)	8	111,302	$0,1222.10^{-6}$	-
	Shamanski (m=10)	11	8,498	$0,4593.10^{-6}$	-
	Shamanski (m=20)	20	67,822	$0,1786.10^{-6}$	-

Para procedermos um comparativo entre os métodos com relação aos tempos computacionais, tomamos como referência o tempo do Newton-Puro, atribuindo o valor de 100% a este tempo e relacionando-o com os demais. Tomamos também o método de Shamanski com parâmetro m=10, pois foi com o qual a convergência teve um melhor

desempenho computacional, ou seja tanto em relação ao número de atualizações da Jacobiana, bem como em relação ao tempo computacional.

Tabela 3.8: Comparativo em porcentagem

Rotina	fdennys1	fdennys2	fconte1	fconte2	ffausett
Newton Puro	100	div.	100	100	100
Newton Discreto	309	100	103	213	52
Shamanski(m=10)	74	22	30	67	19

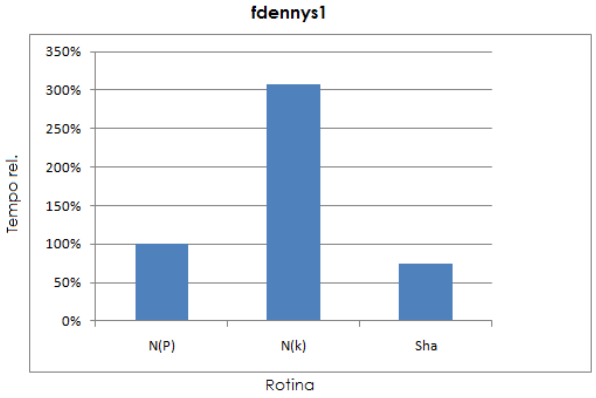
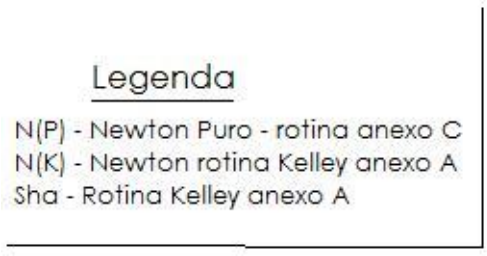


Figura 3.9: fdennys1



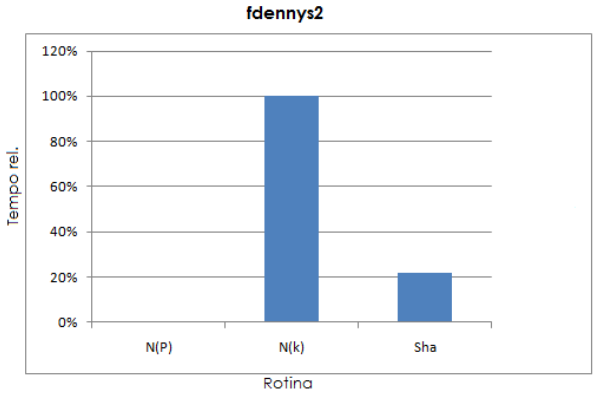


Figura 3.10: fdennys2

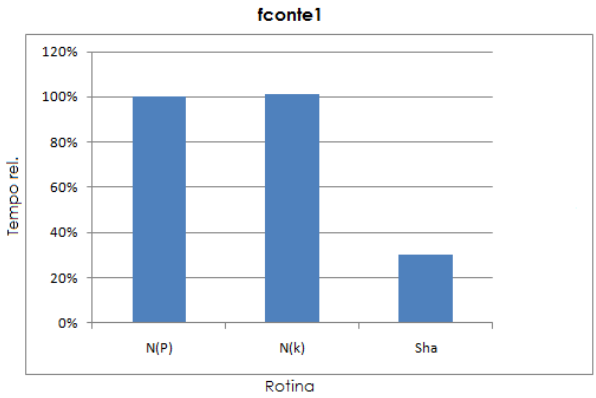


Figura 3.11: fconte1

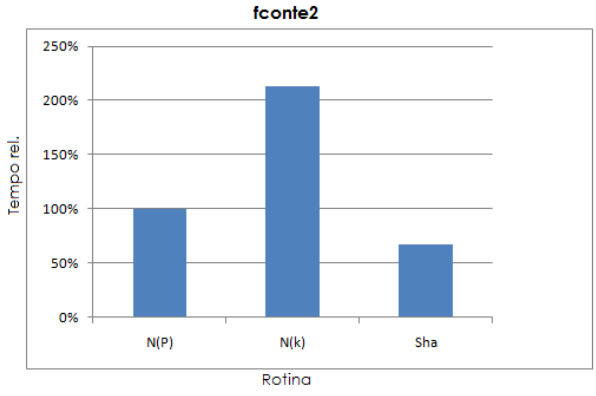


Figura 3.12: fconte2

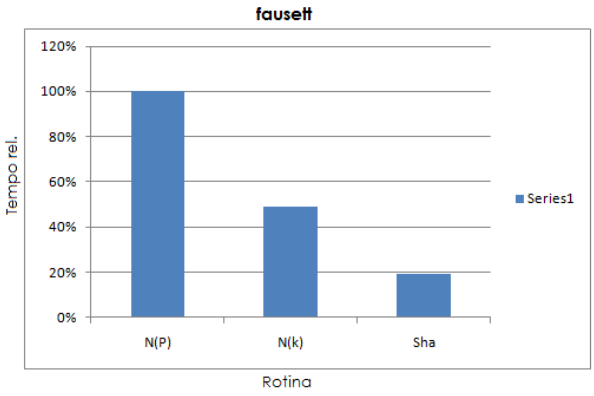


Figura 3.13: fausett

Dos resultados obtidos, podemos concluir que:

Houve um único caso em que não houve convergência, que foi no método de Newton-Puro, para a função fdennys2, enquanto que os métodos de Newton-Puro e discreto sempre realizam menos iterações que o método de Shamanski, qualquer que seja o parâmetro **m** usamos.

Newton-Puro e discreto não diferem no número de iterações, o que mostra que aproximar a Jacobiana por diferenças finitas não altera significativamente a velocidade de convergência no método de Newton.

Em relação aos valores de **m** usados no método de Shamanski, observa-se que são necessárias mais iterações à medida que o valor do parâmetro aumenta, conforme era esperado. No entanto, o tempo é menor para **m=10**, revelando um melhor compromisso entre o número total de iterações realizadas e de atualizações da Jacobiana.

Utilizamos anteriormente os diversos métodos de forma a analisar o comportamento dos mesmos aplicados a sistemas de pequenas dimensões (max = 3). Agora passamos a analisar o comportamento destes métodos para sistemas de equações não lineares de grande porte, e para tanto, vamos aplicá-los ao sistema tridiagonal de Broyden [7].

Broyden tridiagonal

$$\begin{cases} f_1(x) = (3 - 2x_1)x_1 - 2x_2 + 1 = 0 \\ f_i(x) = (3 - 2x_i)x_i - x_{i-1} - 2x_{i+1} + 1 = 0 \\ f_n(x) = (3 - 2x_n)x_n - x_{n-1} + 1 = 0 \end{cases} \tag{3.1}$$

para $i = 2, ..., (n-1)$ e $x^0 = (-1, -1, ..., -1)^T$

A seguir são listados, na tabela 3.9, os resultados obtidos, onde, para efeito comparativo com o método de Shamanski escolhemos o parâmetro $\mathbf{m = 10}$, pois como comentado anteriormente é o que demanda o menor tempo computacional, mesmo que com um número de iterações intermediário entre os parâmetros $\mathbf{m = 5}$ **e** $\mathbf{m = 20}$.
A notação utilizada na tabela 3.9 é a mesma utilizada da tabela 3.7.

Observamos que, em termos do número de iterações realizadas, o comportamento dos métodos foi semelhante ao descrito na tabela 3.7. Newton-Puro e discreto sempre realizam menos iterações que Shamanski.

Quanto ao tempo computacional, observa-se que o do Newton-Puro sempre é inferior ao de Newton rotina Kelley (nsol) para Newton e Shamanski, isto motivado pelo fato do maior tempo envolvido na atualização da Jacobiana pelo processo discreto, tendo como menor o de Shamanski pela menor quantidade de atualizações em relação ao Newton.

Os gráficos comparativos dos tempos relativos envolvidos, são similares aos das funções de pequenas dimensões.

Tabela 3.9: Valores computados

n	Rotina	K	tempo(ms)	$\|\|F\|\|$	$\|\|s\|\|$
5	Newton-Puro	4	19,443	$0,8156.10^{-11}$	$0,2019.10^{-6}$
	Newton Discreto	4	91,421	$0,8102.10^{-1}$	-
	Shamanski(m=10)	11	95,484	$0,6001.10^{-11}$	-
10	Newton-Puro	4	41,520	$0,7548.10^{-11}$	$0,1942.10^{-6}$
	Newton Discreto	4	65,288	$0,7467.10^{-11}$	-
	Shamanski(m=10)	11	88,400	$0,5700.10^{-11}$	-
20	Newton-Puro	4	34,025	$0,7548.10^{-11}$	$0,1942.10^{-6}$
	Newton Discreto	4	104,248	$0,7429.10^{-11}$	-
	Shamanski(m=10)	11	103,500	$0,5666.10^{-11}$	-
50	Newton-Puro	4	38,354	$0,7548.10^{-11}$	$0,1942.10^{-6}$
	Newton Discreto	4	351,331	$0,7355.10^{-11}$	-
	Shamanski(m=10)	11	83,133	$0,5602.10^{-6}$	-
100	Newton-Puro	4	40,265	$0,7548.10^{-11}$	$0,1942.10^{-6}$
	Newton Discreto	4	143,982	$0,7273.10^{-11}$	-
	Shamanski(m=10)	11	155,683	$0,5531.10^{-11}$	-
200	Newton-Puro	4	38,975	$0,7548.10^{-11}$	$0,1942.10^{-6}$
	Newton Discreto	4	539,297	$0,7157.10^{-11}$	-
	Shamanski(m=10)	11	405,368	$0,5431.10^{-11}$	-
500	Newton-Puro	4	102,349	$0,7548.10^{-11}$	$0,1942.10^{-6}$
	Newton Discreto	4	3856,776	$0,6929.10^{-11}$	-
	Shamanski(m=10)	11	2028,253	$0,5233.10^{-11}$	-
1000	Newton-Puro	4	229,592	$0,7548.10^{-11}$	$0,1942.10^{-6}$
	Newton Discreto	4	28903,875	$0,6672.10^{-11}$	-
	Shamanski(m=10)	11	15007,386	$0,5010.10^{-11}$	-

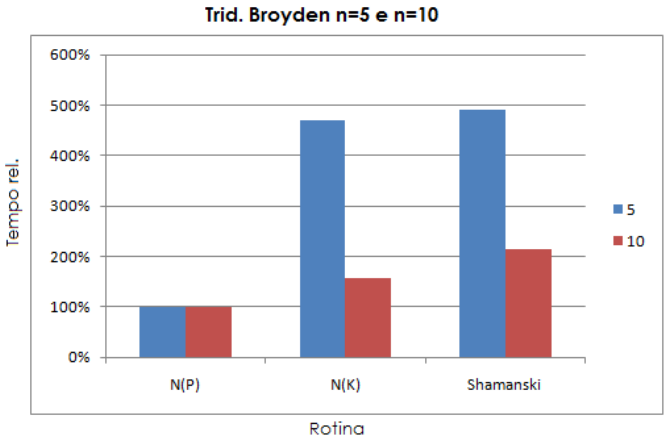


Figura 3.14: Broyden Tridiagonal: n=5 e n=10

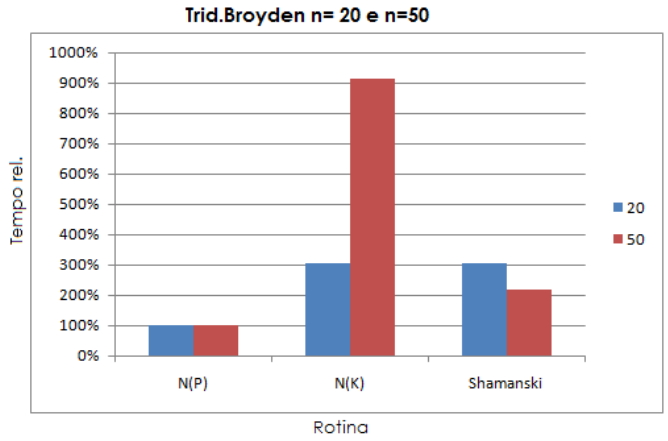


Figura 3.15: Broyden Tridiagonal: n=20 e n=50

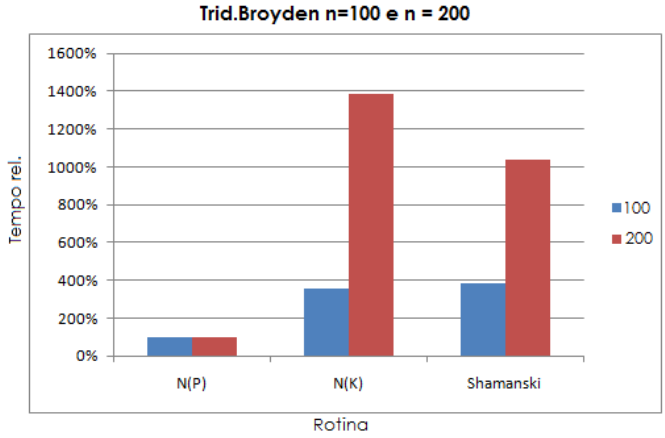


Figura 3.16: Broyden Tridiagonal: n=100 e n=200

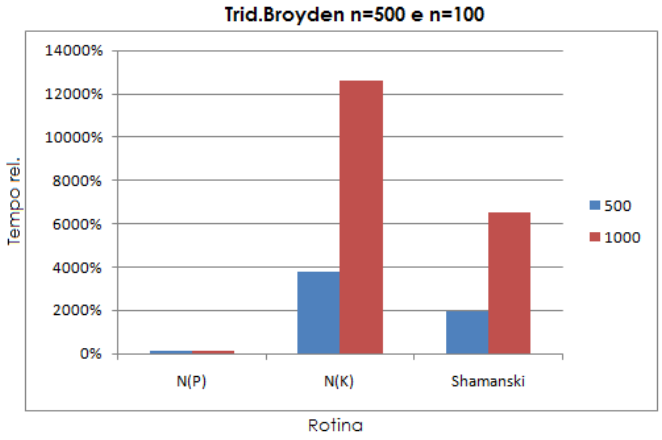


Figura 3.17: Broyden Tridiagonal: n=500 e n=1000

3.3 Método de Broyden

Embasado no fato que o custo por iteração do método de Newton para problemas densos, com n variáveis, é substancialmente caro, isto é:

- O cálculo da matriz Jacobiana tem custo computacional da ordem de n^2 operações;

- A resolução do sistema linear tem custo computacional da ordem de (n^3) operações, surge uma outra família de métodos obedecendo a filosofia quase-Newton, que é a dos métodos secantes.

Assim como o método de Newton é a generalização para sistemas do algoritmo para determinação dos zeros de funções, os métodos secantes são as generalizações do método das secantes para o problema multidimensional.

Da mesma forma que antes, na iteração k a função $F(x)$ é aproximada por :

$L^k(x) = F(x^k) + B_k(x - x^k)$.

A equação secante vem da aproximação da derivada,

Em $\Re : f'(x_k) \simeq \dfrac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$

Em $\Re^n : B_k(x^k - x^{k-1}) \simeq F(x^k) - F(x^{k-1})$

e existem infinitas matrizes que satisfazem a última equação.

O método proposto por Broyden em 1965 [4], [9] é um dos mais conhecidos entre os métodos secantes, ou seja, aqueles métodos em que a matriz Jacobiana $J(x^k)$ é aproximada por uma matriz B_k que satifaz a equação secante. A fórmula para B_{k+1} consiste numa correção de posto um sobre a matriz B_k, e assim temos:

Fórmula de Broyden.

$$B_{k+1} = B_k + \frac{(y^k - B_k s^k)(s^k)^T}{(s^k)^T(s^k)} \tag{3.2}$$

onde, $y^k = F(x^{k+1}) - F(x^k) \quad \text{e} \quad s^k = x^{k+1} - x^k$.

O objetivo é se evitar a avaliação da matriz Jacobiana, a cada iteração, mas a resolução do sistema linear é também simplificada pois, o fato de B_{k+1} ser uma correção de posto um sobre B_k, permite o cálculo de $(B_{k+1})^{-1}$ com $\mathrm{O}(n^2)$ operações através da fórmula de Sherman-Morrison [4]. O uso da fórmula (3.2) não é adequado para o caso

esparso pois, ainda que B_k tenha o mesmo padrão de esparsidade de $J(x^k)$, B_{k+1} pode não resultar esparsa.

Neste caso, em geral se dá preferência pela aproximação de $J(x^k)^{-1}$. Com o uso da fórmula de Sherman-Morrison [4], [9], obtemos a atualização para a inversa de B_{k+1} por:

$$B_{k+1}^{-1} = B_k^{-1} + \frac{(s_k - B_k^{-1}y_k)s_k^T B_k^{-1}}{s_k^T B_k^{-1} y_k} \tag{3.3}$$

Problema 3.6 *fdennys1*

$$\text{F(x)} = \begin{cases} x_1^2 + x_2^2 - 2 \\ exp(x_1 - 1) + x_2^3 - 2 \end{cases}$$

Neste caso ao aplicar o método de Broyden (brsol) - rotina Kelley [9], com os parâmetos $\mathbf{x}^0 = [1, 5]$, e $[40, 39, 0]$, (vide Anexo E), a sequência de pontos (vetores) que é gerada pelo método não convergiu.

Procedemos a alteração do ponto inicial para $\mathbf{x}^0 = [1.2, 1.5]$, mais próximo da solução que era conhecida, e obtivemos os resultados a seguir:

Tabela 3.10: Método de Broyden / Método de Newton Discreto

Método de Broyden	.	it.	Método de Newton Discreto	.
1.2	1.5	x^0	1.2	1.5
-0.490000	-1.096403	x^1	0.911363	1.167576
-0.201712	0.501952	x^2	0.984884	1.027189
2.352372	1.204746	x^3	0.999570	1.000882
0.473571	0.849263	x^4	1.000000	1.000000
0.803120	0.928917	x^5	-	-
1.093883	1.005736	x^6	-	-
1.011337	1.004782	x^7	-	-
0.994710	0.997929	x^8	-	-
0.999893	0.999959	x^9	-	-
1.000000	1.000000	x^{10}	-	-

O método de Broyden precisou de 10 iterações, enquanto o de Newton somente 4 iterações, sendo que para Broyden $||F|| = 1,6676.10^{-6}$, e tempo de 124,611ms, e para Newton $||F|| = 2,4255.10^{-6}$, e tempo de 91,580ms.

Problema 3.7 *fdennys2*

$$\begin{cases} f_1(x) = x_1 + x_2 - 3 \\ f_2(x) = x_1^2 - x_2^2 - 9 \end{cases}$$

A sequência de iterações pelo método de Broyden, para efeito de comparação com o método de Newton, com os seguintes parâmetros; $\mathbf{x}^0 = (1,5)^T$, $[40,39,0]$ (vide Anexo E) e $[40,1,0]$ (vide anexo A). está na tabela 3.11.

Tabela 3.11: Método de Broyden / Método de Newton Discreto

Método de Broyden	.	it.	Método de Newton Discreto	.
1	5	x^0	1	5
-2.000000	38.000000	x^1	1.250000	1.750000
-1.2261067	4.017301	x^2	3.000000	0.000000
-0.456610	3.525082	x^3	-	-
0.891676	2.083751	x^4	-	-
3.714010	-0.706438	x^5	-	-
2.996925	0.003097	x^6	-	-
2.999767	0.000231	x^7	-	-
3.000170	0.000017	x^8	-	-
3.000000	0.000000	x^9	-	-

O método de Broyden precisou de 9 iterações, enquanto o de Newton somente de 2 iterações, sendo que para Broyden $||F|| = 7,166.10^{-8}$, e o tempo é de 127,355ms e para Newton, $||F|| = 7,7427.10^{-7}$ e o tempo é de 97,892ms.

Problema 3.8 *fconte1*

$$\begin{cases} f_1(x) = x_1 + e^{x_1 - 1} + (x_2 + 3)^2 - 27 \\ f_2(x) = \dfrac{e^{x_2 - 2}}{x_1} + x_3^2 - 10 \\ f_3(x) = x_3 + sin(x_2 - 2) + x_2^2 - 7 \end{cases}$$

Neste sistema, ao aplicarmos o método de Broyden (brsol) - rotina Kelley [9], com os parâmetos $\mathbf{x}^0 = (4, 4, 4)$, e $[40, 39, 0]$, a sequência de pontos (vetores) que é gerada pelo método não convergiu.

Problema 3.9 *fconte2*

Idem ao caso anterior fdennys1.

Problema 3.10 *ffausett*

$$\begin{cases} f_1(x) = x_1^2 + x_2^2 + x_3^2 - 1 \\ f_2(x) = x_1^2 + x_3^2 - 0.25 \\ f_3(x) = x_1^2 + x_2^2 + 4x_3 \end{cases}$$

A sequência de iterações geradas pelo Método de Broyden (brsol) - rotina Kelley [9], com os parâmetos $\mathbf{x}^0 = (1, 1, 0)$, $[40, 39, 0]$ e $[40, 1, 0]$, aplicado a este sistema de equações, forneceu os resultados que estão listado na tabela 3.12.

O método de Broyden precisou de 21 iterações, enquanto o de Newton Discreto, somente de 4 iterações, sendo que para Broyden $||F|| = 1.208.10^{-6}$ e tempo de 127,028ms, e para Newton, $||F|| = 6.32110^{-7}$ e tempo de 107,163ms.

O método de Broyden também foi aplicado ao sistema de equações **Broyden tridiagonal** (3.1), porém o mesmo divergiu para todas as alternativas testadas, isto é: para n=5, n=10, n=20, n=50, n=100, n=200, n=500 e n=1000.

A partir deste conjunto de testes realizados, podemos verificar que o método de Broyden sempre realiza mais iterações que o método de Newton, quando ambos convergem, o que era esperado. Como estes problemas são de pequeno porte, a comparação do tempo de execução não é significativa.

A mesma comparação não pode ser feita para os problemas de porte maior até agora testados, uma vez que, nestes casos, a sequência gerada pelo método de Broyden não convergiu. No entanto, no próximo capítulo mostraremos os resultados do método de Broyden para um sistema não linear de médio porte, gerado a partir de uma aplicação física, seguido de uma análise de desempenho.

Tabela 3.12: Método de Broyden / Método de Newton Discreto

Mét. de Broyden	.	.	it.	Mét. de Newton Discreto	.	.
1	1	0	x^0	1	1	0
0.000000	0.250000	-2.000000	x^1	0.625000	.0875000	-0.250000
-1.094628	-1.090364	0.837098	x^2	0.468056	0.866071	-0.236111
7.328552	7.694604	-2.213840	x^3	0.441559	0.866025	-0.236068
-0.753429	-2.085309	0.299905	x^4	0.440764	0.866025	-0.236068
-1.929380	-1.670975	-0.087845	x^5	-	-	-
-1.912965	-2.502040	-0.142753	x^6	-	-	-
14.475482	9.390450	1.680764	x^7	-	-	-
-2.710762	-2.035715	-0.265595	x^8	-	-	-
-3.356908	-2.359304	-0.285718	x^9	-	-	-
-1.549597	-1.267533	-0.239501	x^{10}	-	-	-
-1.096896	-1.023310	-0.233334	x^{11}	-	-	-
-0.723693	-0.860489	-0.233013	x^{12}	-	-	-
-0.562266	-0.824329	-0.234241	x^{13}	-	-	-
-0.490237	-0.831443	-0.235017	x^{14}	-	-	-
-0.461282	-0.847388	-0.235485	x^{15}	-	-	-
-0.446462	-0.860503	-0.235862	x^{16}	-	-	-
-0.441176	-0.865628	-0.236041	x^{17}	-	-	-
-0.440726	-0.866040	-0.236065	x^{18}	-	-	-
-0.440763	-0.866015	-0.236067	x^{19}	-	-	-
-0.440766	-0.866019	-0.236067	x^{20}	-	-	-
-0.440763	-0.866024	-0.236067	x^{21}	-	-	-

Capítulo 4

H-equação Chandrasekhar

4.1 Introdução

As principais pesquisas de Lord Rayleigh [1842, 1919] foram principalmente em matemática, voltadas à ótica e sistemas vibratórios. Posteriormente trabalhou em pesquisas em outras áreas da física, tais como: som, teoria das ondas, visão colorida, eletrodinâmica, eletromagnetismo, dispersão da luz, vazão de liquídos, densidade de gases, viscosidade, capilaridade e fotografia.

De acordo com as leis físicas originadas pelas investigações de Lord Rayleigh em 1871, surge o problema específico sobre os campos de radiação da dispersão da luz na atmosfera, e que explicam sobre a iluminação e polarização da luz solar. Mas as equações que governam o particular problema de Rayleigh tiveram que esperar setenta e cinco anos para sua formulação e solução. Isto ocorreu quando Arthur Schuster formulou em 1905 o problema da Transferência Radioativa, de forma a explicar o aparecimento da absorção e emissão de linhas no espectro estrelar, e por Karl Schwarschild que introduziu em 1906 o conceito do equilíbrio radioativo nas atmosferas estrelares, o que motivou pesquisas neste campo.

Desde estes tempos, o objeto da Transferência Radioativa tem sido investigado principalmente por astrofísicos, e nos anos recentes tem atraído a atenção dos físicos tam-

bém, desde que essencialmente o mesmo problema ocorre na teoria da difusão dos neutrons.

Algumas definições inicialmente são introduzidas, a saber:

i) Intensidade específica.

A análise do campo de radiação requer que nós consideremos uma fonte de energia radiante dE_v, em um determinado intervalo de frequência $(v, v + dv)$, o qual é transportado através de um elemento de área d_σ em uma determinada direção, confinando um elemento de ângulo sólido dw, durante um tempo dt.

$$dE_v = I_v cos(v) dv d\sigma dw dt, \tag{4.1}$$

onde I_v é denominado de intensidade de radiação.

Um campo de radiação é dito *isotrópico* se a intensidade de radiação é independente da direção naquele ponto; e se a intensidade é a mesma em todos os pontos e todas as direções, o campo de radiação é dito ser *homogêneo* e isotrópico.

ii) O fluxo líquido.

O fluxo líquido em todas as direções é dado por:

$$dv d\sigma dt \int I_v cos(v) dw, \tag{4.2}$$

onde a integração deve ser sobre todo o ângulo sólido:

$$\pi F_v = \int I_v cos(v) dv. \tag{4.3}$$

A quantidade que ocorre na expressão (4.2) é chamada de *fluxo líquido* e define a razão do fluxo de energia radiante através de $d\sigma$ por unidade de área e por unidade de intervalo de freqüência.

iii) Densidade de radiação.

A densidade de energia $u_v dv$ de radiação no intervalo de freqüência considerado $(v, v+dv)$ em qualquer ponto é a quantidade de energia radiante por unidade de volume, no intervalo de freqüência considerado.

A integral da densidade de energia u é similar em termos da integral de intesidade I, então:

$$u = \int u_v dv = \frac{1}{c} \int I dw. \tag{4.4}$$

4.2 H-Equação Chandrasekhar

A equação de Chandrasekhar (4.5) é usada para resolver o problema de distribuição de transferência radioativa [5], [11].

$$F(H)(\mu) = H(\mu) - \left(1 - \frac{c}{2}\int_0^1 \frac{\mu H(v)dv}{\mu + v}\right)^{-1} = 0. \tag{4.5}$$

Discretizamos a integral no intervalo [0, 1] pela regra do ponto médio, dada por:

$$\int_0^1 f(\mu)d\mu \sim \frac{1}{N}\sum_{j=1}^{N} f(\mu_j) \tag{4.6}$$

onde $\mu_j = (j - \frac{1}{2})/N$ para $1 \leq j \leq N$. O resultado do problema discretizado é:

$$F(x)_i = x_i - \left(1 - \frac{c}{2N}\sum_{j=1}^{N} \frac{\mu_i x_j}{\mu_i + \mu_j}\right)^{-1} = 0. \tag{4.7}$$

Sabe-se que (4.5) e sua análoga discreta (4.7) têm duas soluções para $c \in (0, 1)$. Porém, somente uma destas soluções tem significado físico [1] [9]. Sabe-se também que o método de Newton com aproximação inicial $x^0 = (0, 0, 0, ..., 0)^T$ ou $x^0 = (1, 1, 1, ..., 1)^T$ converge para a solução desejada [9].

Utilizamos como parâmetros $N = 100$ e $c = 0.9$, e o vetor unitário como aproximação inicial e com estes valores comparamos os métodos de Newton Discreto, Newton Modificado e Shamanski (com parâmetro m=2 [9]), obtendo os resultados a seguir:

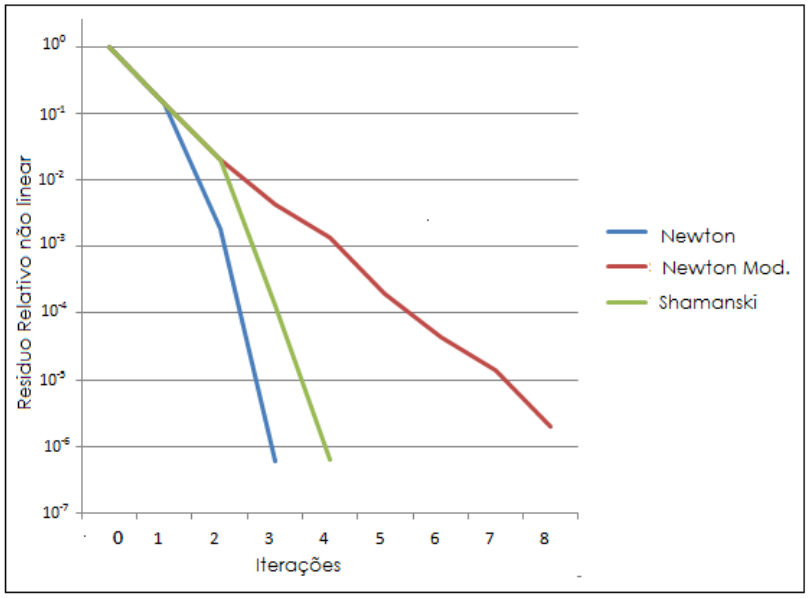
Newton D.	F_{rel}
Iter. (k)	$\frac{\|\|F(x_k)\|\|}{\|\|Fx_0\|\|}$
0	1.10^{0}
1	$1,478.10^{-1}$
2	$2,650.10^{-3}$
3	$7,710.10^{-7}$

Newton Mod.	F_{rel}
Iter. (k)	$\frac{\|\|F(x_k)\|\|}{\|\|Fx_0\|\|}$
0	1.10^{0}
1	$1,478.10^{-1}$
2	$3,070.10^{-2}$
3	$6,410.10^{-3}$
4	$1,388.10^{-3}$
5	$2,969.10^{-4}$
6	$6,334.10^{-5}$
7	$1,353.10^{-5}$
8	$2,889.10^{-6}$

Shamanski	F_{rel}
Iter. (k)	$\frac{\|\|F(x_k)\|\|}{\|\|Fx_0\|\|}$
0	1.10^{0}
1	$1,478.10^{-1}$
2	$3,070.10^{-2}$
3	$1,161.10^{-4}$
4	$7,980.10^{-7}$

Na figura 4.1 representamos os valores de $||F(x_k)||/||F(x_0)||$ para os três métodos. A concavidade da iteração no método de Newton Discreto indica sua convergência mais rápida do que a do método de Shamanski e do método de Newton Modificado.

Figura 4.1



Newton, Newton Modificado, Shamanski para H-Equação N=100, c= 0.9

Os tempos computacionais para obtenção das soluções foram: Newton Discreto - 530,595ms; Newton Modificado - 444,658ms e Shamanski - 696,800ms.

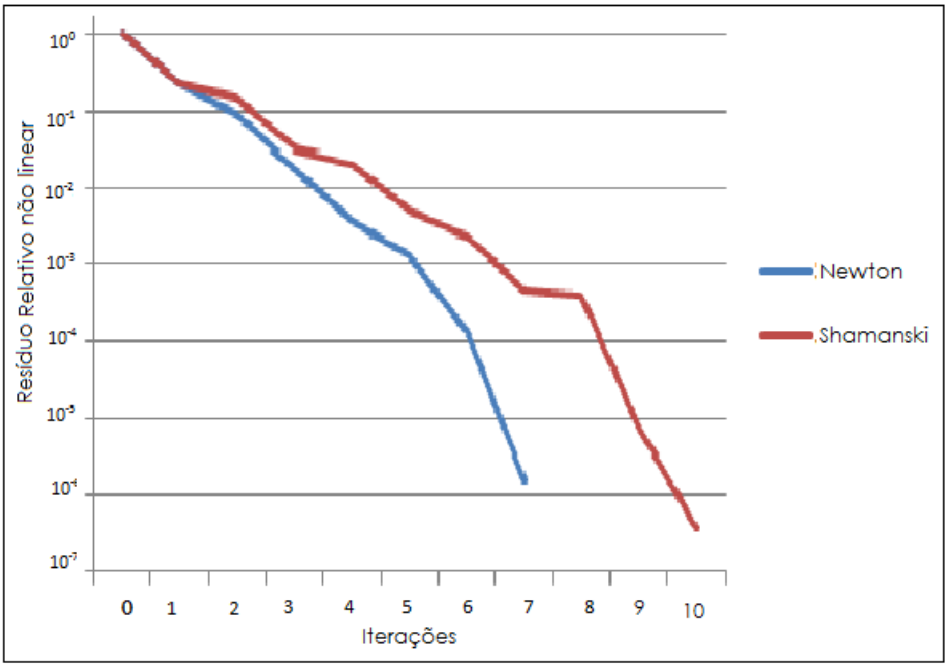
Procedemos também à resolução da equação de Chandrasekhar com $c = 0,9999$. Neste caso a solução está próxima da singularidade da Jacobiana. A par do parâmetro c, todos os demais permaneceram os mesmos do exemplo anterior. Nesta situação o método de Newton Discreto convergiu em 7 iterações, e tempo computacional de 1.381,014ms, enquanto que o método de Newton Modificado não convergiu em 40 iterações. Quando fixamos o número máximo em 200 iterações, Newton Modificado convergiu com 188 iterações e tempo de 767,663ms, o de Shamanski (m=2) com 10 iterações e tempo de 981,557ms. Seguem os resultados obtidos

Newton D.	F_{rel}
Iter. (k)	$\frac{\|\|F(x_k)\|\|}{\|\|Fx_0\|\|}$
0	1.10^{0}
1	$3,454.10^{-1}$
2	$9,540.10^{-2}$
3	$2,430.10^{-2}$
4	$5,850.10^{-3}$
5	$1,155.10^{-3}$
6	$1,212.10^{-4}$
7	$2,101.10^{-6}$

Shamanski	F_{rel}
Iter. (k)	$\frac{\|\|F(x_k)\|\|}{\|\|Fx_0\|\|}$
0	1.10^{0}
1	$3,454.10^{-1}$
2	$1,891.10^{-1}$
3	$5,211.10^{-2}$
4	$2,875.10^{-2}$
5	$7,000.10^{-3}$
6	$3,774.10^{-3}$
7	$6,604.10^{-4}$
8	$6,000.10^{-4}$
9	$8,442.10^{-6}$
10	$5,745.10^{-7}$

Neste caso, apesar do método de Newton modificado realizar muitas iterações a mais que os métodos de Newton Discreto e Shamanski, gastou o menor tempo de CPU.

Figura 4.2



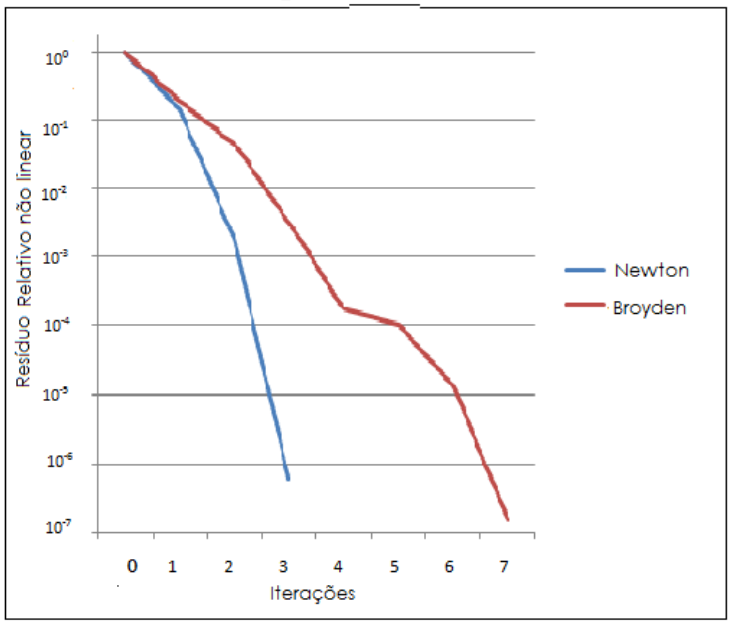
Newton, Shamanski
para H-Equação N=100, c= 0.9999

Além destas comparações, procedemos também um comparativo com o método de Broyden, (rotina Kelley - brsol.m) [9], em relação ao método de Newton Discreto - (nsol.m). Os parâmetros utilizados foram os mesmos que os anteriores, com N = 100 e c=0.9, sendo que o método de Broyden convergiu em 7 iterações e tempo computacional de 144,803ms. Devemos observar aqui que o tempo computacional gasto pelo método de Broyden para efetuar 7 iterações foi bem inferior ao de Newton, que realizou 3 iterações. Seguem os resultados obtidos.

Newton D.	F_{rel}
Iter. (k)	$\frac{\|\|F(x_k)\|\|}{\|\|Fx_0\|\|}$
0	1.10^{0}
1	$1,478.10^{-1}$
2	$2,650.10^{-3}$
3	$7,710.10^{-7}$

Broyden	F_{rel}
Iter. (k)	$\frac{\|\|F(x_k)\|\|}{\|\|Fx_0\|\|}$
0	1.10^{0}
1	$2,827.10^{-1}$
2	$6,571.10^{-2}$
3	$5,050.10^{-3}$
4	$2,565.10^{-4}$
5	$1,017.10^{-4}$
6	$1,356.10^{-5}$
7	$1,600.10^{-8}$

Figura 4.3



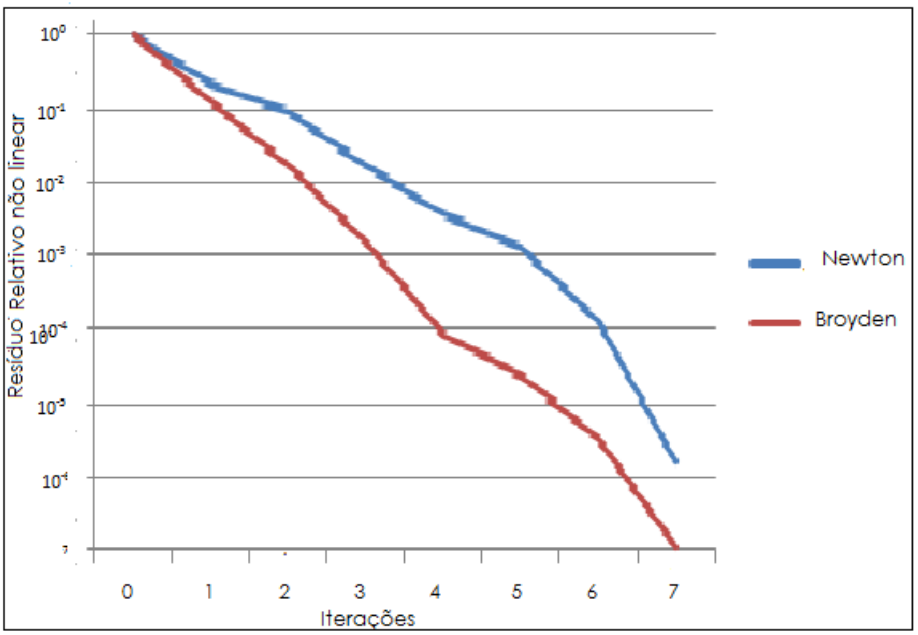
Newton - Broyden
para H-Equação N=100, c= 0.9

Analogamente, comparativo do método de Broyden com o de Newton Discreto, com parâmetros N=100 e c=0,9999. Ambos convergiram com 7 iterações e tempo computacional de 1381,014ms e 234,317ms, respectivamente. Novamente o método de Broyden converge em menos tempo, confirmando que, para este problema, o custo de avaliar a Jacobiana por diferenças finitas em toda iteração é alto. Seguem os resultados.

Newton D.	F_{rel}
Iter. (n)	$\frac{\|\|F(x_n)\|\|}{\|\|Fx_0\|\|}$
0	1.10^{0}
1	$3,454.10^{-1}$
2	$9,540.10^{-2}$
3	$2,430.10^{-2}$
4	$5,850.10^{-3}$
5	$1,155.10^{-3}$
6	$1,212.10^{-4}$
7	$2,101.10^{-6}$

Broyden	F_{rel}
Iter. (n)	$\frac{\|\|F(x_n)\|\|}{\|\|Fx_0\|\|}$
0	1.10^{0}
1	$1,044.10^{-1}$
2	$2,428.10^{-2}$
3	$1,867.10^{-3}$
4	$9,457.10^{-5}$
5	$3,751.10^{-5}$
6	$5,011.10^{-6}$
7	$6,000.10^{-8}$

Figura 4-4



Newton - Broyden
para H-Equação N = 100 c=0,9999

Os bons resultados conseguidos com os métodos alternativos mostram a importância de utilização destes, com exceção do caso em que o método de Newton Modificado necessitou de 188 iterações (N=100 e c = 0,9999). Isto implica que uma boa aplicação dos métodos apresenta, na maioria dos testes, convergência rápida e resultados bastante precisos.

Considerações finais e perspectivas futuras

Relembrando o objetivo do presente trabalho, "O objetivo do projeto é discutir os métodos, algoritmos, e análise envolvida na solução computacional de problemas não lineares e transcendentais", observamos que não nos preocupamos com aspectos teóricos relacionados à existência e unicidade da solução de sistemas não lineares.

É também aparente que podemos encontrar ou não soluções aproximadas para a maioria dos problemas não lineares. Mas os métodos abordados apresentaram, em geral, desempenho muito bom na prática.

Os métodos trabalhados foram Newton (Puro), Newton Modificado, Newton Discreto, Shamanski e Broyden. Estudamos motivação, algoritmos e análise de convergência de cada um destes métodos. Trabalhamos com suas implementações computacionais aplicadas a experimentos numéricos que nos permitiram realizar a comparação entre os mesmos. Destacamos que nossa opção foi pelo estudo dos métodos mais clássicos para resolução de sistemas não lineares, uma vez que todos aqui abordados têm importância comprovada, dados os inúmeros trabalhos que sobre eles ainda são publicados atualmente.

Nosso trabalho envolveu várias equações de dimensões reduzidas, e equações de grande porte com sistemas tridiagonais [7], além da aplicação destes métodos à resolução da equação de Chandrasekhar, a qual, discretizada, gerou um sistema não linear de porte médio.

Observamos que o método de Newton teve um ótimo comportamento na resolução de todos os problemas testados, mesmo que envolvendo tempo computacional superior em alguns casos. Entretanto os demais métodos apresentaram algumas deficiências

em relação a algumas equações específicas (tridiagonais), ressaltando, neste caso o de Broyden para equações de grande porte, onde o mesmo não conseguiu obter solução.

Como perspectivas futuras, acredito que a implementação destes métodos, em linguagens de alto desempenho, poderia ser experimentada, para confrontrar com os resultados do MatLab, onde foram rodadas todas as questões em pauta.

Referências Bibliográficas

[1] **Chandrasekhar** S. [1960], Radiative Transfer, Dover Publications, Inc. New York, NY, USA.

[2] **Conte** S. D., **DE Boor** C. [1987], Elementary Numerical Analysis,third edition, McGraw-Hill International Editions, Singapore.

[3] **Cunha**, C. [1993], Métodos Numéricos - Para as engenharias e ciências aplicadas, Editora da Universidade Estadual de Campinas, UNICAMP.

[4] **Dennis**, Jr J.E., **Schnabel**, B. [1983], Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Englewood Cliffs, NJ, Prentic-Hall Inc.

[5] **Fausett**, L. V. [1999], Applied Numerical Analysis using Matlab, Prentice Hall, A Pearson Education Company, Upper Addle River NJ.

[6] **Gilat**, A. [2005], Matlab An Introduction with Applications, second edition, John Willey Sons. Inc., Columbia, Ohio.

[7] **Gomes Ruggiero,** M. A.; **Lopes**, V. L. [1996], Cálculo Numérico-aspectos teóricos e computacionais, segunda edição, Makron Books, São Paulo - SP - Brasil.

[8] **Hubbard**, J. R.; **Hubbard**, B. B. [1999], Vector Calculus, Linear Algebra, and Differential Forms, Prentice Hall Inc., NJ.

[9] **Kelley**, C.T. [1995], Iterative Methods for Linear and Nonlinear Equations, Society for Industrial and Applied Mathematics, SIAM, Philadelphia, PA.

[10] **Salas**, S.L.; **Hille**, E.; **Anderson**, J. T. [1986], Calculus one and several variables, fifth edition, John Wiley & Sons Inc., NY.

[11] **Shamanskii** V. E. [1967], A modification of Newton's Method, Ukran, Mat. Zh .19 -pp 133-138.

[12] **Shenk**, Al. [1988], Calculus and Analytic Geometry, fourth edition, Scott, Foreman and Company, San Diego, CA.

Anexos

- **Anexo A - Programa Matlab para solução de equações não lineares.**

```
function [sol,  it_hist,   ierr] = nsol(x, f, tol, parms)
% Newton solver, locally convergent solver for f(x) = 0
% Hybrid of Newton, Shamanskii, Chord
% C. T. Kelley, November 26, 1993
% This code comes with no guarantee or warranty of any kind.
% function [sol, it_hist, ierr] = nsol(x, f, tol, parms)
% inputs:
% initial iterate = x
% function = f
% tol = [atol, rtol] relative/absolute
% error tolerances
% parms = [maxit, isham, rsham]
% maxit = maxmium number of iterations
% default = 40
% isham, rsham: The Jacobian matrix is computed and factored after isham updates
of x or whenever the ratio of successive infinity norms of the nonlinear residual exceeds
rsham.
% isham = 1, rsham = 0 is Newton's method,
% isham = -1, rsham = 1 is the chord method
% isham = m, rsham = 1 is the Shamanskii method.
% defaults = [40, 1000, .5]
% output:
% sol = solution
% it_hist = infinity norms of nonlinear residuals
```

```
% for the iteration
% ierr = 0 upon successful termination
% ierr = 1 if either after maxit iterations the termination criterion is not satsifiedor the ratio of successive nonlinear residuals exceeds 1. In this latter case, the iteration is terminted.
% internal parameter:
% debug = turns on/off iteration statistics display as the iteration progresses
% Requires: diffjac.m, dirder.m
% Here is an example. The example computes pi as a root of sin(x) with Newton's method and plots the iteration history.
% x=3; tol=[1.d-6, 1.d-6]; params=[40, 1, 0];
% [result, errs,it_hist] = nsol(x, 'sin', tol, params);
% result
% semilogy(errs)
% set the debug parameter, 1 turns display on, otherwise off
debug=1;
% initialize it_hist, ierr, and set the iteration parameters
% ierr = 0;
% maxit=40;
% isham=1000;
% rsham=.5;
if nargin == 4
maxit=parms(1); isham=parms(2); rsham=parms(3);
end
rtol=tol(2); atol=tol(1);
it_hist=[];
n = length(x);
fnrm=1;
itc=0;
% evaluate f at the initial iterate compute the stop tolerance
f0= feval(f,x);
```

```
fnrm=norm(f0,inf);
fnrm
it_hist=[it_hist,fnrm];
fnrmo=1;
itsham=isham;
stop_tol=atol+rtol*fnrm;
% main iteration loop
while(fnrm > stop_tol \{&} itc < maxit)
% keep track of the ratio (rat = fnrm/frnmo) of successive residual norms and the iteration counter (itc)
rat=fnrm/fnrmo;
rat
outstat(itc+1, :) = [itc fnrm rat];
fnrmo=fnrm;
itc=itc+1;
% evaluate and factor the Jacobian on the first iteration, every isham iterates, or if the ratio of successive residual norm is too large
if(itc == 1 | rat > rsham | itsham == 0)
itsham=isham;
[l, u] = diffjac(x,f,f0);
end
itsham=itsham-1;
% compute the step
tmp = -l\f0;
step = u\tmp;
xold=x;
x = x + step
f0= feval(f,x);
fnrm=norm(f0,inf)
    it_hist=[it_hist,fnrm];
rat=fnrm/fnrmo;
```

```
if debug==1
disp([itc fnrm rat])
end
outstat(itc+1, :)=[itc fnrm rat];
% if residual norms increase, terminate, set error flag.
if rat >= 1
ierr=1;
sol=xold;
disp('increase in residual')
disp(outstat)
return;
end
sol=x;
% if parms
debug==1
disp(outstat)
end
% on failure, set the error flag.
if fnrm > stop_tol
ierr = 1;
end.
```

Para operacionalizar nsol.m são necessárias as *functions*:

i) diffjac.m

```
function [l, u] =diffjac(x, f, f0)
% compute a forward difference Jacobian f'(x), return lu factors
% uses dirder.m to compute the columns
% C. T. Kelley, November 25, 1993
% This code comes with no guarantee or warranty of any kind.
% inputs:
% x, f = point and function
% f0 = f(x), preevaluated
```

```
n=length(x);
for j=1:n
zz=zeros(n,1);
zz(j)=1;
jac(:,j)=dirder(x,zz,f,f0);
end
[l, u] = lu(jac);
```

ii) dirder.m

```
function z = dirder(x,w,f,f0)
% Finite difference directional derivative
% Approximate f'(x) w
% C. T. Kelley, November 25, 1993
% This code comes with no guarantee or warranty of any kind.
% function z = dirder(x,w,f,f0)
% inputs:
% x, w = point and direction
% f = function
% f0 = f(x), in nonlinear iterations
% f(x) has usually been computed
% before the call to dirder
% Hardwired difference increment.
epsnew=1.d-7;
n=length(x);
% scale the step
if norm(w) == 0
z=zeros(n,1);
return
end
epsnew = epsnew/norm(w);
if norm(x) > 0
epsnew=epsnew*norm(x);
```

```
end
% del and f1 could share the same space if storage is more important than clarity
del=x+epsnew*w;
f1=feval(f,del);
z = (f1 - f0)/epsnew;
```

- **Anexo B - Programa Matlab para solução de equações não lineares e transcendentais.**

```
function[x, y]=Secant(fun, a, b, tol, max)
% Find a zero using secant method.
% fun,    string containing name of function
% a, b     first two estimates of zero
% tol ;    tolerance for change in computed zero
% max;    maximum number of iterations
% x;    vector aproximation to zero.
% y;    vector of function values fun(x)
x(1)=a;    x(2)=b;
y(1)=feval(fun, x(1));    y(2)=feval(fun, x(2));
for i=2:max
x(i+1)=x(i)-y(i)*(x(i)-x(i-1))/(y(i)-y(i-1));
y(i+1)=feval(fun, x(i+1));
if (abs(x(i+1)-x(i))<tol)
disp('Secant method has converged'); break;
end
if y(i) == 0
disp('exact zero found'); break;
end
iter=i;
end
if(iter>=max)
disp('zero not found to desired tolerance');
end
n =length(x);    k=1:n;    out=[k', x' y'];
disp(' step    x    y')
disp(out)
```

- **Anexo C - Programa Matlab para solução de sistemas de equações não lineares.**

```
function [x0,k] = newton(e)
x0=input('entre com a aproximação inicial ');
tic
toc
tic;
k=0;
F=feval('fcarlos',x0);
normf=norm(F,inf);
while (normf >= e)
A=feval('jacmatcarlos',x0)
b=-F
s0=A\b;
x1=x0+s0
x0=x1;
F=feval('fcarlos',x0)
normf=norm(F,inf);
k=k+1
toc;
end
```

Para operacionalizar newton.m são necessárias:

i) uma *function* que forneça a função que define o sistema não linear (no exemplo fcarlos.m)

ii) uma *function* que forneça a Jacobiana da função (no exemplo jacmatcarlos.m).

- **Anexo D - Programa Matlab Newton Modificado para solução de sistemas de equações não lineares.**

```
function [x0,k] = newtonmodificado(e)
x0=input('entre com a aproximação inicial ');
tic
toc
tic;
k=0;
F=feval('fcarlos',x0);
normf=norm(F,inf);
A=feval('jacmatcarlos',x0)
while (normf >= e)
b=-F;
s0=A\b;
x1=x0+s0;
x0=x1;
F=feval('fcarlos',x0);
normf=norm(F,inf)
k=k+1;
toc;
end
x0
```

Para operacionalizar o programa são necessário os programas:

i) O programa com extensão m da função. (no exemplo fcarlos.m)

ii) O programa com extensão m da Jacobiana da função (no exemplo jacmatcarlos.m).

- **Anexo E - Programa Matlab Broyden rotina Kelley brsol**

```
function [sol, it_hist, ierr] = brsol(x,f,tol, parms)
% Broyden's Method solver, locally convergent
% solver for f(x) = 0
% C. T. Kelley, June 29, 1994
% This code comes with no guarantee or warranty of any kind.
% function [sol, ithist, ierr] = brsol(x,f,tol,parms)
% inputs:
% initial iterate = x
% function = f
% tol = [atol, rtol] relative/absolute
% error tolerances for the nonlinear iteration
% parms = [maxit, maxdim, linprob]
% maxit = maxmium number of nonlinear iterations
% default = 40
% maxdim = maximum number of Broyden iterations
% before restart, so maxdim+3 vectors are stored (see text). By making the code
a bit more subtle (putting z and F in the same place) one can reduce this overhead to
maxdim+2 vectors.
% default = 40
% linprob = 0 for nonlinear problem
% = 1 for linear problem
% if linprob = 1 an increase in the residual does not result in termination
% default = 0
% output:
% sol = solution
% ithist(maxit) = scaled l2 norms of nonlinear residuals
% ierr = 0 upon successful termination
% ierr = 1 if either after maxit iterations
```

```
% the termination criterion is not satsified or the ratio of successive nonlinear residuals exceeds 1. In this latter case, the iteration is terminated.
% internal parameter:
% debug = turns on/off iteration statistics display as the iteration progresses
% set the debug parameter, 1 turns display on, otherwise off
debug=1;
% initialize it_hist, ierr, and set the iteration parameters
ierr = 0;
maxit=40; maxdim=39; linprob = 0; it_histx=zeros(maxit);
if nargin == 4
maxit=parms(1); maxdim=parms(2)-1; linprob=parms(3);
end
rtol=tol(2); atol=tol(1); n = length(x); fnrm=1; itc=0; nbroy=0;
% evaluate f at the initial iterate
% compute the stop tolerance
f0=feval(f,x);
fc=f0;
fnrm=norm(f0)/sqrt(n);
it_hist(itc+1)=fnrm;
fnrmo=1;
stop_tol=atol + rtol*fnrm;
outstat(itc+1, :) = [itc fnrm 0];
% initialize the iteration history storage matrices
stp=zeros(n,maxdim);
stp_nrm=zeros(maxdim,1);
% Set the initial step to -F, compute the step norm
stp(:,1) = -fc;
stp_nrm(1)=stp(:,1)'*stp(:,1);
% main iteration loop
while(itc < maxit)
nbroy=nbroy+1;
```

```
% keep track of successive residual norms and the iteration counter (itc)
fnrmo=fnrm; itc=itc+1;
% compute the new point, test for termination before adding to iteration history
x = x + stp(:,nbroy)
fc=feval(f,x);
fnrm=norm(fc)/sqrt(n);
 it_hist(itc+1)=fnrm;
rat=fnrm/fnrmo
outstat(itc+1, :) = [itc fnrm rat];
if debug==1
disp(outstat(itc+1,:))
disp(fnrm)
fnrm
disp(itc)
itc
end
% test for termination before computing the next w
    if fnrm <= stop_tol
sol=x;
disp(fnrm)
fnrm
disp(itc)
itc
return;
end
% if residual norms increase, terminate, set error flag
% if rat >= 1 & linprob == 0
% ierr=1;
% disp('increase in residual')
% disp(outstat)
% return;
```

```
% end
% if there's room, compute the step and step norm and add to the iteration history
if nbroy < maxdim+1
z=-fc;
if nbroy > 1
for kbr = 1:nbroy-1
                    z=z+stp(:,kbr+1)*((stp(:,kbr)'*z)/stp_nrm(kbr));
end
end
% store the new step and step norm
zz=stp(:,nbroy)'*z;
                   zz=zz/stp_nrm(nbroy);
stp(:,nbroy+1)=z/(1-zz);
          stp_nrm(nbroy+1)=stp(:,nbroy+1)'*stp(:,nbroy+1);
else
% out of room, time to restart
stp(:,1)=-fc;
          stp_nrm(1)=stp(:,1)'*stp(:,1);
nbroy=0;
end
% end while
end
sol=x;
disp(fnrm)
fnrm
disp(itc)
itc
if debug==1
disp(outstat)
disp(fnrm)
fnrm
```

```
disp(itc)
itc
end
% on failure, set the error flag
if fnrm > stop_tol
ierr = 1;
disp(fnrm)
fnrm
disp(itc)
itc
end.
```

Para operacionalizar o programa são necessários os programas:

i) O programa com extensão m da função. (no exemplo fconte1.m)

ii) O programa principal com extensão m `broy_conte1` do exemplo.

```
% Programa principal para a resolucao de Sistemas nao - lineares pelos metodo de Broyden usando a rotina de Kelley .
x = [4;4;4] ;
parms = input('Entre com os parametos iniciais: ');
f = 'fconte1';
tol = input('Entre com o erro aceitavel: ');
tic;
[sol, it_hist, ierr] = brsol(x,f,tol, parms)
toc
disp('A melhor aproximacao para a solucao e: ');
sol.
```