

Série Arquimedes, Volume 2, Anais do DINCON 2003, pp. 2951-3009.
2º Congresso Temático de Aplicações de Dinâmica e Controle da
Sociedade Brasileira de Matemática Aplicada e Computacional (SBMAC).
São José dos Campos, SP, Brasil, 18-22 Agosto de 2003.
Editores: J. M. Balthazar, G. N. da Silva, M. Tsuchida,
M. Boaventura, L. S. Góes e J. D. S. Silva.

NOÇÕES BÁSICAS DE ÁLGEBRA COMPUTACIONAL E APLICAÇÕES NAS CIÊNCIAS, VIA MATLAB®.

Otilia Helena Gerotto/ Faculdade de
Engenharia Mecânica, Departamento de
Projeto Mecânico, C. P. 6122, 13083-970,
UNICAMP, Campinas, SP, Brasil.
E-mail: ogerotto@linkway.com.br

José Manoel Balthazar/ Departamento de
Estatística, Matemática Aplicada e Computação,
UNESP- Rio Claro, CP 178, 13500-230 Rio Claro,
SP, Brasil e Prof. Visitante Departamento de
Projeto Mecânico, Faculdade de Engenharia
Mecânica, UNICAMP, Campinas
E-mail: jmbaltha@rc.unesp.br

1. RESUMO

Este trabalho apresenta ao leitor, em caráter introdutório a revisão de algumas técnicas de solução para Sistemas algébricos não-lineares utilizando-se dos recursos da linguagem MATLAB®. Trata-se, neste trabalho da solução de Sistemas de Equações que serão submetidos à métodos numéricos Iterativos quando se utiliza diferentes Técnicas de soluções, usuais nas diversas aplicações da ciência. Efetuam-se comparações entre elas, mostrando a utilidade e eficácia de cada uma delas.

2. PALAVRAS CHAVES

Sistemas Não Lineares , Métodos Iterativos. Algoritmos em MATLAB®

3. INTRODUÇÃO

O objetivo deste trabalho é o apresentar uma revisão do desenvolvimento e análise de Técnicas de solução de sistemas de equações não-lineares, através da utilização de Métodos Iterativos para obtenção de uma solução aproximada, cuja aproximação seja dada pelo usuário.

Os Métodos apresentados, neste trabalho, foram programados e testados pelos autores, em linguagem MATLAB®.

Apresentam-se Planilhas dos Resultados das análises efetuadas, no decorrer do texto. Esclarece-se que trata-se de trabalho inicial, destinados aos estudantes que pretendem efetuar estudos pós-graduados nas áreas de ciência da Engenharia ou áreas afins.

Os Métodos Iterativos, são amplamente utilizados na literatura corrente e, tem-se as mais diversas aplicações onde eles são exigidos.

A realização deste trabalho, tem por objetivo apresentar o comportamento de alguns métodos iterativos, documentar e apresentar algumas conclusões de forma a permitir a sua utilização para o usuário.

No presente trabalho, inicialmente mostrou-se cada Método Iterativo e suas potencialidades, a análise de seus algoritmos e suas aplicações na Ciência da Engenharia. Em seguida, programou-se cada Método em MATLAB® para posterior uso de um conjunto de equações algébricas não lineares de acordo com a aplicabilidade do Método Iterativo.

Logo após a programação dos principais Métodos Iterativos, escolheu-se vários conjuntos de equações para submetê-los aos Métodos programados onde observou-se o comportamento da “Performance” de cada Método, o que permitiu documentar-se os resultados e, tirar algumas conclusões importantes que poderão ser úteis aos leitores.

Organizou-se este texto, subdividido-o nos itens seguintes:

- no item 4; é realizado um estudo inicial sobre Modelagem e Teoria dos Erros e suas implicações ao se tratar de soluções computacionais,
- no item 5, discute-se alguns Métodos Iterativos para se obter Zeros Reais de Funções Reais;
- no item 6, são apresentados os principais Métodos Iterativos utilizados na resolução de problemas de Programação Não Linear.
- Neste parte do trabalho tem-se todo o material necessário para se utilizar os métodos aplicados a um conjunto de sistemas de equações. Neste caso, o conjunto de testes está documentado e, servirá de apoio para as análises que o leitor achar necessárias para sua utilização.
- No item 7, apresentam-se algumas comparações entre os diversos métodos estudados.
- No item 8, apresentam-se as rotinas dos métodos numéricos utilizados, neste trabalho
- No item 9, apresentam-se os rotinas “function” utilizadas no item 8;
- No item 10 incluem-se um adendo de programas de álgebra linear computacional com o objetivo de completar o texto, apresentado.
- No item 11, listam-se os programas, utilizados no item 10.
- No item 12, apresentam-se algumas conclusões do presente trabalho.
- No item 13, apresentam-se as referências bibliográficas.
- No item 14, fazem-se alguns agradecimentos.

4. NOÇÕES BÁSICAS DE MODELAGEM MATEMÁTICA

Sabe-se que um modelo é uma representação(substitutiva) da realidade.

O verbo modelar significa construir representações idealizadas de uma situação real; logo um modelo matemático pode ser entendido(ou definido)como uma formulação ou como uma equação que expressa as características essenciais de um sistema(Físico, de Engenharia, de Biologia, etc.) ou um processo, em termos matemáticos.

Existem muitos tipos de modelos, alguns são determinísticos enquanto outros são probabilísticos(em natureza); alguns são modelos simplesmente descritivos, enquanto que outros são modelos de decisão, alguns são usados para fazer cálculos, enquanto que outros para investigar ou prever propósitos.

Nos casos em que está-se interessado no presente trabalho, técnica de construção de modelos matemáticos pode ser sumariizado como apresentado na Figura 4-1.

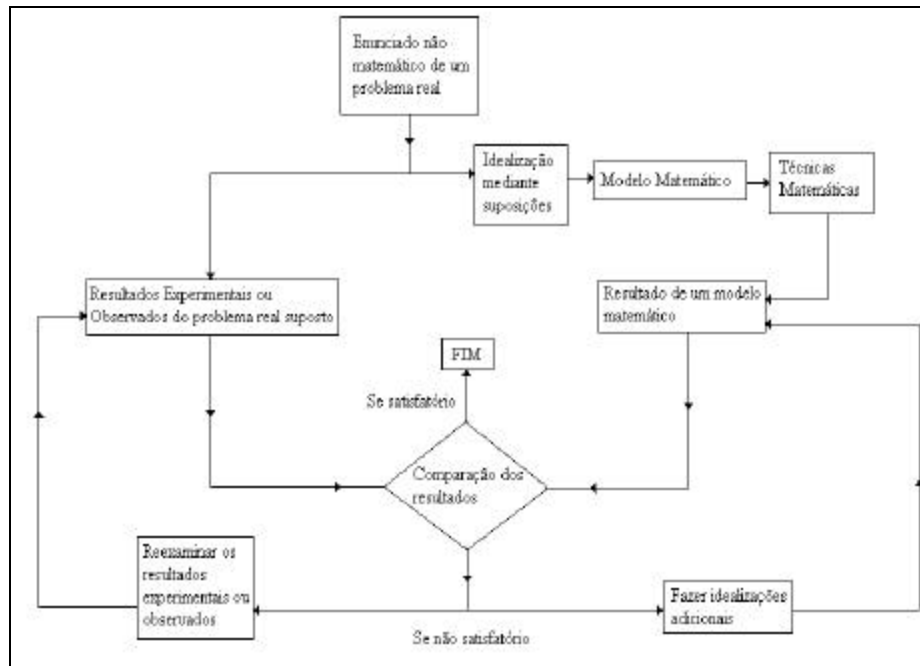


Figura 4.1- 1 Esquema Técnicas de Modelagem de Problemas Reais

É extremamente difícil classificar os modelos matemáticos do sistema. O material, discutido aqui, serve como ponto de partida para outras discussões mais amplas. Os resultados obtidos através de cuidadosos ensaios experimentais são sempre diferentes daqueles que seriam obtidos através de cuidadosos ensaios experimentais.

Desta forma, é claro que não existe um único modelo matemático para o problema real, mas vários, cada um com diferentes graus de aproximação e de validade.

Nos estágios iniciais de um projeto de engenharia, física, biologia, etc. procura-se começar com um modelo matemático mais simples com o objetivo de adquirir um bom conhecimento do problema, em tela, sem um esforço excessivo.

Entretanto à medida que se deseja modelos matemáticos com menos limitação, torna-se necessário acrescentar novos aspectos do problema real que poderão acarretar na elaboração de modelos matemáticos mais complexos e de difícil solução.

O **estudo de modelos** tem como objetivo principal desenvolver os fundamentos necessários, tendo em vista às aplicações em engenharia, física, biologia, etc. Ressalta-se que este estudo mostra semelhanças de comportamento para uma grande variedade de sistemas Engenharia, Física, Biologia, etc.

Finalmente, pode-se dizer que, geralmente, os erros, cometidos no processo de estudo de um problema real, provém de três fontes, distintas:

- Modelagem Matemática do Problema(inevitável);
- Precisão dos Dados e;
- Erros de Arredondamentos(na fase de resolução)e truncamentos dos Processos

Nos estudos qualitativos e quantitativos dos problemas de Engenharia, Física, Biologia, etc., deve-se utilizar algoritmos de análise, adequados para modelos matemáticos que foram associados à estes problemas. Um algoritmo pode ser entendido como uma seqüência de instrução ordenadas, de maneira, a dar, em seu decurso a solução para um problema específico. Nossa preocupação será com os algoritmos voltados para o processamento numérico; eles devem ter a seguinte características:

- Algoritmo deve identificar todas as etapas do modelo;
- Algoritmo pode falhar por violar restrições físicas da máquina, que são detectadas, em tempo de execução;

- Como muitos problemas numéricos são resolvidos por métodos iterativos (repetitivos), faz-se necessário estabelecer um Critério de Parada para que o algoritmo possa terminar após um número finito de passos, sendo aconselhável que o número de passos seja determinado, à priori;
- Algoritmo deve ter independência de máquina;
- algoritmo deve ter eficiência (Economia);
- Resultado \leq Valor Aproximado \pm Limite Erro.

Ressalta-se a importância do conhecimento da precisão dos dados depende de vários fatores inclusive do problema analisado, porém tendo essa premissa em mãos, deve-se assegurar a minimização dos erros resultantes da representação de números no computador, como também os erros resultantes das operações numéricas efetuadas nesse processo.

Sabe-se que a representação de um número, depende da base disponível na máquina em operação(base binária, decimal, hexadecimal, etc.) e do máximo de dígitos usados na sua representação nessa máquina(quantos inteiros, quantas decimais, etc.).

Dependendo, então, das opções, logo acima assinaladas, tem-se um resultado que se pode dizer: *mais ou menos preciso*.

Outro ponto que pode-se observar é que um número pode ter uma representação finita em uma certa base, decimal, por exemplo ou infinita em outra base. Tomando-se, como um exemplo: o número 0.11 na base decimal que quando representado no sistema binário não possui representação finita. Diz-se, então, que um computador representa um número real no sistema denominado de Aritmética de Ponto Flutuante, onde a sua mantissa possui um certo número de dígitos e o expoente de 10 indica o expoente no intervalo considerado. Como, em geral, um certo número sempre vai estar limitado à sua representação na aritmética de ponto flutuante, sabe-se então que ele nunca poderá ser totalmente representado, sofrendo sempre um *arredondamento* ou um *truncamento*, dependendo das regras que se aplicam ao se desprezar os dígitos restantes. Como destacado anteriormente, um número ao ser usado por um determinada máquina, será tratado com um certo truncamento ou arredondamento, o que com certeza vai resultar em um certo erro a ser considerado e tratado pelo programa do usuário.

O usuário, rodando o programa que está sendo processado, vai sendo gerado erros que vão se propagando pelas diversas fases do programa, ocasionando erros imensos no final da execução deste. Logo, tem-se que ter sempre em mente que um erro é sempre relativo, motivo pelo qual este erro deve ser previsto e, tratado pelo programa.

Então, define-se este erro (E)(veja por ex. Burden e Faires, 1993) como sendo : Aproximação da iteração atual(E_{atual}) - iteração anterior($E_{anterior}$) dividida pela aproximação atual e impõem-se que ele deve estar dentro de uma certa precisão(ϵ) dada a priori, i.e.,

$$E = \left| \frac{E_{atual} - E_{anterior}}{E_{atual}} \right| < \epsilon$$

A seguir, discute-se os tópicos de interesse destas notas.

5. ZEROS REAIS DE FUNÇÕES REAIS

Nas mais diversas áreas das ciências ocorrem situações que envolvem a resolução de uma equação do tipo $f(x) = 0$. Desta forma, deve-se desenvolver métodos numéricos para resoluções de equações lineares e não lineares usados na busca das raízes das equações, ou os Zeros Reais de $f(x)$. Em geral, os métodos, utilizados apresentam duas fases distintas:

- Fase I : Localização ou Isolamento das Raízes, que consiste em obter um intervalo que contém a raiz e,
- Fase II - Refinamento : Escolhidas as aproximações iniciais no intervalo encontrado na Fase I, deve-se melhorá-las sucessivamente até se obter uma aproximação para a raiz dentro de uma precisão pré- fixada ϵ .

Utiliza-se dos e *Métodos Iterativos* ou seja, de uma Sequência de Instruções que se repetem até que seja obtida a precisão desejada. Existem vários exemplos de métodos iterativos para se obter os zeros reais de funções, como por exemplo: Método da Bissecção, Método da Posição Falsa, Método do Ponto Fixo (MPF), os que não serão detalhados neste trabalho, pois são amplamente conhecidos dos textos introdutórios de cálculo numérico.

Observa-se que os métodos iterativos para se obter zeros reais de funções fornecem apenas uma aproximação para a solução exata.

É necessário destacar que toda vez que se escolhe um determinado método a ser aplicado, tem-se que fazer um estudo de sua convergência, pois pode-se estar aplicando um método que se distancia das raízes da equação, ou que apresenta um número muito grande de iterações levando a um esforço extra computacional, por vezes desnecessário.

Deve-se escolher o algoritmo a ser utilizado, observando-se os quesitos: Deve-se então analisar-se e escolher-se um método mais apropriado para cada caso analisado. Além do teste de parada que é utilizado para cada método, deve-se ter o cuidado de se estipular um *número máximo de iterações*, pois, deve-se evitar que o programa(em execução) entre em “*looping*” devido a erros no próprio programa ou à inadequação do método usado para o problema em questão. Pode-se acrescentar que, no caso polinomial, se $n = 2$, sabe-se determinar os zeros de $f_2(x)$. Existem fórmulas fechadas, semelhantes à fórmula para polinômios de grau 2, mas bem mais complicadas, para zeros de polinômios de grau 3 e 4. Agora, para $n \geq 5$, em geral, não existem fórmulas explícitas e então deve-se recorrer a usar métodos iterativos para encontrar zeros de polinômios.

Em seguida, apresenta-se alguns métodos iterativos para se encontrar zeros de polinômios, os quais são: Método de Newton-Raphson, um dos métodos mais eficazes para determinar zeros reais de funções reais e o Método dos Gradientes Conjugados.

5.1 MÉTODO DE NEWTON-RAPHSON

Quando se estuda um Método do Ponto Fixo(Burden e Faires, 1993) pode-se ver que: Uma das condições de convergência é que $|f'(x)| < M < 1$, $\forall x \in I$ onde I é o intervalo centrado na raiz e que a convergência do método depende diretamente da função de iteração escolhida. Este método é útil, por exemplo, na busca do mapa de Poincaré, quando se quer determinar uma solução periódica (Nayfeh e Balachandran, 1994).

Sabe-se que o Método de Newton-Raphson obtém sucessivas aproximações de uma raiz $f(x) = 0$, usando-se a seguinte formula de recorrência:

$$x(n) = x(n-1) - \frac{f(x_{n-1})}{f'(x_{n-1})} \text{ para } n=1, 2, \dots$$

No caso de uma equação não linear de uma variável, geometricamente o Método de Newton, consiste em se tomar um modelo local linear da função de $f(x)$ em torno de x_k , e este modelo é a reta tangente à função em x_k .

Este método necessita alguma aproximação inicial(arbitrada) para a raiz e, que a derivada de $f(x)$, exista dentro da faixa de interesse do usuário.

A partir da aproximação x_0 dada, obtém-se a tangente à curva. Essa tangente corta o eixo x , e fornece uma nova aproximação x_1 em relação à curva. Nessa nova aproximação obtém-se uma nova tangente à curva que por sua vez corta o eixo x em uma nova aproximação x_2 . Esse processo é repetido até que algum critério de convergência seja satisfeito. É fácil traduzir esse procedimento geométrico em um método numérico para achar as raízes(reais), desde que a tangente do ângulo entre o eixo x e a tangente sejam iguais $f(x_0)/(x_1 - x_0)$ e a inclinação desta própria tangente igual à derivada de $f(x)$, ou seja $f'(x_0)$, a derivada de $f(x)$ no ponto x_0 . Então tem-se a equação $f'(x_0) = f(x_0)/(x_1 - x_0)$. Então, a aproximação x_1 é dada por $x_1 = x_0 - f(x_0)/f'(x_0)$ e assim por diante. O Algoritmo do Método de Newton-Raphson é exibido, a seguir.

Seja a equação $f(x) = 0$. Tem-se que:

Passo1. Dados Iniciais: x_0 : aproximação inicial

e_1 e e_2 : precisões informadas

Passo 2. Se $|f(x_0)| < e_1$, faça $\bar{x} = x_0$. FIM.

Passo 3. $k = 1$

Passo 4. $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$

Passo 5. Se $|f(x_1)| < e_1$ ou se $|x_1 - x_0| < e_2 \Rightarrow$ faça $\bar{x} = x_1$. FIM.

Passo 6. $x_0 = x_1$

Passo 7 $k = k + 1$

Volte ao passo 4.

Observa-se que no item 8 e 9 que o algoritmo, exibido, logo acima, encontra-se programado e testado pelos autores, para solucionar algumas equações. Analisando-se com cuidado os exemplos testados, verifica-se que uma escolha cuidadosa da aproximação inicial é, em geral, essencial para o bom desempenho do Método de Newton-Raphson e se observa que quanto maior for a precisão exigida nos cálculos, exige-se mais recursos da máquina, onde estiver sendo executados os programas

5.2 - MÉTODO DOS GRADIENTES CONJUGADOS

Este método é usado para se resolver o problema de minimização de $f(x) \forall x \in \Re$ onde $f(x)$ é uma função não linear de x onde x é um enésimo componente do vetor coluna. Este é o chamado de um problema de otimização não linear. Estes problemas aparecem, em muitas aplicações, como por exemplo, em problemas de redes neurais onde uma necessidade importante é o de se achar pesos ou valores que minimizem a diferença entre a entrada e a necessária saída da rede.

O enfoque padrão para se solucionar este problema é o de se assumir uma aproximação inicial x_0 e então proceder-se a uma aproximação desenvolvida, utilizando-se uma fórmula iterativa da forma

$$x_{k+1} = x_k + s d_k \quad \text{para } k = 0, 1, 2, \dots$$

Para usar essa fórmula é necessário determinar valores para o escalar s e para o vetor d_k . O vetor d_k representa a direção da pesquisa e o escalar s determina o quão longe pode-se ir nessa direção.

Este algoritmo encontra-se programado e testado pelos autores, no item 8 e 9.

Seja a equação $f(x) = 0$. Tem-se que:

Passo 1: Informe o valor de x_0 e a tolerância e

Faça $k = 0$

Calcule a direção conjugada inicial $d^k = -g^k$

Passo 2: Se $norm(d^k) > e \rightarrow x^k$ é a solução e d^k a direção conjugada. FIM

Caso contrário: faça do Passo 3 ao Passo 7.

Passo 3: Obtenha $d^k = f'(x)$ e $dkneg = -d^k$

Passo 4: Faça do Passo 5 ao 7 em todos os valores de x , a partir da primeira coluna até terminar

Passo 5: Faça: $dkant = d^k$; $dknegant = dkneg$;

Calcule $s^k = \mathbf{m}^k / r^k$

Calcule $x^{k+1} = x^k + s_k \cdot dkneg$

Passo 5: Calcule a nova direção conjugada:

$$d^k = f'(x^{(k+1)}); A = d^k \cdot d^k; B = (dkant' \cdot dkant); Beta = A / B$$

Passo 6: Obtenha o novo valor da direção conjugada negativa:

$$dkneg = -d^k + dknegant \cdot Beta$$

Passo 7: Faça $x = x1$. Volte ao Passo 5

Passo 8: Ao terminar o laço dentro dos valores de x, volte ao Passo1.

6. RESOLUÇÃO DE SISTEMAS NÃO-LINEARES

No processo de resolução de problemas de determinação das soluções de regime permanente das equações de modulação e fase, resultantes da utilização de métodos perturbativos(**Nayfeh, 1981**), é freqüente nos depararmos com a necessidade de se obter a solução de um Sistema de Equações Não Lineares(**Burden e Faires, 1993**). Este problema pode ser formulado da seguinte forma: Dada uma função não linear $F : D \subset \Re^n \rightarrow \Re^n, F = (f_1, \dots, f_n)^T$, quer se determinar as soluções da equação algébrica não-linear $F(x) = 0$

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases}$$

ou usando a notação matricial :

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad \text{e} \quad F(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{pmatrix}$$

tem-se que cada função $f_i(x)$ é uma função não linear em $x, f_i : \Re^n \rightarrow \Re, i=1, \dots, n$, e portanto $F(x)$ é uma função não linear $x, F : \Re^n \rightarrow \Re^n$.

Ressalta-se que os métodos para resolução de sistemas não lineares, em geral são iterativos, isto é, a partir de uma condição inicial x_0 gera-se uma seqüência (x_k) de vetores que a principio, converge para a solução desejada, observadas certas condições. Note que em qualquer método iterativo é necessário estabelecer critérios de parada, para se “aceitar” um ponto x_k como aproximação para a solução exata do sistema não linear ou para se detectar a convergência ou não do processo.

A seguir, estudaremos os 5 métodos de resolução de sistemas não lineares, mais conhecidos:

- Método de Newton; Método de Quase-Newton;
- Método de Broyden;
- Método “Steepest Descent” e
- Método dos Gradientes Conjugados,

6.1 MÉTODO DE NEWTON

O método mais amplamente estudado e conhecido para resolver sistemas de equações não lineares é o Método de Newton.

No caso de uma equação não linear a uma variável, viu-se anteriormente(**ITEM 5.1**) que, geometricamente, esse método consiste em se tomar um modelo local linear da função $f(x)$ em torno de x_k , sendo que este modelo é a reta tangente à função em x_k . Ampliando-se a motivação de se construir um modelo local linear para o caso de um sistema de equações não lineares, tem-se:

Conhecida a aproximação $x^{(k)} \in D$, e para qualquer $x \in D$, existe $c_i \in D$, tal que:

$$f_i(x) = f_i(x^{(k)}) + \nabla f_i(c_i)^T (x - x^{(k)}) \quad i = 1, \dots, n$$

e aproximando-se $\nabla f_i(c_i)$ por $\nabla f_i(x^{(k)})$, $i = 1, \dots, n$ tem-se um modelo local linear para $f_i(x)$ em torno de $x^{(k)}$:

$$f_i(x) \approx f_i(x^{(k)}) + \nabla f_i(x^{(k)})^T (x - x^{(k)}), i = 1, \dots, n$$

E, portanto, o modelo local linear para $F(x)$ em torno de $x^{(k)}$ será:

$$F(x) \approx L_k(x) = F(x^{(k)}) + J(x^{(k)})(x - x^{(k)})$$

Note que a nova aproximação $x^{(k+1)}$ será o zero do modelo local linear $L_k(x)$, desta forma tem-se

$$L_k(x) = 0 \Leftrightarrow J(x^{(k)})(x - x^{(k)}) = -F(x^{(k)})$$

, denotando-se $(x - x^{(k)})$ por $s^{(k)}$ tem-se que $x^{(k+1)} = x^{(k)} + s^{(k)}$, onde $s^{(k)}$ (passo de Newton) é solução do sistema linear:

$$J(x^{(k)})s = -F(x^{(k)})$$

onde $J(x)$ é a Matriz Jacobiana em $x^{(k)}$. Observa-se que uma iteração de Newton consiste basicamente de: avaliação da matriz Jacobiana em $x^{(k)}$, i.e., da resolução do sistema linear $J(x^{(k)})s = -F(x^{(k)})$.

O Algoritmo, encontra-se programado e testado pelos autores no item 8 e 9.

Dados x_0 , $e_1 > 0$ e $e_2 > 0$, faça-se:

Passo 1: Calcula-se $F(x^{(k)})$ e $J(x^{(k)})$;

Passo 2: se $\|F(x^{(k)})\| < e_1$, faça $\bar{x} = x^{(k)}$ e FIM;

Caso contrário:

Passo 3: obtém-se $s^{(k)}$, solução do sistema linear: $J(x^{(k)})s = -F(x^{(k)})$;

Passo 4: faça: $x^{(k+1)} = x^{(k)} + s^{(k)}$;

Passo 5: se $\|x^{(k+1)} - x^{(k)}\| < e_2$, faça $\bar{x} = x^{(k+1)}$ e FIM;

Caso contrário:

Passo 6: $k = k + 1$;

Volta-se ao passo 1

6.2 MÉTODO DE QUASE NEWTON(OU DE NEWTON MODIFICADO)

Uma fragilidade (significante) do Método de Newton está no fato de que, a cada iteração: uma matriz Jacobiana precisa ser calculada, sendo necessário resolver um sistema linear envolvendo esta matriz.

Uma modificação do Método de Newton, bastante usada nas aplicações, consiste em se tomar a cada iteração k , a matriz $J(x^{(0)})$, em vez de $J(x^{(k)})$: a partir de uma aproximação inicial $x^{(0)}$, a seqüência $\{x^{(k)}\}$ é gerada através de $x^{(k+1)} = x^{(k)} + s^{(k)}$, onde $s^{(k)}$ é a solução do sistema linear:

$$J(x^{(0)})s = -F(x^{(k)})$$

Desta forma a Matriz Jacobiana é avaliada apenas uma vez, e para todo k , o sistema linear a ser resolvido a cada iteração terá a mesma matriz de coeficientes: $J(x^{(0)})$.

A seguir, exhibe-se o algoritmo, deste método, cujo programa foi testado pelos autores, no item 8 e 9.

Dados x_0 , $e_1 > 0$ e $e_2 > 0$, **faça:**

Passo 1: Calcula-se $J(x^{(0)})$

Passo 2: Calcula-se $F(x^{(k)})$

Passo 3: se $\|F(x^{(k)})\| < e_1$, **faz-se** $\bar{x} = x^{(k)}$ e **FIM**;

Caso contrário:

Passo 4: obtém-se $s^{(k)}$, **solução do sistema linear:** $J(x^{(0)})s = -F(x^{(k)})$;

Passo 5: **faz-se:** $x^{(k+1)} = x^{(k)} + s^{(k)}$;

Passo 6: se $\|x^{(k+1)} - x^{(k)}\| < e_2$, **faz-se** $\bar{x} = x^{(k+1)}$ e **FIM**;

Caso contrário:

Passo 7: $k = k + 1$;

Volta-se ao passo 2.

6.3 MÉTODO DE BROYDEN

O método de Newton, descrito anteriormente não provê um modo prático de se solucionar qualquer sistema de equações não lineares, mas apenas para sistemas menores, i.e., o método exige que o usuário providencie não somente as funções definições, como também as definições das n^2 derivadas parciais das funções. Então, para um sistema de 10 incógnitas, o usuário deve providenciar 110 funções definições.

Além domais, como já foi observado anteriormente, os métodos Quasi-Newton evitam o cálculo das derivadas parciais obtendo as aproximações à elas envolvendo apenas as funções valores. Através da utilização dos Métodos Quasi-Newton se obtém sucessivas aproximações satisfatórias à matriz Jacobiana a cada nova iteração. Broyden e outros têm mostrado que sob certas circunstâncias estas fórmulas fornecem aproximações satisfatórias à inversa Jacobiana. A aproximação inicial para a inversa Jacobiana é usualmente tomada como um escalar múltiplo de uma matriz.

O sucesso do algoritmo depende da natureza das funções a serem resolvidas e das aproximações iniciais da solução. A aproximação inicial para a inversa jacobiana é usualmente tomada como um escalar múltiplo de uma matriz. O sucesso do algoritmo depende da natureza das funções a serem resolvidas e das aproximações iniciais informadas. O programa do método de Broyden cujo algoritmo está programado e testado pelos autores está listado no, no item 8 e 9.

Passo 1: Obtém-se uma aproximação inicial à solução.

Faz-se o contador $r = \text{zero}$.

Passo 2: Calcula-se a Jacobiana da aproximação inicial. B^r

Passo 3: Calcula-se $p^r = -B^r f^r$ onde $f^r = f(x^r)$

Passo 4: Determina-se r de tal modo que $\|f(x^r + t_r p^r)\| < \|f^r\|$ onde os símbolos $\| \|$ denotam que a norma do vetor deve ser obtida.

Passo 5: Calcula-se $x^{r+1} = x^r + t_r p^r$

Passo 6: Calcula-se $f^{r+1} = f(x^{r+1})$.

Se $\|f^{r+1}\| < e$ **onde** e **é a tolerância informada, FIM.**

Passo 7: Usa-se a fórmula abaixo para obter a necessária aproximação à matriz Jacobiana:

$$B^{r+1} = B^r - (B^r y^r - p^r)(p^r)^T B^r / \{(p^r)^T B^r y^r\} \text{ onde } y^r = f^{r+1} - f^r$$

Passo 8: Faz-se a $i = i + 1$ e volte ao passo 3.

6.4 MÉTODO “STEEPEST DESCENT”

A vantagem dos métodos Newton e Quase Newton para solucionar sistemas de equações não lineares é a sua velocidade de convergência, uma vez que uma acurada aproximação inicial pode ser sempre fornecida. A fragilidade desses métodos é que uma acurada aproximação inicial é necessária para garantir a convergência dos métodos.

O Método “Steepest Descent”, considerado, neste texto, i converge linearmente para a solução, mas vai sempre convergir, mesmo para uma aproximação inicial não muito acurada. Como consequência, este método é usado para achar aproximações iniciais suficientemente acuradas para as técnicas baseadas em Newton.

Este método determina um mínimo local para uma função multivariável da forma: $g : \Re^n \rightarrow \Re$. A conexão entre a minimização da função de \Re^n para \Re e a solução de um sistema de equações não lineares é esperado pelo fato de que um sistema da forma :

$$\begin{aligned}f_1(x_1, x_2, \dots, x_n) &= 0 \\f_2(x_1, x_2, \dots, x_n) &= 0 \\&\vdots \\f_n(x_1, x_2, \dots, x_n) &= 0\end{aligned}$$

tem uma solução em $x = (x_1, x_2, \dots, x_n)$ precisamente quando a função g definida por :

$$g(x_1, x_2, \dots, x_n) = \sum_{i=1}^n [f_i(x_1, x_2, \dots, x_n)]^2$$

tem o valor mínimo zero. O algoritmo desse método busca uma solução p para o problema de minimização:

$$g(p) = \min_{x \in \Re^n} g(x)$$

dada uma aproximação inicial x .

O algoritmo encontra-se programado e testado pelos autores no , no item 8 e 9.

Entrada: n variáveis; **aproximação inicial:** $x = (x_1, \dots, x_n)$; **tolerância** TOL ; **número máximo de iterações** N .

Saída: solução $x = (x_1, \dots, x_n)^t$ ou uma mensagem de erro.

Passo 1: Faz-se $k = 0$

Passo 2: Enquanto $(k \leq N)$ faz-se do passo 3 ao 15.

Passo 3: Faz-se : $g_1 = g(x_1, \dots, x_n)$; (Note-se que: $g_1 = g(x^{(k)})$).

$z = \nabla g(x_1, \dots, x_n)$; (Note-se que: $z = \nabla g(x^{(k)})$).

$$z_0 = \|z\|_2$$

Passo 4: se $z_0 = 0$ então FIM – mostrando que “Pode haver um mínimo”.

Passo 5: Faz-se $z = z / z_0$;

$$a_3 = 1$$

$$g_3 = g(x - a_3 z)$$

Passo 6: Enquanto $(g_3 \geq g_1)$ faz-se os passos 7 e 8.

Passo 7: Faz-se $a_3 = a_3 / 2$;

$$g_3 = g(x - a_3 z)$$

Passo 8: Se $\mathbf{a}_3 < TOL/2$ então FIM – mostrando que “Pode haver um mínimo”.

Passo 9: Faz-se $\mathbf{a}_2 = \mathbf{a}_3 / 2$;

$$g_2 = g(x - \mathbf{a}_2 z).$$

Passo 10: Faz-se $h_1 = (g_2 - g_1) / \mathbf{a}_2$;

$$h_2 = (g_3 - g_2) / (\mathbf{a}_3 - \mathbf{a}_2);$$

$$h_3 = (h_2 - h_1) / \mathbf{a}_3$$

Passo 11: Faz-se $\mathbf{a}_0 = 0.5(\mathbf{a}_2 - h_1 / h_3)$;

$$g_0 = g(x - \mathbf{a}_0 z)$$

Passo 12: Acha-se \mathbf{a} de $\{\mathbf{a}_0, \mathbf{a}_3\}$ tal que

$$g = g(x - \mathbf{a}z) = \min \{g_0, g_3\}$$

Passo 13: Faz-se $x = x - \mathbf{a}z$

Passo 14: Se $|g - g_1| < TOL$ então FIM – mostrando “Fim Normal”.

Passo 15: Faz-se $k = k + 1$

Passo 16: FIM – mostrando “Numero máximo de iterações foi excedido”.

6.5 MÉTODO DOS GRADIENTES CONJUGADOS

O Método dos Gradientes Conjugados é um método poderoso para solucionar sistemas de equações, e mantêm-se bem próximo do algoritmo descrito item “Zeros Reais de Funções Reais” para solução de problemas de otimização não linear. Aqui, entretanto, a pesquisa é simplificada e o valor do gradiente pode ser calculado dentro do algoritmo.

Pode-se então considerar outra aplicação desse algoritmo, desta vez para a solução de sistemas de equações que têm uma forma especial. Aplica-se o algoritmo para otimizar uma função quadrática positiva que possui a forma padrão:

$$f(x) = (x^T)Ax) / 2 = p^T x + q \quad (*)$$

onde \mathbf{x} e \mathbf{p} são n componentes vetores colunas. \mathbf{A} é uma matriz positiva simétrica $n \times n$ e q é um escalar. O valor mínimo de $f(x)$ será tal que o gradiente de $f(x)$ é zero. Entretanto o gradiente é facilmente encontrado pela diferenciação direta:

$$\nabla f(x) = Ax + p = 0$$

Para se encontrar o valor mínimo é equivalente a se solucionar o sistema de equações que se inicia fazendo-se:

$$Ax = b \quad (**)$$

Desde que se possa usar o método dos gradientes conjugados para achar o mínimo de (*), então pode-se usa-lo para solucionar o sistema de equações (**).

O algoritmo, exibido, a seguir, encontra-se programado e testado pelos autores no , no item 8 e 9.

Passo 1: Informa-se o valor de x_0 e a tolerância ϵ

$$\text{Faz-se } k = 0: x^k = 0, g^k = b, m^k = b^T b$$

$$\text{Calcula-se a direção conjugada inicial } d^k = -g^k$$

Passo 2: Se $\text{norm}(d^k) > \epsilon \rightarrow x^k$ é a solução e d^k a direção conjugada. FIM

Caso contrário: faz-se do Passo 3 ao Passo 7.

Passo 3: Obtém-se $d^k = f'(x)$ e $dkneg = -d^k$

Passo 4: Faz-se do Passo 5 ao 7 em todos os valores de x , a partir da primeira coluna até terminar

Passo 5: Faz-se: $dkant = d^k$; $dknegant = dkneg$;

$$q^k = Ad^k, r^k = (d^k)^T q^k$$

$$\text{Calcula-se } s^k = \mathbf{m}^k / r^k$$

$$\text{Calcula-se } x^{k+1} = x^k + s^{ki} dkneg$$

Passo 5: Calcula-se a nova direção conjugada:

$$d^k = f'(x^{(k+1)}); A = d^k * d^k; B = (dkant * dkant); Beta = A / B$$

Passo 6: Obtém-se o novo valor da direção conjugada negativa:

$$dkneg = -d^k + dknegant * Beta$$

Passo 7: Faz-se $x = x1$. Volte ao Passo 5

7. ANÁLISE DA PERFORMANCE DOS MÉTODOS PARA A EQUAÇÕES E SISTEMAS DE EQUAÇÕES NÃO-LINEARES

Para que se possa entender os resultados obtidos, utilizando-se os diferentes métodos, expostos, neste trabalho, programou-se todos os métodos através de seus respectivos algoritmos, e submeteu-se todos eles à vários grupos de sistema de equações, com diversas aproximações iniciais. Toda a programação foi feita pelos autores, utilizando-se do Software MATLAB®(ver item 8 e 9) onde adotou-se a seguinte notação:

- Método de Newton – *newton*
- Método Quase Newton - *quasnew*
- Método de Broyden – *broyden*
- Método Steepest Descent – *steepest*
- Método dos Gradientes Conjugados – *gradconj*

Os sistemas de equações foram programados de tal forma que função com suas respectivas derivadas estão dispostas na forma seguinte:

- *f308 e f309*;
- *f310 e f311*;
- *f312, f313 e*
- *f314 e f315*.

Finalmente, Todas as funções criadas no Matlab para testar os 5 métodos:

- *tnewton* – para testar o Método de Newton
- *tquasnew* – para testar o Método Quase Newton
- *tbroyden* – para testar o Método de Broyden
- *tsteepest* – para testar o Método Steepest Descent.
- *tgradconj* – Método dos Gradientes Conjugados.

Para facilidade do leitor, analisou-se 5 planilhas, onde se realizaram vários testes, onde obtiveram vários resultados. Colocou-se em **negrito** as primeiras ocorrências dos testes que satisfazem o sistema de equações. Para cada teste , mostrou-se:

- Número de iterações

- As raízes encontradas, listadas logo abaixo do numero de iterações. Além da verificação dos resultados do sistema de equações (Validação), utilizando-se as raízes encontradas.

7.1 MÉTODO NEWTON-RAPHSON PARA UMA EQUAÇÃO NÃO LINEAR

Através dos testes executados, e baseado nos exemplos expostos, a seguir; podemos obter a comprovação de que uma escolha cuidadosa da aproximação inicial é essencial para o bom desempenho do Método de Newton-Raphson. Gostaríamos, também de observar que quanto maior for a precisão exigida nos cálculos, mais recursos estaremos exigindo da máquina onde estiver sendo executados os testes.

Seja inicialmente os casos particulares onde se deseja calcular a raiz de uma equação escalar ou vetorial :

Teste 1: $f(x) = x^6 + x^2 + x - 6$ e $x_0 = 1.5$, utilizando-se o *pmnewra.m* (programa em MATLAB

pmnewra611.m

pmnewra6(1,0,0,0,1,1,-6,1.5,0.0,0.000005)
PMNEWRA6-Pgma do Metodo Newton-Raphson -
para equacoes do 2O. ATE O 6O.Grau
Variaveis da equação: (1, 0, 0, 0,1, 1, -6)

-1.33098210214630
-0.71608301032157 + 1.20843466527941i
-0.71608301032157 - 1.20843466527941i
0.77171922753417 + 1.13031571627189i
0.77171922753417 - 1.13031571627189i
1.21970966772111

aproximacao inicial informada (1.5)

Achamos o valor aproximado com 4 iteracoes

Precisoas Informadas:

a1=0.00000000000000000000

a2=0.00000500000000000000

Valores encontrados para x, a partir do valor inicial:

wxk=1.315573770491803400

wxk=1.233792610663979200

wxk=1.220048534195941100

wxk=1.219709867615793100

Valor aprox.de x ->

xaprox=1.219709667721178900

ESTA E´ UMA EQUACAO DO 6o.GRAU cujas
raizes sao:

raizes =

COMENTÁRIOS:

Veja que: precisão

ponto inicial

Tivemos:

até a 5a. Casa decimal
próximo de uma das
raízes

4 iterações

Teste 2: $f(x) = x^3 - 10x^2 + 29x - 20$ e $x_0 = 7.0$, utilizando-se o *fnewton.m* - o programa em Matlab e veja a figura 7.1 -1

f302.m

function F=f302(x);

F=x.^3 - 10.0*x.^2 + 29.0*x - 20.0;

F=3*x.^2 - 20*x + 29;

fnewton01.m

[x,it]=fnewton('f302','f303',7,0.00005)

f303.m

function F=f303(x);

x0 = 7

f = 36
d = 1
....
d = 5.6373e-008

f = 2.2549e-007
x = 5.0000
iterações = 6

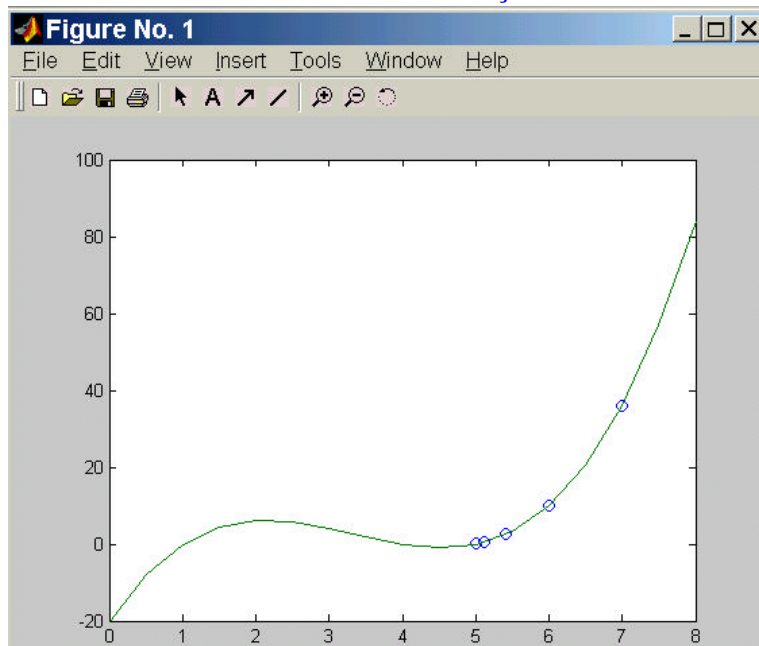


Figura 7.1 - 1 Gráfico da função $f(x) = x^3 - 10x^2 + 29x - 20$ com as iterações do Método de Newton mostrada com um “o”.

Teste 3: $f(x_1, x_2) = (x_1^4 - 16x_1^2 + 5x_1)/2 + (x_2^4 - 16x_2^2 + 5x_2)/2$ e $x_0 = [0,1]$ *utilizando-se o programa mincg.m (Método dos Gradientes Conjugados para uma equação não linear). Para visualizar o gráfico dessa função, criou-se o seguinte programa: fmeshmincg.m . Expoem-se na Fig. 7.1- 2 Gráfico tri-dimensional da função f801. Figura obtida através da execução do Método dos Gradientes Conjugados para uma função não linear. O programa que foi executado é o mincg.e na Fig 7.1 –3 exibe-se o Gráfico de contorno da função f801 , mostrando a localização dos quatro locais mínimos. O caminho de pesquisa do algoritmo também é mostrado.*

f801.m

```
function fv=f801(x);
fv=0.5*( x(1).^4 - 16*x(1).^2 + 5*x(1) ) + 0.5*(
x(2).^4 - 16*x(2).^2 + 5*x(2));
```

f801pd.m

```
function pd=f801pd(x);
pd=zeros(size(x));
pd(1)=4*x(1).^3 - 32*x(1)+5;
pd(2)=4*x(2).^3 - 32*x(2)+5;
```

Para executar utilizou-se deste programa:

tmincg.m

```
global p1 dkNeg;
x0=[0 1];
x1=Mincg('f801','f801pd','ftau2cg',x0,0.000005);
```

com o seguinte resultado:

tmincg01.m

```
res = -2.9035 -2.9035
```

Solution -2.9035 -2.9035

Gradient 1.0e-005 *
-0.0830 -0.1404

x1 = -2.9035 -2.9035

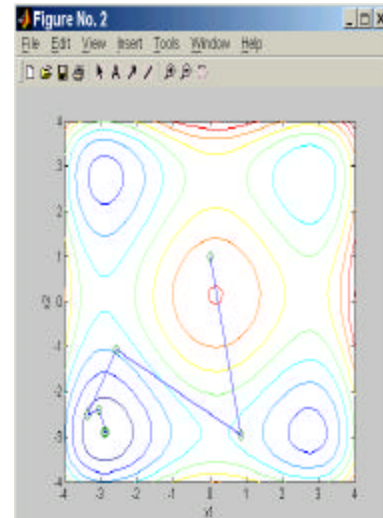
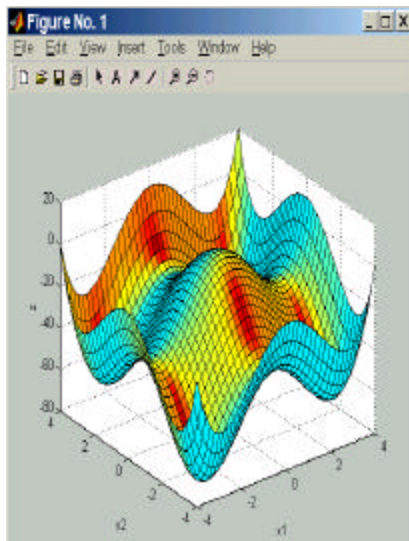


Fig. 7.1- 2 Gráfico tri-dimensional da função f801. Figura obtida através da execução do Método dos Gradientes Conjugados para uma função não linear. O programa que foi executado é o mincg.e na Fig 7.1 –3 exibe-se o Gráfico de contorno da função f801 , mostrando a localização dos quatro locais mínimos. O caminho de pesquisa do algoritmo também é mostrado.

Teste 4. Utilização de *mnewton.m* – programa em Matlab para a solução do sistema de equações

$$\begin{cases} x^2 + y^2 = 4 \\ xy = 1 \end{cases}$$

newton01.m 1.9319
 Newton-Pgma do Metodo Newton - sistemas nao lineares - 0.5176
 Total de iteracoes: 4 Validação :
 Valores encontrados para x, a partir do valor inicial: 1.0e-003 *
 Tolerancia Informada:
 0.0050 0.1207
 -0.0603
 Raizes do sistema :

Teste 5. Utilização de *quasnew.m* – programa em Matlab para a solução do sistema de equações

$$\begin{cases} x^2 + y^2 = 4 \\ xy = 1 \end{cases}$$

quasnew01.m
 quasnew-Pgma Metodo Quase Newton - sist.nao lineares Raizes do sistema :
 1.9361
 0.5147
 Total de iteracoes: 12
 Valores encontrados para x, a partir do valor inicial: Validação :
 Tolerancia Informada: 0.0134
 0.0050 -0.0035

Teste 6. Utilização de broyden.m – programa em Matlab para a solução do sistema de equações

$$\begin{cases} x^2 + y^2 = 4 \\ xy = 1 \end{cases}$$

broyden01.m

```
broyden-Pgma Metodo Broyden - sist.nao lineares
ans =
0.5176
1.9319
Total de iteracoes: 36
```

Teste 7. Utilização de steepest.m - programa em Matlab para a solução do sistema de equações

$$\begin{cases} x^2 + y^2 = 4 \\ xy = 1 \end{cases}$$

```
steepest01.m
Steepest-Pgma Metodo Steepest Descent - sist.nao
lineares
ans =
0.5178
1.9318
Total de iteracoes: 7
Solution
1.9318
```

Teste 8. Método Gradientes Conjugados gradconj.m para o sistema de equações:

$$\begin{cases} x^2 + y^2 = 4 \\ xy = 1 \end{cases}$$

Para testar o método, foi criado o Arquivo de Lote abaixo que gera um sistema de 10 equações com seleção aleatória de elementos

```
tsolvercg.m
n=10
tol=1e-8
A = 10 * rand(n)
b = 10 * rand(n,1)
ada = A * A'
% To ensure a symmetric positive definitive matrix
O resultado que se obteve pode ser visto abaixo:
```

```
sol = gradconj(ada,b,n,tol)
disp('Solution of system is : ')
disp(sol)
accuracy = norm(ada * sol - b)
fprintf('Norm of residuals =% 12.9f\n',accuracy)
```

Solution of system is :

```
0.7201      -15.076      0.5631      10.171
0.1330      31.510      12.167      accuracy 3,43E-
      -31.330      -34.515      =      04
      17.036      Norm of residuals =
0.0000000034
```


Teste 9: Quadro comparativo para o sistema não-linear

$$\begin{cases} x^2 + y^2 = 4 \\ xy = 1 \end{cases}$$

f308 f309	([3;-1.5], 'f308', 'f309', 2, tol)				
Tolerancia	0.05	0.00005	0.0000005	0.000000005	0.00000000005
NEWTON Iterações raizes	03 Validação 1.9397 0.0223 0.5099 -0.0110	04 Validação 1.9319 1.0e-008 * 0.5176 0.3640 -0.1820	05 Validação 1.9319 1.0e-008 * 0.5176 0.3640 -0.1820	05 Validação 1.9319 1.0e-015 * 0.5176 0 -0.1110	06 Validação 1.9319 1.0e-015 * 0.5176 0 -0.1110
QUASNEW Iterações raizes	07 Validação 1.9776 0.1532 0.4922 -0.0266	24 Validação 1.9319 1.0e-003 * 0.5176 0.1380 -0.0901	31 Validação 1.9319 1.0e-004 * 0.5176 0.1220 -0.0524	31 Validação 1.9319 1.0e-004 * 0.5176 0.1220 -0.0524	31 Validação 1.9319 1.0e-004 * 0.5176 0.1220 -0.0524
BROYDEN Iterações raizes	34 Validação 0.5250 -0.0207 1.9245 0.0104	36 Validação 0.5176 1.0e-005 * 1.9319 0.5035 -0.2518	37 Validação 0.5176 1.0e-007 * 1.9319 0.4816 -0.2414	38 Validação 0.5176 1.0e-011 * 1.9319 -0.2277 0.1141	39 Validação 0.5176 1.0e-011 * 1.9319 -0.2277 0.1141
STEEPEST Iterações raizes	06 Validação 1.9303 -0.0065 0.5172 -0.0017	07 Validação 1.9318 1.0e-003 * 0.5178 -0.0676 0.2301	08 Validação 1.9318 1.0e-005 * 0.5176 -0.9873 -0.2445	09 Validação 1.9319 1.0e-007 * 0.5176 -0.1818 -0.0450	10 Validação 1.9319 1.0e-007 * 0.5176 -0.1818 -0.0450
GRADCONJ Iterações raizes	02 Validação 1.7991 -0.6792 0.2898 -0.4785	06 Validação 1.9317 -0.0064 0.5120 -0.0110	08 Validação 1.9318 -0.0011 0.5167 -0.0019	10 Validação 1.9318 1.0e-004 * 0.5175 -0.3173 -0.5479	12 Validação 1.9319 1.0e-004 * 0.5176 -0.3173 -0.5479

Teste 10: Quadro comparativo para o sistema não-linear

$$\begin{cases} \sin(x) + y^2 + \log_e z = 7 \\ 3x + 2y - z = -1 \\ x + y + z = 5 \end{cases}$$

f310 f311		([0;2;2], 'f310', 'f311', 3, tol)			
Tolerancia	0.05	0.00005	0.0000005	0.000000005	0.00000000005
NEWTON Iterações raízes	01 Validação 0.5988 0.0013 2.3962 0.0011 2.0050 0	03 Validação 0.5991 1.0e-006 * 2.3959 0.0783 2.0050 0.1014 0	03 Validação 0.5991 1.0e-006 * 2.3959 0.0783 2.0050 0.1014 0	04 Validação 0.5991 1.0e-014 * 2.3959 -0.0888 2.0050 -0.1776 0	04 Validação 0.5991 1.0e-014 * 2.3959 -0.0888 2.0050 -0.1776 0
QUASNEW Iterações raízes	02 Validação 0.6083 -0.0473 2.3842 -0.0459 2.0076 0	07 Validação 0.5991 1.0e-004 * 2.3959 -0.3366 2.0050 -0.3219 0	11 Validação 0.5991 1.0e-006 * 2.3959 -0.2693 2.0050 -0.2575 0	15 Validação 0.5991 1.0e-008 * 2.3959 -0.2155 2.0050 -0.2061 0	19 Validação 0.5991 1.0e-010 * 2.3959 -0.1725 2.0050 -0.1649 0
BROYDEN Iterações raízes	14 Validação 0.5990 1.0e-003 * 2.3960 0.2103 2.0050 0.4736 0.0030	16 Validação 0.5991 1.0e-005 * 2.3959 -0.2867 2.0050 -0.5105 0.0068	19 Validação 0.5991 1.0e-008 * 2.3959 0.1112 2.0050 0.3864 0.0011	19 Validação 0.5991 1.0e-008 * 2.3959 0.1112 2.0050 0.3864 0.0011	20 Validação 0.5991 1.0e-011 * 2.3959 -0.0306 2.0050 -0.1059 -0.0003
STEEPEST Iterações raízes	06 Validação 0.1923 0.0215 2.4837 -0.1038 1.9376 -0.3863	166 Validação 0.5544 0.0118 2.4069 -0.0075 1.9979 -0.0408	342 Validação 0.5946 0.0012 2.3970 -0.0007 2.0043 -0.0041	518 Validação 0.5986 1.0e-003 * 2.3960 0.1183 2.0049 -0.0747 -0.4086	694 Validação 0.5990 1.0e-004 * 2.3959 0.1190 2.0050 -0.0751 -0.4111
GRADCONJ Iterações raízes	O sistema não se mostrou estável	O sistema não se mostrou estável	O sistema não se mostrou estável	O sistema não se mostrou estável	O sistema não se mostrou estável

Teste 11: Quadro comparativo para o sistema não-linear

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - 1/2 = 0 \\ x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.06 = 0e^{-x_1 x_2} + 20x_3 + \frac{10p-3}{3} = 0 \\ e^{-x_1 x_2} + 20x_3 + \frac{10p-3}{3} = 0 \end{cases}$$

F312 F313	([0.1;0.1;-0.1], 'f312', 'f313', 3, tol)									
Tolerancia	0.05	0.00005	0.0000005	0.000000005	0.0000000005					
NEWTON Iterações raizes	01 Validação 0.5000 0.0000 0.0016 -0.0259 -0.5236 0.0000	03 Validação 0.5000 1.0e-007 * 0.0000 0.0002 -0.5236 -0.1254 0.0002	04 Validação 0.5000 1.0e-007 * 0.0000 0.0002 -0.5236 -0.1254 0.0002	04 Validação 0.5000 1.0e-014 * -0.0000 0 -0.5236 0 -0.1776	05 Validação 0.5000 1.0e-014 * -0.0000 0 -0.5236 0 -0.1776					
QUASNEW Iterações raizes	01 Validação 0.5000 -0.0000 0.0088 -0.1483 -0.5232 0.0043	09 Validação 0.5000 1.0e-003 * 0.0000 0.0000 -0.5236 -0.5131 0.0127	15 Validação 0.5000 1.0e-005 * 0.0000 0.0000 -0.5236 -0.8063 0.0200	22 Validação 0.5000 1.0e-007 * 0.0000 0.0000 -0.5236 -0.6346 0.0157	29 Validação 0.5000 1.0e-009 * 0.0000 0.0000 -0.5236 -0.4994 0.0124					
BROYDEN Iterações raizes	20 Validação 0.5000 0.0000 -0.0004 0.0071 -0.5236 0.0000	24 Validação 0.5000 1.0e-004 * 0.0000 0.0000 -0.5236 0.2112 0.0000	26 Validação 0.5000 1.0e-007 * -0.0000 -0.0000 -0.5236 0.4405 0.0000	27 Validação 0.5000 1.0e-010 * -0.0000 -0.0000 -0.5236 0.4575 0.0000	27 Validação 0.5000 1.0e-010 * -0.0000 -0.0000 -0.5236 0.4575 0.0000					
STEEPEST Iterações raizes	12 Validação 0.2642 -0.7074 -0.0119 -0.0029 -0.5280 -0.0844	87 Validação 0.4929 -0.0214 -0.0005 0.0009 -0.5234 0.0039	136 Validação 0.4993 -0.0022 -0.0000 -0.0000 -0.5236 -0.0003	186 Validação 0.4999 1.0e-003 * -0.0000 -0.2186 -0.5236 -0.0021 -0.0258	236 Validação 0.5000 1.0e-004 * -0.0000 -0.2150 -0.5236 -0.0020 -0.0254					
GRADCONJ Iterações raizes	06 Validação 0.2975 -0.6075 0.0269 -0.7073 -0.5852 -1.2402	23 Validação 0.4925 -0.0170 -0.1988 0.0091 -0.5271 0.0336	29 Validação 0.4977 -0.0014 -0.1995 0.0007 -0.5287 0.0028	33 Validação 0.4981 1.0e-003 * -0.1996 -0.2741 -0.5288 0.1441 0.5412	39 Validação 0.4981 1.0e-004 * -0.1996 -0.2302 -0.5288 0.1211 0.4546					

Obs: obteve a mesma
raiz que Broyden
c/aprox.inicial 0,0,0

Teste 12: Quadro comparativo para o sistema não-linear

$$\begin{cases} 3x_1 - \cos(x_2 x_3) - 1/2 = 0 \\ x_1^2 - 81(x_2 + 0.1)^2 + \sin(x_3) + 1.06 = 0e^{-x_1 x_2} + 20x_3 + \frac{10p-3}{3} = 0 \\ e^{-x_1 x_2} + 20x_3 + \frac{10p-3}{3} = 0 \end{cases}$$

F312 F313	([0;0;0], 'f312', 'f313', 3, tol)									
Tolerancia	0.05		0.00005		0.0000005		0.000000005		0.00000000005	
NEWTON Iterações raizes	01 Validação 0.5000 0.0000 0.0017 -0.0280 -0.5236 0.0000		03 Validação 0.5000 1.0e-007 * 0.0000 0.0003 -0.5236 -0.1719 0.0002		04 Validação 0.5000 1.0e-007 * 0.0000 0.0003 -0.5236 -0.1719 0.0002		04 Validação 0.5000 1.0e-014 * 0.0000 -0.0222 -0.5236 -0.0222 -0.1776		05 Validação 0.5000 1.0e-014 * 0.0000 -0.0222 -0.5236 -0.0222 -0.1776	
QUASNEW Iterações raizes	02 Validação 0.5000 -0.0000 -0.0015 0.0230 -0.5240 -0.0078		03 Validação 0.5000 1.0e-005 * 0.0000 -0.0000 -0.5236 -0.4728 -0.5181		04 Validação 0.5000 1.0e-006 0.0000 -0.0000 -0.5236 -0.0347 0.1379		06 Validação 0.5000 1.0e-008 * 0 0.0924 0.1284		07 Validação 0.5000 1.0e-010 * -0.0000 0 0.0860 -0.2654	
BROYDEN Iterações raizes	17 Validação 0.4981 0.0000 -0.1998 -0.0032 -0.5288 0.0000		19 Validação 0.4981 1.0e-005 * -0.1996 -0.0013 -0.5288 0.7011 0.0081		20 Validação 0.4981 1.0e-007 * -0.1996 0.0005 -0.5288 -0.5537 -0.0061		21 Validação 0.4981 1.0e-008 * -0.1996 0.0010 -0.5288 -0.3921 -0.0045		23 Validação 0.4981 1.0e-012 * -0.1996 0.0008 -0.5288 -0.3075 -0.0018	
STEEPEST Iterações raizes	07 Validação 0.3243 -0.5272 -0.0085 -0.0168 -0.5284 -0.0939		76 Validação 0.4926 -0.0222 -0.0005 0.0007 -0.5235 0.0027		126 Validação 0.4993 -0.0022 -0.0000 0.0001 -0.5236 0.0003		176 Validação 0.4999 1.0e-003 * -0.0000 -0.2251 -0.5236 0.0076 0.0270		227 Validação 0.5000 1.0e-004 * -0.0000 -0.2147 -0.5236 -0.0044 -0.0383	
GRADCONJ Iterações raizes	02 Validação 0.4605 -0.1184 0.0051 -0.0837 -0.4784 0.9012		09 Validação 0.4985 0.0009 -0.1989 0.0113 -0.5288 0.0003		11 Validação 0.4983 1.0e-003 * -0.1996 0.3465 -0.5289 0.0247 -0.8638		12 Validação 0.4982 1.0e-003 * -0.1996 0.2864 -0.5288 0.0248 0.1156		16 Validação 0.4982 1.0e-004 * -0.1996 0.4311 -0.5288 0.0133 0.1842	

Obs: = Broyden

Teste 13: Quadro comparativo para o sistema não-linear (os valores dos parâmetros foram obtidos da pagina 149 (Nayfeh , 2000))

$$\begin{cases} m\ddot{a}_1 + (A_1 a_1 a_2 \sin(\mathbf{g}_1) + g a_1 \cos(\mathbf{g}_2)) = 0 \\ m\ddot{a}_2 - A_2 a_1^2 \sin(\mathbf{g}_1) = 0 \\ v_1 \dot{a}_1 - A_1 a_1 a_2 \cos(\mathbf{g}_1) - g a_1 \cos(\mathbf{g}_2) = 0 \\ v_2 \dot{a}_2 - A_2 a_1^2 \cos(\mathbf{g}_1) = 0 \end{cases}$$

F314 F315	([0;0;0], 'f314', 'f315', 4, tol)									
Tolerancia	0.05	0.00005	0.0000005	0.000000005	0.0000000005					
NEWTON Iterações raizes	0 Validação 0.1000 0.0298 0.1000 0.0158 1.0000 0.0082 1.0000 0.0073									
QUASNEW Iterações raizes	0 Validação 0.1000 0.0298 0.1000 0.0158 1.0000 0.0082 1.0000 0.0073									
BROYDEN Iterações raizes	Validação 0 0.1000 0.0298 0.1000 0.0158 1.0000 0.0082 1.0000 0.0073	Validação 07 1.0e-005 * 0.0000 0.2111 -0.0000 -0.4592 0.9721 0.1334 0.9519 -0.2296	Validação 09 1.0e-006 * -0.0000 -0.1424 0.0000 0.2423 0.9721 -0.0900 0.9519 0.1212	Validação 10 1.0e-008 * -0.0000 -0.1150 0.0000 0.1948 0.9721 -0.0726 0.9519 0.0974	Validação 11 1.0e-011 * -0.0000 -0.4969 0.0000 0.8444 0.9721 -0.3139 0.9519 0.4222					
STEEPEST Iterações raizes	01 Validação -0.0122 -0.0033 0.0735 0.0147 0.7637 -0.0009 0.3813 0.0073	02 Validação -0.0008 -0.0002 0.0739 0.0148 0.8510 -0.0001 0.4519 0.0074	03 Validação -0.0000 -0.0000 0.0739 0.0148 0.8556 -0.0000 0.4564 0.0074	03 Validação -0.0121 -0.0033 0.0735 0.0147 0.7642 -0.0009 0.3817 0.0073	02 Validação -0.0122 -0.0033 0.0735 0.0147 0.7637 -0.0009 0.3813 0.0073					
GRADCONJ Iterações raizes	03 alidação : 0.0333 0.0086 0.0538 0.0103 0.9795 0.0035 0.9781 0.0051	Validação 20 1.0e-003 * 0.0006 0.1358 0.0018 0.3645 0.9637 0.0845 0.9522 0.1822	Validação 31 1.0e-004 * 0.0000 0.1046 0.0002 0.3456 0.9633 0.0659 0.9514 0.1728	Validação 42 1.0e-005 * 0.0000 0.0686 0.0000 0.2793 0.9633 0.0433 0.9513 0.1396	Validação 52 1.0e-006 * 0.0000 0.0484 0.0000 0.2417 0.9633 0.0306 0.9513 0.1209					

8. ROTINAS DOS MÉTODOS EM MATLAB

A Seguir, apresentam-se as rotinas, utilizadas, neste trabalhos

8.1 ROTINA 01: MÉTODO NEWTON-RAPHSON

pmnewra.m

```
function xaprox=pmnewra6(a,b,c,d,e,f,g,x0,a1,a2)
% metodo de Newton-Raphson p/equações de 2o.a 6o.grau
% a,b,c,d,e,f,g = valores para a equacao - os valores sao
posicionais para o polinomio
% x0 = valor da aproximacao inicial informado
% a1 e a2 = intervalos de aproximacoes informados
%
if nargin < 10
    error('Numero de Argumentos Insuficientes')
end
aproximic=x0;
if a > 0
    % equacao do 6o.grau
    wgrau='6o.GRAU';
    wraizes=[a b c d e f g];
    raizes=roots(wraizes);
else
    if b > 0
        % equacao do 5.grau
        wgrau='5o.GRAU';
        wraizes=[b c d e f g];
        raizes=roots(wraizes);
    else
        if c > 0
            % equacao do 4.grau
            wgrau='4o.GRAU';
            wraizes=[c d e f g];
            raizes=roots(wraizes);
        else
            if d > 0
                % equacao do 3.grau
                wgrau='3o.GRAU';
                wraizes=[d e f g];
                raizes=roots(wraizes);
            else
                % equacao do 2. grau
                wgrau='2o.GRAU';
                wraizes=[e f g];
                raizes=roots(wraizes);
            end
        end
    end
end
%
wfim=0;
witer=0;
x0ini=x0;
wxk=x0;
Fx0=(a*x0.^6 + b*x0.^5 + c*x0.^4 + d*x0.^3 + e*x0.^2 +
f*x0 + g);
if abs(Fx0) <= a1
```

```
    xaprox=x0;
    witer=0;
    wfim=1;
end
wcontador=0;
while wfim==0
    wcontador=wcontador+1;
    if wcontador > 30
        wfim=2;
        break
    end
    Fx0= ( a*x0.^6 + b*x0.^5 + c*x0.^4 + d*x0.^3 +
e*x0.^2 + f*x0 + g);
    FLx0=(6*a*x0.^5 + 5*b*x0.^4 + 4*c*x0.^3 + 3*d*x0.^2
+ 2*e*x0.^1 + f);
    x1= x0 - Fx0/ FLx0;
    Fx1=( a*x1.^6 + b*x1.^5 + c*x1.^4 + d*x1.^3 +
e*x1.^2 + f*x1 + g);
    if abs(Fx1) < a1
        xaprox=x1;
        wfim=3;
        break
    end
    Result4 = x1 - x0;
    if abs(Result4) < a2
        xaprox=x1;
        wfim=4;
        break
    end
    witer=wcontador;
    x0=x1;
    wxk(wcontador)=x0;
end
save PMNEWRA6 wxk witer raizes
disp('PMNEWRA6-Pgma do Metodo Newton-Raphson - para
equacoes do 2O. ATE O 6O.Grau')
disp(['Variaveis da equacao: (',num2str(a),', ',num2str(b),',
',num2str(c),', ',num2str(d),')'])
disp([' ',num2str(e),', ',num2str(f),',
',num2str(g),')'])
disp(' ')
disp(['aproximacao inicial informada (',num2str(aproximic),')'])
disp(' ')
disp('Precisoos Informadas:')
fprintf('a1=%1.18f \n',a1)
fprintf('a2=%1.18f \n',a2)
disp(' ')
disp(['ESTA E´ UMA EQUACAO DO ',num2str(wgrau),
' cujas raizes sao:'])
format long
raizes
disp('Arquivo PMNEWRA6xx.m - contem os displays da
execucao')
```

```
disp('Arquivo PMNEWRA6xx.mat - contem WITER-numero
de iterações e WXX-valores aprox.de x')
disp(' ')
disp(['Achamos o valor aproximado com ',num2str(witer),'
iteracoes'])
```

```
disp(' ')
disp('Valores encontrados para x, a partir do valor inicial:')
fprintf('wxk=%1.18f \n',wxk)
disp('Valor aprox.de x -> ')
fprintf('xaprox=%1.18f \n',xaprox)
```

Através dos testes executados, conforme os exemplos a seguir, pode-se comprovar que uma escolha cuidadosa da aproximação inicial é essencial para o bom desempenho do Método de Newton-Raphson. Gostaríamos de observar que quanto maior for a precisão exigida nos cálculos, mais recursos estaremos exigindo da máquina onde estiver sendo executados os testes.

$$f(x) = x^2 + x - 6 \text{ e } x_0 = 1.5$$

pmnewra601.m

pmnewra6(0,0,0,0,1,1,-6,1.5,0.0,0.000000005)
PMNEWRA6-Pgma do Metodo Newton-Raphson - para
equacoes do 2O. ATE O 6O.Grau
Variaveis da equação: (0, 0, 0, 0,1, 1, -6)

aproximação inicial informada (1.5)

Precisoas Informadas:
a1=0.000000000000000000
a2=0.000000005000000000

ESTA E' UMA EQUACAO DO 2o.GRAU cujas raizes sao:

raizes = -3 e 2

Arquivo PMNEWRA601.m - contem os displays da execucao

Arquivo PMNEWRA601.mat - contem WITER-numero de iterações e WXX-valores aprox.de x

Achamos o valor aproximado com 4 iteracoes

Valores encontrados para x, a partir do valor inicial:
wxk=2.062500000000000000
wxk=2.000762195121951400
wxk=2.000000116152868200
wxk=2.0000000000000002700
Valor aprox.de x ->
xaprox=2.000000000000000000

COMENTÁRIOS:

Veja que: precisão até a 8a. Casa decimal
ponto inicial próximo de uma das raízes

Tivemos: 4 iterações

$$f(x) = x^2 + x - 6 \text{ e } x_0 = 1.5$$

pmnewra602.m

pmnewra6(0,0,0,0,1,1,-6,1.5,0.0,0.000005)
PMNEWRA6-Pgma do Metodo Newton-Raphson - para
equacoes do 2O. ATE O 6O.Grau
Variaveis da equação: (0, 0, 0, 0,1, 1, -6)

aproximação inicial informada (1.5)

Precisoas Informadas:
a1=0.000000000000000000
a2=0.000005000000000000

ESTA E' UMA EQUACAO DO 2o.GRAU cujas raizes sao:

raizes = -3 e 2

Arquivo PMNEWRA602.m - contem os displays da execucao

Arquivo PMNEWRA602.mat - contem WITER-numero de iterações e WXX-valores aprox.de x

Achamos o valor aproximado com 3 iteracoes

Valores encontrados para x, a partir do valor inicial:
wxk=2.062500000000000000
wxk=2.000762195121951400
wxk=2.000000116152868200
Valor aprox.de x ->
xaprox=2.0000000000000002700

COMENTÁRIOS:

Veja que: precisão até a 5a. Casa decimal
ponto inicial próximo de uma das raízes

Tivemos: 3 iterações

$$f(x) = 3x^3 - 9x + 3 \quad \text{e} \quad x_0 = 1.5$$

pmnewra603.m

pmnewra6(0,0,0,1,0,-9,3,1.5,0.0,0.000000005)
PMNEWRA6-Pgma do Metodo Newton-Raphson - para
equacoes do 2O. ATE O 6O.Grau
Variaveis da equacao: (0, 0, 0, 1,0, -9, 3)

aproximação inicial informada (1.5)

Precisoas Informadas:

a1=0.000000000000000000

a2=0.000000005000000000

ESTA E´ UMA EQUACAO DO 3o.GRAU cujas raizes sao:

raizes = -3.15452300869520
2.81691405272937
0.33760895596584

Arquivo PMNEWRA603.m - contam os displays da
execucao

Arquivo PMNEWRA603.mat - contam WITER-numero de
iteracoes e WXX-valores aprox.de x

Achamos o valor aproximado com 11 iteracoes

Valores encontrados para x, a partir do valor inicial:

wxk=-1.666666666666666500

wxk=18.388888888888850000

wxk=12.366010403481944000

wxk=8.402306719815133100

wxk=5.835338164832405600

wxk=4.233873551002961500

wxk=3.322910960561274300

wxk=2.917338931107861500

wxk=2.822191665411167600

wxk=2.816929875843750400

wxk=2.816914052872279100

Valor aprox.de x ->

xaprox=2.816914052729369000

COMENTÁRIOS:

Veja que: precisão até a 8a. Casa decimal
ponto inicial não muito próximo das raízes

Tivemos: 11 iterações

$$f(x) = 3x^3 - 9x + 3 \quad \text{e} \quad x_0 = 1.5$$

pmnewra604.m

pmnewra6(0,0,0,1,0,-9,3,1.5,0.0,0.0005)
PMNEWRA6-Pgma do Metodo Newton-Raphson -
parequacoes do 2O. ATE O 6O.Grau
Variaveis da equacao: (0, 0, 0, 1,0, -9, 3)

aproximação inicial informada (1.5)

Precisoas Informadas:

a1=0.000000000000000000

a2=0.000500000000000000

ESTA E´ UMA EQUACAO DO 3o.GRAU cujas raizes sao:

raizes = -3.15452300869520
2.81691405272937
0.33760895596584

Arquivo PMNEWRA604.m - contam os displays da
execucao

Arquivo PMNEWRA604.mat - contam WITER-numero de
iteracoes e WXX-valores aprox.de x

Achamos o valor aproximado com 10 iteracoes

Valores encontrados para x, a partir do valor inicial:

wxk=-1.666666666666666500

wxk=18.388888888888850000

wxk=12.366010403481944000

wxk=8.402306719815133100

wxk=5.835338164832405600

wxk=4.233873551002961500

wxk=3.322910960561274300

wxk=2.917338931107861500

wxk=2.822191665411167600

wxk=2.816929875843750400

Valor aprox.de x ->

xaprox=2.816914052872279100

COMENTÁRIOS:

Veja que: precisão até a 3a. Casa decimal
ponto inicial não muito próximo das raízes

Tivemos: 11 iterações

$$f(x) = 3x^3 - 9x + 3 \quad \text{e} \quad x_0 = 1.5$$

pmnewra605.m

pmnewra6(0,0,0,1,0,-9,3,0.5,0.0,0.000000005)
PMNEWRA6-Pgma do Metodo Newton-Raphson - para
equacoes do 2O. ATE O 6O.Grau
Variaveis da equacao: (0, 0, 0, 1,0, -9, 3)

aproximacao inicial informada (0.5)

Precisoas Informadas:

a1=0.000000000000000000
a2=0.000000005000000000

ESTA E´ UMA EQUACAO DO 3o.GRAU cujas raizes sao:

raizes = -3.15452300869520
2.81691405272937

COMENTÁRIOS:

Veja que: precisão
ponto inicial

Tivemos:

0.33760895596584

Arquivo PMNEWRA605.m - contem os displays da execucao

Arquivo PMNEWRA605.mat - contem WITER-numero de iteracoes e WXX-valores aprox.de x

Achamos o valor aproximado com 3 iteracoes

Valores encontrados para x, a partir do valor inicial:

wxk=0.33333333333333370
wxk=0.337606837606837630
wxk=0.337608955965312820
Valor aprox.de x ->
xaprox=0.337608955965837730

até a 8a. Casa decimal
próximo das raízes

somente 3 iterações

$$f(x) = x^6 + x^2 + x - 6 \text{ e } x_0 = 1.5$$

pmnewra611.m

pmnewra6(1,0,0,0,1,1,-6,1.5,0.0,0.000005)
PMNEWRA6-Pgma do Metodo Newton-Raphson - para
equacoes do 2O. ATE O 6O.Grau
Variaveis da equacao: (1, 0, 0, 0,1, 1, -6)

aproximacao inicial informada (1.5)

Precisoas Informadas:

a1=0.000000000000000000
a2=0.000005000000000000

ESTA E´ UMA EQUACAO DO 6o.GRAU cujas raizes sao:

raizes =
-1.33098210214630
-0.71608301032157 + 1.20843466527941i
-0.71608301032157 - 1.20843466527941i

COMENTÁRIOS:

Veja que: precisão
ponto inicial

Tivemos:

0.77171922753417 + 1.13031571627189i
0.77171922753417 - 1.13031571627189i
1.21970966772111

Arquivo PMNEWRA611.m - contem os displays da execucao

Arquivo PMNEWRA611.mat - contem WITER-numero de iteracoes e WXX-valores aprox.de x

Achamos o valor aproximado com 4 iteracoes

Valores encontrados para x, a partir do valor inicial:

wxk=1.315573770491803400
wxk=1.233792610663979200
wxk=1.220048534195941100
wxk=1.219709867615793100
Valor aprox.de x ->
xaprox=1.219709667721178900

até a 5a. Casa decimal
próximo de uma das raízes

4 iterações

8.2 ROTINA 02: MÉTODO NEWTON

fnewton.m

```
function [res, it]=fnewton(func,dfunc,x,tol)
% x is an initial starting value, tol is required accuracy
format short
w=[0:0.5:8];
it=0; x0=x
f=feval(func,x0)           % resultado da funcao para x0
wcontador=1;
```

```
x0ac(wcontador)=x0;           % estou salvando x0
fac(wcontador)=f;             % estou salvando o resultado
da funcao para x0
d=feval(func,x0)/feval(dfunc,x0)
while abs(d) > tol
    x1=x0-d;
    it=it+1;
    x0=x1;
```

```

d=feval(func,x0)/feval(dfunc,x0)
wcontador=wcontador + 1;
f=feval(func,x0)
x0ac(wcontador)=x0;          % estou salvando x0
fac(wcontador)=f;            % estou salvando o resultado
da funcao para x0
com os pontos de x0 e o result.funcao p/x0

```

```

end;
res=x0;
plot(w,feval(func,w));        % grafico da funcao para
valores pre-definidos de w
plot(x0ac,fac,'o',w,feval(func,(w))); % grafico da funcao

```

$$f(x) = x^3 - 10x^2 + 29x - 20 \quad \text{e} \quad x_0 = 7.0$$

f302.m

```

function F=f302(x);
F=x.^3 - 10.0*x.^2 + 29.0*x - 20.0;

```

```

d = 0.2973
f = 2.5646

```

f303.m

```

function F=f303(x);
F=3*x.^2 - 20*x + 29;

```

```

d = 0.1012
f = 0.5249

```

fnewton01.m

```

[x,it]=fnewton('f302','f303',7,0.00005)

```

```

d = 2.1235e-004
f = 8.4987e-004

```

```

x0 = 7
f = 36
d = 1

```

```

d = 5.6373e-008
f = 2.2549e-007

```

```

d = 0.5882
f = 10

```

```

x = 5.0000
it = 6

```

8.3 ROTINA 03 MÉTODO DOS GRADIENTES CONJUGADOS

mincg.m

```

function res=mincg(f,derf,ftau,x,tol)
% METODO DOS GRADIENTES CONJUGADOS
% dk -----> direcao conjugada
% dkAnt -----> valor anterior da direcao conjugada
% dkNeg -----> negativo da direcao conjugada
% dkNegAnt --> valor anterior do negativo da direcao conjuga
% sk -----> valor minimizado da funcao dada
% p1 -----> valor de x que vai ser passado para a funcao
ftau2cg
%
% step 0 : Input value P/x0 and accuracy E. Set k = 0 e
compute dk=-Vf(xk)
global p1 dkNeg ;
n=size(x) ;
noiter=0 ;
% Vamos calcular Vf(xk), a primeira direcao conjugada
(prim.deriv.x) ;
dk = feval(derf,x) ;
% LOOP PRINCIPAL ;
while norm(dk) > tol ;
    noiter = noiter + 1 ;
    % agora definimos a primeira direcao conjugada
    dk = feval(derf,x) ;
    % definimos o negativo dessa direcao conjugada
    dkNeg=-dk ;

```

```

% LOOP INTERIOR - processando de 1 ate n (todas
colunas de x) ;
for inner = 1:n ;
    % vou salvar a direcao conjugada atual (dk)
    dkAnt = dk ;
    % vou salvar o negativo da direcao conj.atual (dkNeg)
    dkNegAnt = dkNeg ;
    % aqui vou achar Sk que minimiza a funcao f(x)
    inner ;
    p1 = x
    dkNeg
    ftau
    % sk = fmin(ftau,-10,10,[0 0.00005])
    sk = teste(ftau,-10,10)
    % vou achar o novo x que e o x1 ou -> x(k+1) = x(k) +
    S(k)*-d(k) ;
    pause
    x1 = x + sk * dkNeg ;
    % agora vou calcular a nova direcao conjugada c/o novo
    dk = feval(derf,x1) ;
    % vou calcular Beta
    A = (dk' * dk) ;
    B = (dkAnt' * dkAnt) ;
    Beta= A / B ;

```

```

% agora vou calcular o novo valor da direção conjugada
negativa ;
dkNeg = -dk + dkNegAnt * Beta ;
% agora salvo o novo x1 em x para continuar o loop
;
x = x1 ;
end ;
end ;

```

```

res=x1
disp('Iterações')
disp(noiter)
disp('Solution')
disp(x1)
disp('Gradient')
disp(dk)

```

$$f(x_1, x_2) = (x_1^4 - 16x_1^2 + 5x_1)/2 + (x_2^4 - 16x_2^2 + 5x_2)/2 \text{ e } x_0 = [0,1]$$

f801.m

```

function fv=f801(x);
fv=0.5*( x(1).^4 - 16*x(1).^2 + 5*x(1) ) + 0.5*( x(2).^4 -
16*x(2).^2 + 5*x(2));

```

```

x0=[0 1];
x1=Mincg('f801','f801pd','ftau2cg',x0,0.000005);

```

com o seguinte resultado:

f801pd.m

```

function pd=f801pd(x);
pd=zeros(size(x));
pd(1)=4*x(1).^3 - 32*x(1)+5;
pd(2)=4*x(2).^3 - 32*x(2)+5;

```

tmincg01.m

```

res = -2.9035 -2.9035

Solution -2.9035 -2.9035

Gradient 1.0e-005 *
-0.0830 -0.1404

```

Para executar utilizou-se deste programa:

```

x1 = -2.9035 -2.9035

```

tmincg.m

```

global p1 dkNeg;

```

Para visualizar o gráfico dessa função, criou-se o seguinte programa:

fmeshmincg.m

```

clf
[x,y]=meshgrid(-4.0:0.2:4.0,-4.0:0.2:4.0)
z=0.5*( x.^4 - 16*x.^2 + 5*x ) + 0.5*( y.^4 - 16*y.^2 + 5*y)
figure(1)
surfl(x,y,z)
axis([-4 4 -4 4 -80 20])
xlabel('x1')
ylabel('x2')
zlabel('z')
x1=[0 0.8618 -2.5776 -3.3682 -3.0484 -2.8512 -2.9056 -
2.9035]

```

```

y1=[1 -2.9644 -1.0750 -2.5092 -2.3899 -2.8824 -2.8968 -
2.9035]
figure(2)
contour(x,y,z)
xlabel('x1')
ylabel('x2')
hold on
plot(x1,y1,x1,y1,'o')
xlabel('x1')
ylabel('x2')
hold off

```

8.4 ROTINA 04: MÉTODO DE NEWTON

Podemos observar que o programa abaixo trata qualquer sistema de equações não lineares de até 2 incógnitas. Como parâmetros externos ao programa informamos a matriz do sistema não linear, e não é necessário o programa chamar nenhuma função externa. Após o programa, teremos 3 exemplos para teste do método. Logo após, mostra-se a programação do mesmo método de Newton, só que nesse caso podemos observar que o sistema de equações é externo ao programa – estamos utilizando o *feval* do Matlab. Neste anexo teremos então o programa *pmnewton.m* (sem *feval*) e três exemplos exercícios. Apresenta-se o programa *newtoni.m* (com *feval*) e o respectivo teste do método.

pmnewton.m

```

function PMNEWTON(a,b,e)
% Metodo Iterativo de Newton p/obtencao de raizes de
equacoes de sistemas NAO lineares
% as equacoes podem ter ate 2 variaveis: x1 e x2

```

```

% a = possui a matriz com os valores das equacoes do sistema
não linear
% b = possui a matriz com os valores da aproximação inicial
x0
% e = tolerancia a ser respeitada
%

```

```

if nargin < 3
    error('Numero de Argumentos Insuficientes')
end
%
wfim=0;
% Vamos ver qual e' o tamanho da matriz do sist.linear
wtotcol=size(a,2); % descubro quantas colunas tem a matriz
informada
wtotlin=size(a,1); % descubro quantas linhas tem a matriz
informada
waux=wtotlin+1;
if wtotcol ~= waux
    error('Matriz do Sistema Informada nao esta com Total
Colunas = Linhas + 1 ')
    wfim=1;
end
% b tem que ter somente 1 coluna, e o total de linhas = total de
linhas da matriz a
if size(b,2) ~= 1
    error('Matriz com as aprox.iniciais tem mais de uma
coluna')
    wfim=1;
end
if size(b,1) ~= size(a,1)
    error('Matriz com as aprox.iniciais nao tem mesmo
num.linha da matriz do sist.linear')
    wfim=1;
end
%
xk0(1,1)=b(1);
xk0(2,1)=b(2);
%
wfim=0;
wxk=b;
witer=0;
wcontador=0;
while wfim==0
    wcontador=wcontador+1;
    if wcontador > 10
        witer=wcontador;
        wxk(1,wcontador)=xk1(1,1);
        wxk(2,wcontador)=xk1(2,1);
        wxk(3,wcontador)=wxmax;
        wfim=3;
        break
    end
    %
    Arg11=a(1,1)*xk0(1,1)+a(1,2)*xk0(2,1)+a(1,3);
    Arg21=a(2,1)*xk0(1,1)^2+a(2,2)*xk0(2,1)^2+a(2,3);
    Fx=[Arg11;Arg21];
    wxmax=max(abs(Fx));

```

```

if wxmax < e
    witer=wcontador;
    wxk(1,wcontador)=Fx(1,1);
    wxk(2,wcontador)=Fx(2,1);
    wxk(3,wcontador)=wxmax;
    wfim=1;
    break
end
Arg11=(a(1,1)*1);
Arg12=(a(1,2)*1);
Arg21=(2*a(2,1)*xk0(1,1));
Arg22=(2*a(2,2)*xk0(2,1));
Jx=[Arg11,Arg12;Arg21,Arg22];
Fxmenos=Fx*[-1];
InvJx=inv(Jx);
S = InvJx * Fxmenos;
xk1=xk0+S;
wxmax=max(abs(S));
if wxmax < e
    witer=wcontador;
    wxk(1,wcontador)=xk1(1,1);
    wxk(2,wcontador)=xk1(2,1);
    wxk(3,wcontador)=wxmax;
    wfim=2;
    break
end
witer=wcontador;
wxk(1,wcontador)=xk1(1,1);
wxk(2,wcontador)=xk1(2,1);
wxk(3,wcontador)=wxmax;
xk0(1,1)=xk1(1,1);
xk0(2,1)=xk1(2,1);
end
save PMNEWTON wxk witer;
disp('PMNEWTON-Pgma do Metodo Newton - para sist.NAO
lineares com 2 incognitas')
disp('Matriz do Sistema NAO Linear informado:')
a
disp('Matriz das Aproximações Iniciais informada :')
b
disp('Precisao Informada:')
fprintf('e=%1.14f \n',e)
disp('Arquivo PMNEWTONxx.m - contem os displays da
execucao')
disp('Arquivo PMNEWTONxx.mat - contem WITER-numero
de iterações e WXX-valores aprox.de x')
disp(['Achamos o valor aproximado com ',num2str(witer),'
iteracoes'])
disp('Valores encontrados para x, a partir do valor inicial:')
format long
wxk

```

$$F(x) = \begin{bmatrix} x_1 + x_2 - 3 \\ x_1^2 + x_2^2 - 9 \end{bmatrix}$$

pmnewton01.m

```

pmnewton(a,b,e)
PMNEWTON-Pgma do Metodo Newton - para sist.NAO
lineares com 2 incognitas

```

Matriz do Sistema NAO Linear informado:

```

a = 1    1   -3
     1    1   -9

```

Matriz das Aproximações Iniciais informada :

b = 1
5

Precisao Informada:
e=0.005000000000000

Arquivo PMNEWTONxx.m - contem os displays da execucao

Arquivo PMNEWTONxx.mat - contem WITER-numero de iterações e WXX-valores aprox.de x
Achamos o valor aproximado com 4 iteracoes
Valores encontrados para x, a partir do valor inicial:

wxk =
-0.6250 -0.0919 -0.0027 -0.0000
3.6250 3.0919 3.0027 3.0000

$$\begin{cases} f_1(x_1, x_2) = x_1^2 - x_2^2 - 2 = 0 \\ f_2(x_1, x_2) = x_1^2 - \frac{x_2^2}{9} - 1 = 0 \end{cases}$$

pmnewton02.m

pmnewton(a,b,e)
PMNEWTON-Pgma do Metodo Newton - para sist.NAO
lineares com 2 incognitas
Matriz do Sistema NAO Linear informado:
Matriz das Aproximações Iniciais informada :

a = 1.0000 1.0000 -2.0000
1.0000 -0.1111 -1.0000

b = 1
5

Arquivo PMNEWTON02.m - contem os displays da execucao
Arquivo PMNEWTON02.mat - contem WITER-numero de iterações e WXX-valores aprox.de x
Achamos o valor aproximado com 5 iteracoes
Valores encontrados para x, a partir do valor inicial:

Precisao Informada:
e=0.005000000000000

wxk =
0.4643 1.2884 1.0677 1.0492 0.0000
1.5357 0.7116 0.9323 0.9508 0.0003

$$\begin{cases} f_1(x_1, x_2) = x_1^2 - x_2 - 0.2 = 0 \\ f_2(x_1, x_2) = x_2^2 - x_1 + 1 = 0 \end{cases}$$

pmnewton03.m

pmnewton(a,b,e)
PMNEWTON-Pgma do Metodo Newton - para sist.NAO
lineares com 2 incognitas
Matriz do Sistema NAO Linear informado:

a = 1.0000 -1.0000 -0.2000
1.0000 -1.0000 1.0000

Matriz das Aproximações Iniciais informada :

b = 1
5

Precisao Informada:
e=0.005000000000000

Arquivo PMNEWTON03.m - contem os displays da execucao
Arquivo PMNEWTON03.mat - contem WITER-numero de iterações e WXX-valores aprox.de x
Achamos o valor aproximado com 3 iteracoes
Valores encontrados para x, a partir do valor inicial:

wxk =
3.375000000000000 -2.399999999999999 -0.000000000000000
3.175000000000000 -2.599999999999999 0.000000000000001

Programa *newton.m* para sistemas de equações utilizando-se do *feval*, isto é, chamando o sistema de equações de dentro do próprio programa.

newton.m

```
function [xv,it]=newton(x,f,jf,n,tol)
global i
% User must supply initial approximations for the N variables
x,
% the definitions of the functions of the system of equations
% and the partial derivatives in the form of the Jacobian
matrix.
it=0;
xv=x;
size(x,2);
n;
if size(x,1) ~= n
    error('Numero de variaveis difere da esperada')
end
wfim=0;
it = 0;
while wfim==0
    if it > 10
        wxk(:,it)= xv1;      % Salvar o atual xv, para rastrear
        xv=xv1;
        wxkmax(1,it) = wxmax ; % Salvar o valor maximo da
iteração
        wfim=3;
        break
    end
    fr=feval(f,xv);
    wxmax=max(abs(fr));
    if wxmax < tol
        if it == 0
            wxk(:,1)= xv;      % Salvar o atual xv, para rastrear
            wxkmax(1,1) = wxmax; % Salvar o valor maximo da
iteração
        else
            wxk(:,it)= xv;      % Salvar o atual xv, para rastrear
            wxkmax(1,it) = wxmax; % Salvar o valor maximo da
iteração
```

```
        end
        wfim=1;
        break
    end
    Jr=feval(jf,xv);
    fr;
    S=inv(Jr)*-fr;
    x0 = xv;
    xv1= x0 + S;
    wxmax=max(abs(S));
    if wxmax < tol
        wxk(:,it)= xv1;      % Salvar o atual xv, para rastrear
        xv=xv1;
        wxkmax(1,it) = wxmax; % Salvar o valor maximo da
iteração
        wfim=2;
        break
    end
    it = it + 1;
    wxk(:,it)= xv1;      % Salvar o atual xv, para rastrear
    wxkmax(1,it) = wxmax; % Salvar o valor maximo da
iteração
    xv=xv1;
end
save NEWTON wxk wxkmax it;
wfim
disp('Newton-Pgma do Metodo Newton - sistemas nao
lineares -');
disp('Arquivo NEWTONxx - contem os displays da
execucao');
disp('Arquivo NEWTONxx - contem IT-numero de iterações e
WKK-valores aprox.de x');
disp(['Total de iteracoes: ',num2str(it)])
disp('Valores encontrados para x, a partir do valor inicial:');
wxk;
wxkmax;
```

Para executar o programa foi criado o Arquivo de Lote:

tnewton01.m

```
function fun=tnewton01(tol)
sol=newton([3;-1.5],'f308','f309',2,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
disp(sol)
disp(' Validacao : ')
val=f308(sol);
disp(val)
```

$$\begin{cases} x^2 + y^2 = 4 \\ xy = 1 \end{cases}$$

newton01.m

```
function fun=newton01(tol)
sol=newton([3;-1.5],f308',f309',2,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
disp(sol)
disp(' Validação : ')
val=f308(sol);
disp(val)
```

```
% set of vector to size required by function newtonmv
f=zeros(2,1);
f(1) = x.^2 + y.^2 - 4;
f(2) = x*y - 1;
```

f309.m

```
function jf=f309(v)
x=v(1);
y=v(2);
% set jf matrix to size required by fuction newtonmv
jf=zeros(2,2);
% each row of jf is assigned the appropriate partial derivatives
jf(1,:) = [2*x 2*y];
jf(2,:) = [y x];
```

f308.m

```
function f=f308(v)
x=v(1);
y=v(2);
```

Como resultado obteve-se:

newton01.m

```
Newton-Pgma do Metodo Newton - sistemas nao lineares -
Arquivo NEWTONxx - contem os displays da execucao
Arquivo NEWTONxx - contem IT-numero de iteracoes e
WXX-valores aprox.de x
Total de iteracoes: 4
Valores encontrados para x, a partir do valor inicial:
Tolerancia Informada:
0.0050
```

Raizes do sistema :

```
1.9319
0.5176
```

Validação :

```
1.0e-003 *
```

```
0.1207
```

```
-0.0603
```

8.5 ROTINA 05: MÉTODO QUASE NEWTON

Neste item apresenta-se o programado de dois modos distintos. O primeiro deles trata qualquer sistema de equações não lineares de até 2 incógnitas. Como parâmetros externos ao programa informamos a matriz do sistema não linear, e não é necessário o programa chamar nenhuma função externa. Tem-se um exemplo para se testar esse programa e em seguida, apresenta-se o mesmo Método programado, com o sistema de equações sendo chamado de dentro do programa, com *feval*.

pmquasen.m

```
function PMQUASEN(a,b,e)
% Metodo Iterativo de QUASE-Newton p/obtencao de raizes
de equacoes de sistemas NAO lineares
% as equacoes podem ter ate 2 variaveis: x1 e x2
% a = possui a matriz com os valores das equacoes do sistema
linear
% b = possui a matriz com os valores da aproximacao inicial
x0
% e = tolerancia a ser respeitada
%
if nargin < 3
error('Numero de Argumentos Insuficientes')
end
%
wfim=0;
% Vamos ver qual e' o tamanho da matriz do sist.linear
wtotcol=size(a,2); % descubro quantas colunas tem a matriz
informada
wtotlin=size(a,1); % descubro quantas linhas tem a matriz
informada
```

```
waux=wtotlin+1;
if wtotcol ~= waux
error('Matriz do Sistema Informada nao esta com Total
Colunas = Linhas + 1 ')
wfim=1;
end
% b tem que ter somente 1 coluna, e o total de linhas = total de
linhas da matriz a
if size(b,2) ~= 1
error('Matriz com as aprox.iniciais tem mais de uma
coluna')
wfim=1;
end
if size(b,1) ~= size(a,1)
error('Matriz com as aprox.iniciais nao tem mesmo
num.linha da matriz do sist.linear')
wfim=1;
end
%
xk0(1,1)=b(1);
xk0(2,1)=b(2);
```

```

x0(1,1)=b(1);
x0(2,1)=b(2);
%
wfim=0;
wxk=b;
witer=0;
wcontador=0;
% TRATAMENTO DA MATRIZ JACOBIANA
Arg11=(a(1,1)*1);
Arg12=(a(1,2)*1);
Arg21=(2*a(2,1)*x0(1,1));
Arg22=(2*a(2,2)*x0(2,1));
Jx=[Arg11,Arg12;Arg21,Arg22];
InvJx=inv(Jx);
% InvJx sera obtido apenas uma vez
while wfim==0
    wcontador=wcontador+1;
    if wcontador > 10
        witer=wcontador;
        wxk(1,wcontador)=xk1(1,1);
        wxk(2,wcontador)=xk1(2,1);
        wxk(3,wcontador)=wxmax;
        wfim=3;
        break
    end
    Arg11=a(1,1)*xk0(1,1)+a(1,2)*xk0(2,1)+a(1,3);
    Arg21=a(2,1)*xk0(1,1)^2+a(2,2)*xk0(2,1)^2+a(2,3);
    Fx=[Arg11;Arg21];
    wxmax=max(abs(Fx));
    if wxmax < e
        witer=wcontador;
        wxk(1,wcontador)=Fx(1,1);
        wxk(2,wcontador)=Fx(2,1);
        wxk(3,wcontador)=wxmax;
        wfim=1;
        break
    end
end

```

```

Fxmehos=Fx*[-1];
S = InvJx * Fxmehos;
xk1=xk0+S;
wxmax=max(abs(S));
if wxmax < e
    witer=wcontador;
    wxk(1,wcontador)=xk1(1,1);
    wxk(2,wcontador)=xk1(2,1);
    wxk(3,wcontador)=wxmax;
    wfim=2;
    break
end
witer=wcontador;
wxk(1,wcontador)=xk1(1,1);
wxk(2,wcontador)=xk1(2,1);
wxk(3,wcontador)=wxmax;
xk0(1,1)=xk1(1,1);
xk0(2,1)=xk1(2,1);
end
save PMQUASEN wxk witer;
disp('PMQUASEN-Pgma do Metodo QUASE-Newton - para
sist.lineares com 2 incognitas')
disp('Matriz do Sistema Linear informado:')
a
disp('Matriz das Aproximações Iniciais informada :')
b
disp('Precisao Informada:')
fprintf('e=%1.14f\n',e)
disp('Arquivo PMQUASENxx.m - contem os displays da
execucao')
disp('Arquivo PMQUASENxx.mat - contem WITER-numero
de iterações e WKK-valores aprox.de x')
disp(['Achamos o valor aproximado com ',num2str(witer),'
iteracoes'])
disp('Valores encontrados para x, a partir do valor inicial:')
wxk

```

$$\begin{cases} x_1 + x_2 - 3 = 0 \\ x_1^2 + x_2^2 - 9 = 0 \end{cases}$$

pmquasen01.m

pmquasen(a,b,e)
 PMQUASEN-Pgma do Metodo QUASE-Newton - para
 sist.lineares com 2 incognitas
 Matriz do Sistema Linear informado:

```

a = 1    1   -3
    1    1   -9

```

Matriz das Aproximações Iniciais informada :

```

b = 1
    5

```

Precisao Informada:

e=0.0500000000000000
 Arquivo PMQUASEN01.m - contem os displays da
 execucao
 Arquivo PMQUASEN01.mat - contem WITER-numero
 de iterações e WKK-valores aprox.de x
 Achamos o valor aproximado com 3 iteracoes
 Valores encontrados para x, a partir do valor inicial:

```

wxk =
-0.6250000000000000    -0.0585937500000000    -
0.01379013061523
3.6250000000000000    3.0585937500000000
3.01379013061523
1.6250000000000000    0.5664062500000000
0.04480361938477

```

Progrma *quasnew.m* para sistemas de equações utilizando-se do *feval*, isto é, chamando o sistema de equações de dentro do próprio programa.

quasnew.m

```
function [xv,it]=quasnew(x,f,jf,n,tol)
global it
% User must supply initial approximations for the N variables
x,
% the definitions of the functions of the system of equations
% and the partial derivatives in the form of the Jacobian
matrix.
it=0;
xv=x;
size(x,2);
n;
if size(x,1) ~= n
    error('Numero de variaveis difere da esperada')
end
wfim=0;
it = 0;
Jr=feval(jf,xv)           % resolvido somente uma vez
while wfim==0
    if it > 30
        wxk(:,it)= xv1;      % Salvar o atual xv, para rastrear
        xv=xv1;
        wxkmax(1,it) = wxmax; % Salvar o valor maximo da
iteração
        wfim=3;
        break
    end
    fr=feval(f,xv)
    wxmax=max(abs(fr));
    if wxmax < tol
        if it == 0
            wxk(:,1)= xv;      % Salvar o atual xv, para rastrear
            wxkmax(1,1) = wxmax; % Salvar o valor maximo da
iteração
        else
            wxk(:,it)= xv;      % Salvar o atual xv, para rastrear
            wxkmax(1,it) = wxmax; % Salvar o valor maximo da
iteração
```

```
end
wfim=1;
break
end
% Jr=feval(jf,xv);
fr;
S=inv(Jr)*-fr;
x0 = xv;
xv1= x0 + S;
wxmax=max(abs(S));
if wxmax < tol
    wxk(:,it)= xv1;      % Salvar o atual xv, para rastrear
    xv=xv1;
    wxkmax(1,it) = wxmax; % Salvar o valor maximo da
iteração
    wfim=2;
    break
end
it = it + 1;
wxk(:,it)= xv1;      % Salvar o atual xv, para rastrear
wxkmax(1,it) = wxmax; % Salvar o valor maximo da
iteração
xv=xv1;
end
wfim
save QUASNEW wxk wxkmax it;
disp('quasnew-Pgma Metodo Quase Newton - sist. nao lineares
');
disp('Arquivo QUASNEWxx - contem os displays da
execucao');
disp('Arquivo QUASNEWxx - contem IT-num.iteracoes e
WXX-valores aprox.de x');
disp(['Total de iteracoes: ',num2str(it)])
disp('Valores encontrados para x, a partir do valor inicial:');
wxk;
wxkmax;
```

$$\begin{cases} x^2 + y^2 = 4 \\ xy = 1 \end{cases}$$

tquasnew01.m

```
function fun=tquasnew01(tol)
sol=quasnew([3;-1.5],'f308','f309',2,tol);
disp(' Tolerancia Informada:')
disp(tol)
```

obtiv-se o seguinte resultado:

quasnew01.m

```
quasnew-Pgma Metodo Quase Newton - sist.nao lineares
Arquivo QUASNEWxx - contem os displays da execucao
Arquivo QUASNEWxx - contem IT-num.iteracoes e WXX-
valores aprox.de x
Total de iteracoes: 12
Valores encontrados para x, a partir do valor inicial:
```

```
disp('Raizes do sistema : ')
disp(sol)
disp(' Validacao : ')
val=f308(sol);
disp(val)
```

Tolerancia Informada:
0.0050

Raizes do sistema :
1.9361
0.5147

Validacao :

0.0134
-0.0035

8.6. ROTINA 06: MÉTODO DE BROYDEN

broyden.m

```
function[xv,it]=broyden(x,f,n,tol)
% Requires function f and an initial approx x for the n
variables
fr=zeros(n,1)
it=0;
xv=x;
% Set initial Br
Br=eye(n);
fr=feval(f,xv);
wxk(:,1)=xv;
while norm(fr) > tol
    pr= -Br*fr;
    tau=1;
    xv1=xv+(pr*tau);
    xv=xv1;
    oldfr=fr;
    fr=feval(f,xv);
    % Update approximation to Jacobian using Broyden's
    formula
    y = fr - oldfr;
```

```
oldBr=Br;
oyp=oldBr*y-pr;
pB=pr'*oldBr;
for i=1:n
    for j=1:n
        M(i,j) = oyp(i)*pB(j);
    end
end
Br=oldBr-M./(pr'*oldBr*y);
it = it+1;
wxk(:,it)= xv;      % Salvar o atual xv, para rastrear
end
save BROYDEN wxk it;
disp('broyden-Pgma Metodo Broyden - sist.nao lineares ');
disp('Arquivo BROYDENxx - contem os displays da
execucao');
disp('Arquivo BROYDENxx - contem IT-num.iteracoes e
W XK-valores aprox.de x');
disp(['Total de iteracoes: ',num2str(it)])
disp('Valores encontrados para x, a partir do valor inicial:');
wxk;
```

$$\begin{cases} x^2 + y^2 = 4 \\ xy = 1 \end{cases}$$

broyden01.m

broyden-Pgma Metodo Broyden - sist.nao lineares
Arquivo BROYDEN01 - contem os displays da execucao
Arquivo BROYDEN01 - contem IT-num.iteracoes e W XK-valores aprox.de x
Total de iteracoes: 36

ans =
0.5176
1.9319

8.7. ROTINA 07: MÉTODO STEEPEST DESCENT PARA SISTEMAS NÃO LINEARES

steepest.m

```
function res=steepest(x,f,derf,n,tol)
% METODO STEEPEST DESCENT
niter=700;
wfim=0;
size(x,2);
n;
if size(x,1) ~= n
    error('Numero de variaveis difere da esperada')
end
k=1;
while k < niter % enquanto k for menor que o numero de
iteracoes informado, continuamos
    k;
    x;
```

```
gx = feval(f,x) ;
gl = calcg(n,gx);
Jx = feval(derf,x);
Jx';
Fx = feval(f,x);
z = 2*Jx'*Fx;
z0 = norm(z); % 419.554
if z0 == 0
    wfim=1
    break
end
z = z / z0; % z=(-0.0214, -0.0193, 0.99958)
alfa1 =0;
alfa3 = 1;
xaux = x - alfa3 * z ;
gx = feval(f,xaux) ;
```

```

g3 = calcg(n,gx);
while g3 >= g1
    alfa3 = alfa3 / 2;
    xaux = x - alfa3 * z;
    gx = feval(f,xaux);
    g3 = calcg(n,gx);
    limite = tol/2;
    if alfa3 < limite
        break
    end
end
alfa2 = alfa3 / 2;
xaux = x - alfa2 * z;
gx = feval(f,xaux);
g2 = calcg(n,gx);
h1 = (g2 - g1) / alfa2;
h2 = (g3 - g2) / (alfa3 - alfa2);
h3 = (h2 - h1) / alfa3;
alfa0 = 0.5*(alfa2 - h1/h3);
xaux = x - alfa0 * z;
gx = feval(f,xaux);
g0 = calcg(n,gx);
if g0 < g3
    alfa = alfa0;
else
    alfa = alfa3;
end
xaux = x - alfa * z;
gx = feval(f,xaux);
g = calcg(n,gx);
x1 = x - alfa * z;
x = x1;
gfinal = g - g1;

function res=calcg(n,gx)
if n > 2
    valg = gx(1,1).^2 + gx(2,1).^2 + gx(3,1).^2 ; % 111.975
else

if abs(gfinal) < tol
    wgx(:,k)=gx ; % wgx k
    wxk(:,k)= x ; % wxk k
    wgx1x2x3(k)=g ; % wgx1x2x3 k
    wfim=2
    break
end
wgx(:,k)=gx; % wgx k
wxk(:,k)= x ; % wxk k
wgx1x2x3(k)=g ; % wgx1x2x3 k
k;
k = k + 1;
end
res=x;
it = k;
save STEEPEST wxk wgx wgx1x2x3 it;
disp('Steepest-Pgma Metodo Steepest Descent - sist.nao lineares ');
disp('Arquivo STEEPESTxx - contem os displays da execucao');
disp('Arquivo STEEPESTxx - contem IT-num.iteracoes e WXX-valores aprox.de x');
disp(['Total de iteracoes: ',num2str(it)])
disp('Valores encontrados para x, a partir do valor inicial:');
wxk;
disp('Valores de gx a partir do valor inicial:');
wgx;
disp('Valores de g(x1,x2,x3) a partir do valor inicial:');
wgx1x2x3;
disp('Iteracoes');
disp(it);
disp('Solution');
disp(x);

valg = gx(1,1).^2 + gx(2,1).^2 ;
end
res=valg;

```

$$\begin{cases} x^2 + y^2 = 4 \\ xy = 1 \end{cases}$$

steepest01.m

Steepest-Pgma Metodo Steepest Descent - sist.nao lineares
Arquivo STEEPEST01 - contem os displays da execucao
Arquivo STEEPEST01 - contem IT-num.iteracoes e WXX-
valores aprox.de x
Total de iteracoes: 7

Solution

1.9318
0.5178

ans =

1.9318
0.5178

8.8 ROTINA 08. MÉTODO GRADIENTES CONJUGADOS PARA SISTEMAS NÃO LINEARES

gradconj.m

```

function xdash=gradconj(x,a,b,n,tol)
global k f derf
x ;

```

```

a ;
b ;
n ;
tol ;

```

```

xdash=[] ;
gdash=[] ;
ddash=[] ;
qdash=[] ;
q =[] ;
mxitr=n*n ;
xdash=x ;
wxk(:,1)= xdash ;
gdash=b ;
ddash=-gdash ;
muinit=b*b ;
stop_criterion1=1 ;
k=1 ;
mu=muinit ;
wfim='nao' ;
% main stage
while (stop_criterion1==1)
    qdash=a*ddash ;
    q=qdash ;
    r=ddash*q ;
    if (r == 0)
        error('r=0,divide by 0!!!')
    end
    s=mu/r ;
    xdash ;
    s ;
    ddash ;
    xdash=xdash+s*ddash ;      % novo valor xdash
    gdash=gdash+s*q ;          % novo valor gdash
    t=gdash*qdash ;
    beta=t/r ;
    -gdash ;
    beta ;
    ddash ;
    ddash=-gdash+beta*ddash ;   % novo valor ddash
    mu=beta*mu ;                % novo valor de mu
    k=k+1 ;
    a ;
    wxk ;

    val=a*xdash ;
    b ;
    aux1 = val' ;
    aux2 = (b/(norm(val))) * norm(b) ;
    aux3 = (1 - aux1) * aux2 ;
    aux4 = mu/muinit ;
    if aux3 <=tol & aux4 <=tol
        stop_criterion1=0 ;
        wfim='1' ;
        wxk(:,k)= xdash ;      % Salvar o atual xv, para rastrear
        break ;
    end
    % colocado hoje - 30-07-02
    % if k > mxitr
    % stop_criterion1=0 % ;
    % wfim='2' ;
    % break ;
    x=xdash ;
    wxk(:,k)= x                % Salvar o atual xv, para rastrear
    % pause
    jx=feval(derf,x) ;
    a=jx ;
    b=feval(f,x) ;
    gdash=b ;
    mu=b*b ;
    ddash=-gdash ;
end
stop_criterion1
wfim
save GRADCONJ wxk k ;
disp('Gradconj-Pgma Metodo Grad.Conjugados - sist.nao
lineares ');
disp('Arquivo GRADCONJxx - contem os displays da
execucao');
disp('Arquivo GRADCONJxx - contem K-num.iteracoes e
WXX-valores aprox.de x') ;
disp(['Total de iteracoes: ',num2str(k-1)])
disp('Valores encontrados para x, a partir do valor inicial:') ;

```

Para se testar o método, foi criado o Arquivo de Lote abaixo que gera um sistema de 10 equações com seleção aleatória de elementos (VER O CAPÍTULO SEGUINTE)

tsolvercg.m

```

n=10
tol=1e-8
A = 10 * rand(n)
b = 10 * rand(n,1)
ada = A * A'

```

```

% To ensure a symmetric positive definitive matrix
sol = gradconj(ada,b,n,tol)
disp('Solution of system is : ')
disp(sol)
accuracy = norm(ada * sol - b)
fprintf('Norm of residuals =%12.9f\n',accuracy)

```

O resultado que se obteve pode ser visto abaixo:

Solution of system is :

```

0.7201      -15.076
0.1330      31.510
            12.167
            -31.330
            -34.515
            17.036

```

```

0.5631      10.171

```

```

accuracy = 3,43E-04

```

```

Norm of residuals = 0.000000034

```

f310.m

```
function f=f310(v)
x=v(1);
y=v(2);
z=v(3);
% set of vector to size required by function newton
f=zeros(3,1);
f(1)=sin(x) + y*y + log(z) - 7;
f(2)= 3*x + 2.^y - z.^3 + 1;
f(3)= x + y + z - 5;
```

f311.m

```
function jf=f311(v)
x=v(1);
y=v(2);
z=v(3);
% set jf matrix to size required by fuction newton
jf=zeros(3,3);
% each row of jf is assigned the appropriate partial derivatives
jf(1,:)=[cos(x) 2*y 1/z];
jf(2,:)=[3 (2.^y)*log(2) -3*(z.^2)];
jf(3,:)=[1 1 1];
```

9. ARQUIVOS ‘FUNCTION’ UTILIZADAS NAS ROTINAS DO ITEM 8

A Seguir, apresentam-se as funções utilizadas no item 8, deste trabalho

9.1 FUNÇÕES DO 3º. E 4º. SISTEMA DE EQUAÇÕES

f312.m

```
function f=f312(v)
x1=v(1);
x2=v(2);
x3=v(3);
% set of vector to size required by function newtonmv
f=zeros(3,1);
f(1)= 3*x1 - cos(x2*x3) - 1/2;
f(2)= x1.^2 - 81*(x2 + 0.1).^2 + sin(x3) + 1.06;
f(3)= exp(-x1*x2) + 20*x3 + (10*pi - 3)/3;
```

f313.m

```
function jf=f313(v)
x1=v(1);
x2=v(2);
x3=v(3);
% set jf matrix to size required by fuction newtonmv
jf=zeros(3,3);
% each row of jf is assigned the appropriate partial derivatives
jf(1,:)=[3 x3*sin(x2*x3) x2*sin(x2*x3)];
jf(2,:)=[2*x1 -162*(x2 + 0.1) cos(x3)];
jf(3,:)=[-x2*(exp(-x1*x2)) -x1*(exp(-x1*x2)) 20];
```

9.2 FUNÇÕES DO 5º. SISTEMA DE EQUAÇÕES

f314.m

```
function f=f314(v)
global m1 m2 A aa fi
a1=v(1);
a2=v(2);
Y1=v(3);
Y2=v(4);
% set of vector to size required by function newtonmv
f=zeros(4,1);
f(1)=-m1*a1 -(A*a2+aa*a2.^3)*sin(Y1) -fi*a2*sin(Y2);
f(2)=-m2*a2 +1/2*a1*sin(Y1);
f(3)=(A*a2 +aa*a2.^3)*cos(Y1) +fi*a2*cos(Y2);
f(4)=1/2*a1*cos(Y1);
```

f315.m

```
global m1 m2 A aa fi
a1=v(1);
a2=v(2);
Y1=v(3);
Y2=v(4);
% set jf matrix to size required by fuction newtonmv
jf=zeros(4,4);
% each row of jf is assigned the appropriate partial derivatives
jf(1,:)=[-m1 -sin(Y1)*(A+3*aa*a2.^2)
+cos(Y1)*(A*a2+aa*a2.^3) -fi*(sin(Y2)+a2*cos(Y2))];
jf(2,:)=[-m2 1/2*sin(Y1) +1/2*a1*cos(Y1) 1];
jf(3,:)=[1 cos(Y1)*(A+3*aa*a2.^2) -sin(Y1)*(A*a2+aa*a2.^3) fi*(cos(Y2)-a2*sin(Y2))];
jf(4,:)=[1 1/2*cos(Y1) -1/2*a1*sin(Y1) 1];
```

f315.m

```
function jf=f315(v)
```

9.3 ARQUIVO DE LOTE PARA TESTAR MÉTODO DE NEWTON:

Tnewton para ex.01:

```
function fun=newton01(tol)
```

```
sol=newton([3;-1.5], 'f308', 'f309', 2, tol);
disp('Tolerancia Informada:')
```

```
disp(tol)
disp('Raizes do sistema : ')
disp(sol)
```

```
disp(' Validação : ')
val=f308(sol);
disp(val)
```

tnewton para ex.02:

```
function fun=tnewton02(tol)
sol=newton([0;2;2], 'f310', 'f311', 3, tol);
disp(' Tolerancia Informada:')
disp(tol)
```

```
disp('Raizes do sistema : ')
disp(sol)
disp(' Validação : ')
val=f310(sol);
disp(val)
```

tnewton para ex.03:

```
function fun=tnewton03(tol)
sol=newton([0.1;0.1;-0.1], 'f312', 'f313', 3, tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validação : ')
val=f312(sol);
disp(val)
```

tnewton para ex.04:

```
function fun=tnewton04(tol)
sol=newton([0;0;0], 'f312', 'f313', 3, tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validação : ')
val=f312(sol);
disp(val)
```

tnewton para ex.05:

```
function fun=tnewton05(tol)
global m1 m2 A aa fi
m1 = 1.2334;
m2 = m1;
A = -11.4792;
aa = 0.4972;
fi = 0.0235;
% x=[0.1;0.1;1;1]; % a1,a2, y1, y2
% x=[1;0.1;2;1]; % a1,a2, y1, y2
x=[0.9;0.2;1.8;1.7]
```

```
sol=newton(x, 'f314', 'f315', 4, tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
disp(sol)
disp(' Validação : ')
val=f314(sol);
disp(val)
```

9.4 ARQUIVO DE LOTE PARA TESTAR MÉTODO QUASE NEWTON:

tquasnew para ex.01:

```
function fun=tquasnew01(tol)
sol=quasnew([3;-1.5], 'f308', 'f309', 2, tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validação : ')
val=f308(sol);
disp(val)
```

tquasnew para ex.02:

```
function fun=tquasnew02(tol)
sol=quasnew([0;2;2], 'f310', 'f311', 3, tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validação : ')
val=f310(sol);
disp(val)
```

tquasnew para ex.03:

```
function fun=tquasnew03(tol)
sol=quasnew([0.1;0.1;-0.1],'f312','f313',3,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validacao : ')
val=f312(sol);
disp(val)
```

tquasnew para ex.04:

```
function fun=tquasnew04(tol)
sol=quasnew([0;0;0],'f312','f313',3,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validacao : ')
val=f312(sol);
disp(val)
```

tquasnew para ex.05:

```
function fun=tquasnew05(tol)
global m1 m2 A aa fi
m1 = 1.2334;
m2 = m1;
A = -11.4792;
aa = 0.4972;
fi = 0.0235;
% x=[0.1;0.1;1;1]; % a1,a2, y1, y2
% x=[1;0.1;2;1]; % a1,a2, y1, y2
```

```
x=[0.9;0.2;1.8;1.7]
sol=quasnew(x,'f314','f315',4,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
disp(sol)
disp(' Validacao : ')
val=f314(sol);
disp(val)
```

9.5 ARQUIVO DE LOTE PARA TESTAR MÉTODO DE BROYDEN:

tbroyden para ex.01:

```
function fun=tbroyden01(tol)
sol=broyden([3;-1.5],'f308',2,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
disp(sol)
```

```
disp(' Validacao : ')
val=f308(sol);
disp(val)
```

tbroyden para ex.02:

```
function fun=tbroyden02(tol)
sol=broyden([0;2;2],'f310',3,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validacao : ')
val=f310(sol);
disp(val)
```

tbroyden para ex.03:

```
function fun=tbroyden03(tol)
sol=broyden([0.1;0.1;-0.1],'f312',3,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validacao : ')
val=f312(sol);
disp(val)
```

tbroyden para ex.04:

```
function fun=tbroyden04(tol)
sol=broyden([0;0;0],'f312',3,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validacao : ')
val=f312(sol);
disp(val)
```

tbroyden para ex.05:

```
function fun=tbroyden05(tol)
global m1 m2 A aa fi
m1 = 1.2334;
m2 = m1;
A = -11.4792;
aa = 0.4972;
fi = 0.0235;
% x=[0.1;0.1;1;1]; % a1,a2, y1, y2
% OK x=[1;0.1;2;1]; % a1,a2, y1, y2
```

```
x=[0.9;0.2;1.8;1.7]
sol=broyden(x,'f314',4,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
disp(sol)
disp(' Validação : ')
val=f314(sol);
disp(val)
```

9.6 ARQUIVO DE LOTE PARA TESTAR MÉTODO STEEPEST DESCENT:

tsteepest para ex.01:

```
function fun=tsteepest01(tol)
sol=steepest([3;-1.5],'f308','f309',2,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validação : ')
val=f308(sol);
disp(val)
```

tsteepest para ex.02:

```
function fun=tsteepest02(tol)
sol=steepest([0;2;2],'f310','f311',3,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validação : ')
val=f310(sol);
disp(val)
```

tsteepest para ex.03:

```
function fun=tsteepest03(tol)
sol=steepest([0.1;0.1;-0.1],'f312','f313',3,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validação : ')
val=f312(sol);
disp(val)
```

tsteepest para ex.04:

```
function fun=tsteepest04(tol)
sol=steepest([0;0;0],'f312','f313',3,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
```

```
disp(sol)
disp(' Validação : ')
val=f312(sol);
disp(val)
```

tsteepest para ex.05:

```
function fun=tsteepest05(tol)
global m1 m2 A aa fi
m1 = 1.2334;
m2 = m1;
A = -11.4792;
aa = 0.4972;
fi = 0.0235;
% x=[0.1;0.1;1;1]; % a1,a2, y1, y2
% OK x=[1;0.1;2;1]; % a1,a2, y1, y2
```

```
x=[0.9;0.2;1.8;1.7]
sol=steepest(x,'f314','f315',4,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp('Raizes do sistema : ')
disp(sol)
disp(' Validação : ')
val=f314(sol);
disp(val)
```


9.7 ARQUIVO DE LOTE PARA TESTAR MÉTODO GRADIENTES CONJUGADOS:

tgradconj para ex.01:

```
function fun=tgradconj01(tol)
global f derf;
n=2;
x=[3;-1.5];
f = 'f308';
derf = 'f309';
a=feval(derf,x)
b=feval(f,x)
```

```
sol=gradconj(x,a,b,n,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp(' Raizes do sistema: ')
disp(sol)
disp(' Validação : ')
val=f308(sol);
disp(val)
```

tgradconj para ex.02:

```
function fun=tgradconj02(tol)
global f derf;
n=3;
x=[0;2;2];
f = 'f310';
derf = 'f311';
a=feval(derf,x)
b=feval(f,x)
```

```
sol=gradconj(x,a,b,n,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp(' Raizes do sistema: ')
disp(sol)
disp(' Validação : ')
val=f310(sol);
disp(val)
```

tgradconj para ex.03:

```
function fun=tgradconj03(tol)
global f derf;
n=3;
x=[0.1;0.1;-0.1];
f = 'f312';
derf = 'f313';
a=feval(derf,x);
b=feval(f,x);
```

```
sol=gradconj(x,a,b,n,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp(' Raizes do sistema: ')
disp(sol)
disp(' Validação : ')
val=f312(sol);
disp(val)
```

tgradconj para ex.04:

```
function fun=tgradconj04(tol)
global f derf
n=3;
x=[0;0;0];
f = 'f312';
derf = 'f313';
a=feval(derf,x);
b=feval(f,x);
```

```
sol=gradconj(x,a,b,n,tol);
disp(' Tolerancia Informada:')
disp(tol)
disp(' Raizes do sistema : ')
disp(sol)
disp(' Validação : ')
val=f312(sol);
disp(val)
```

tgradconj para ex.05:

```
function fun=tgradconj05(tol)
global derf f;
global m1 m2 A aa fi
m1 = 1.2334;
m2 = m1;
A = -11.4792;
aa = 0.4972;
fi = 0.0235;
% OK x=[1;0.1;2;1]; % a1,a2, y1, y2
% x=[0.1;0.1;1;1]; % a1,a2, y1, y2
% x=[0;0.1;2;1.5];
% x=[0.5;0.5;2;2];
```

```
% x=[1;1;2;1]
x=[0.9;0.2;1.8;1.7]
n=4;
f = 'f314';
derf = 'f315';
k=0;
jx=feval(derf,x);
Aa=jx;
gk=feval(f,x);
b=gk;
sol=gradconj(x,Aa,b,n,tol);
disp(' Tolerancia Informada:')
```

```

disp(tol)
disp('Raizes do sistema : ')
disp(sol)
disp(' Validação : ')
val=f314(sol);
disp(val)

```

10. AIGUNS COMENTÁRIOS EM ALGEBRA LINEAR COMPUTACIONAL

Neste item apresenta-se, brevemente, uma revisão de algebra liunear computacional, lecionada, normalmente, na disciplinba cálculo numérico, com o obejetivo de complementar este texto.

A resolução de sistemas lineares, surge nas mais diversas áreas do conhecimento. Sabe-se que um sistema linear com m equações e n variáveis é escrito usualmente na forma:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$

onde

a_{ij} : coeficientes $1 \leq i \leq m, 1 \leq j \leq n$; x_j : variáveis, $j = 1, \dots, n$; b_i : constantes $i = 1, \dots, m$

A resolução de um sistema linear, consiste no calculo dos valores de $x_j, (j = 1, \dots, n)$, caso eles existam e que eles satisfaçam as m equações simultaneamente.

Usando-se a notação matricial, o sistema linear pode ser assim representado: $Ax=b$ onde A é a matriz dos coeficientes, x é o vetor das variáveis e b é o vetor constante. Então, resolver o sistema linear $Ax = b$ consiste em “dado $b \in \mathfrak{R}_m$ obter, caso exista, $x \in \mathfrak{R}^n$, tal que $Ax=b$ ”.

Os métodos numéricos para a resolução de sistemas lineares podem ser divididos em dois grupos: métodos diretos e métodos iterativos.

Métodos Diretos são aqueles que, exceto por erros de arredondamento, fornecem a solução exata do sistema linear, caso ela exista, após um número finito de operações e,

Métodos Iterativos por sua vez, geram uma seqüência de vetores a partir de uma aproximação inicial $x^{(0)}$, e sob certas circunstâncias esta seqüência converge para a solução do sistema, caso ela exista.

Estudar-se-á, a seguir: alguns métodos iterativos, a começar pelo Método Iterativo de Gauss-Jacobi.

10.1 MÉTODO DE GAUSS-JACOBI

O método de Gauss-Jacobi, consiste em dado uma aproximação inicial $x^{(0)}$ deseja-se obter $x^{(1)}, x^{(2)}, \dots, x^{(k)}$, através da relação recursiva $x^{(k+1)} = Cx^{(k)} + g$.

Para que esse método tenha convergência assegurada, precisa-se que seja respeitado o Critério das Linhas, ou seja, tomando-se todos os elementos de cada linha, somando-os linha a linha, exceto o elemento da linha que fizer parte da diagonal e dividindo-se o resultado pelo elemento da diagonal \Rightarrow tem-se que obter um resultado $<$ que 1. Caso essa regra não seja satisfeita, deve-se permutar uma linha ou coluna, de modo que o Critério das Linhas seja satisfeito. Logo abaixo se encontra o algoritmo deste método

Seja o sistema linear:
$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$

Passo 1 . Dados Iniciais: **a** = matriz com os valores das equações do sistema linear
 b = matriz com os valores das aproximações iniciais de $x^{(0)}$
 e = tolerância a ser respeitada.

Passo 2. Verifica-se se as matrizes informadas estão corretas quanto às linhas e colunas

$$\begin{array}{l}
 \text{Passo 3.} \left\{ \begin{array}{l} x_1^{(k)} = b(1) \\ x_2^{(k)} = b(2) \\ x_3^{(k)} = b(3) \\ \vdots \\ x_n^{(k)} = b(n) \end{array} \right. \\
 \text{Passo 4.} \left\{ \begin{array}{l} x_1^{(k+1)} = 0 * x_1^{(k)} - \frac{a_{12}}{a_{11}} * x_2^{(k)} - \frac{a_{13}}{a_{11}} * x_3^{(k)} - \dots - \frac{a_{1n}}{a_{11}} * x_n^{(k)} \\ x_2^{(k+1)} = \frac{a_{21}}{a_{22}} x_1^{(k)} - 0 * x_2^{(k)} - \frac{a_{23}}{a_{22}} * x_3^{(k)} - \dots - \frac{a_{2n}}{a_{22}} * x_n^{(k)} \\ x_3^{(k+1)} = \frac{a_{31}}{a_{33}} x_1^{(k)} - \frac{a_{32}}{a_{33}} * x_2^{(k)} - 0 * x_3^{(k)} - \dots - \frac{a_{3n}}{a_{33}} * x_n^{(k)} \end{array} \right.
 \end{array}$$

Passo 5. Calculando $d_r^{(1)}$ tem-se:

$$\left\{ \begin{array}{l} r_1 = |x_1^{(K+1)} - x_1^{(k)}| \\ r_2 = |x_2^{(K+1)} - x_2^{(k)}| \\ r_3 = |x_3^{(K+1)} - x_3^{(k)}| \\ \vdots \\ r_n = |x_n^{(K+1)} - x_n^{(k)}| \end{array} \right. \Rightarrow d_r^{(1)} = \frac{\max_{1 \leq r \leq n} |r|}{\max_{1 \leq i \leq n} |x_i^{(k+1)}|}$$

Passo 6. Se $d_r < e \Rightarrow$ faça $\bar{x} = x_1$. FIM.

Passo 7. $x_0 = x_1$

Passo 8. $k = k + 1$

Volte ao passo 4.

Para que o Método de Gauss-Jacobi tenha convergência assegurada, é necessário que seja obedecido o Critério das Linhas, ou seja, tomando-se todos os elementos de cada linha, somando-os linha a linha, exceto o elemento da linha que fizer parte da diagonal, e dividindo o resultado pelo elemento da diagonal \Rightarrow tem-se que se obter um resultado *menor* que 1.

Se essa condição não seja satisfeita, deve-se permutar uma linha ou coluna, de tal modo que ao se checar novamente o Critério das Linhas, ele seja satisfeito. Muitas vezes encontra-se sistemas lineares que mesmo permutando-se linhas e colunas, o Critério das Linhas não é satisfeito, necessitando-se de inúmeras iterações

até que se obtenha um resultado satisfatório. Nesse caso, é necessário procurar um outro método iterativo que exija menos esforço computacional, como por exemplo o próximo método a ser estudado, o de Gauss-Seidel.

10.2 MÉTODO DE GAUSS- SEIDEL

Da mesma forma que no método de Gauss-Jacobi, no método de Gauss-Seidel o sistema linear $Ax=b$ é descrito na forma equivalente $x = Cx + g$ por separação diagonal. O Método de Gauss-Seidel consiste em dado x_0 , uma aproximação inicial, precisa-se obter x_1, x_2, \dots, x_k , através da relação recursiva da forma: $x^{(k+1)} = Cx^{(k)} + g$, sendo que no momento de se calcular $x_j^{(k+1)}$ usa-se todos os valores de k já calculados anteriormente. Da mesma maneira que em Gauss-Jacobi, aqui é necessário que seja satisfeito o critério das linhas. Pode-se observar abaixo, o algoritmo desse método.

Seja o sistema linear:
$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{cases}$$

Passo 1. Dados Iniciais: **a** = matriz com os valores das equações do sistema linear
b = matriz com os valores das aproximações iniciais de $x^{(0)}$
e = tolerância a ser respeitada.

2). Verifica-se se as matrizes informadas estão corretas quanto às linhas e colunas

3).
$$\begin{cases} x_1^{(k)} = b(1) \\ x_2^{(k)} = b(2) \\ x_3^{(k)} = b(3) \\ \vdots \\ x_n^{(k)} = b(n) \end{cases}$$

4).
$$\begin{cases} x_1^{(k+1)} = 0 * x_1^{(k)} - \frac{a_{12}}{a_{11}} * x_2^{(k)} - \frac{a_{13}}{a_{11}} * x_3^{(k)} - \dots - \frac{a_{1n}}{a_{11}} * x_n^{(k)} \\ x_2^{(k+1)} = \frac{a_{21}}{a_{22}} x_1^{(k+1)} - 0 * x_2^{(k)} - \frac{a_{23}}{a_{22}} * x_3^{(k)} - \dots - \frac{a_{2n}}{a_{22}} * x_n^{(k)} \\ x_3^{(k+1)} = \frac{a_{31}}{a_{33}} x_1^{(k+1)} - \frac{a_{32}}{a_{33}} * x_2^{(k+1)} - 0 * x_3^{(k)} - \dots - \frac{a_{3n}}{a_{33}} * x_n^{(k)} \end{cases}$$

3). $x^{(k+1)} = Cx^{(k)} + g$

4). Calculando $d_r^{(1)}$ tem-se:

$$\left\{ \begin{array}{l} r_1 = |x_1^{(K+1)} - x_1^{(k)}| \\ \vdots \end{array} \right.$$

$$\begin{aligned}
r_2 &= |x_2^{(K+1)} - x_2^{(k)}| \\
r_3 &= |x_3^{(K+1)} - x_3^{(k)}| \\
&\vdots \\
r_n &= |x_n^{(K+1)} - x_n^{(k)}| \Rightarrow d_r^{(1)} = \frac{\max_{1 \leq r \leq n} |r|}{\max_{1 \leq i \leq n} |x_i^{(k+1)}|}
\end{aligned}$$

5). Se $d_r < \epsilon \Rightarrow$ faça $\bar{x} = x_1$. FIM.

6). $x_0 = x_1$

7). $k = k + 1$

Volte ao passo 4.

10.3 PROGRAMAÇÃO LINEAR

Programação Linear é normalmente considerada como um método de pesquisa operacional, mas existe uma série muito grande de aplicações. O problema que será exposto pode ser expresso em sua forma padrão como :

$$\begin{array}{ll}
\text{Minimizar } C^T x & \text{“função objetivo”} \\
\text{sujeita a } Ax = b & \text{“equações de restrição” onde } x \geq 0
\end{array}$$

Onde x é o vetor coluna dos n componentes que se deseja determinar. As constantes dadas do sistema são fornecidas pelo vetor coluna b , uma matriz: $A \ m \times n$ e um vetor coluna c . Todas as equações e a função que se deseja minimizar estão na forma linear.

O problema é de otimização e geralmente representa a necessidade de minimizar uma função linear $c^T x$, chamada de função objetivo, sujeita a satisfazer um sistema de desigualdades linear. Apesar de ter sido dada a forma “standard”, muitas outras formas deste problema, podem aparecer, as quais são facilmente convertidas a esta considerada.

Por exemplo, as restrições podem ser inicialmente desigualdades e, estas podem ser facilmente convertidas em igualdades, adicionando-se ou subtraindo-se variáveis adicionais introduzidas ao problema, em questão. O objetivo pode ser maximizar a função, em vez de minimizá-la. Novamente isto é facilmente obtido alterando-se os sinais dos coeficientes c .

Alguns exemplos práticos onde a Programação Linear pode ser aplicada são:

- problema de dietas alimentares em hospitais, requerendo redução de custos de alimentos, enquanto permanece-se oferecendo a melhor dieta.
- Problema de redução de perda padrão em indústrias.
- Problema de se otimizar o lucro, sujeito a restrições de disponibilidade de materiais.
- Problema de otimização de rotinas de chamadas telefônicas.

Um importante algoritmo numérico para solucionar esses problemas é chamado de Método Simplex e foi introduzido por Dantzig (1963).

10.3 -1 MÉTODO SIMPLEX DE DANTZIG

Este algoritmo foi aplicado em problemas de guerra, relacionados com problemas de distribuição de materiais. Sem descrever em detalhes como ele funciona, o princípio geral de seu funcionamento é o de gerar uma sequência de pontos, os quais correspondem matematicamente aos vértices de uma região multidimensional ótima. O algoritmo vai de um vértice a outro, a cada vez aumentando o valor da função objetivo, até que o resultado ótimo seja obtido. Todos esses pontos estão na área da região provável, e para problemas maiores, existirão um grande número desses pontos. O Método Simplex de Dantzig pode ser melhor ilustrado considerando-se um problema simples de programação linear. Tome-se como exemplo uma indústria de componentes eletrônicos, onde tem-se:

- $\left\{ \begin{array}{l} x_1 \Rightarrow \text{conjunto de resistores} \Rightarrow \text{para cada } x_1, \text{ ganha-se 7 de lucro.} \\ x_2 \Rightarrow \text{conjunto de capacitores} \Rightarrow \text{para cada } x_2, \text{ ganha-se 13 de lucro} \end{array} \right.$

- $\left\{ \begin{array}{l} \text{Cada um dos 2 conjuntos serão produzidos em 2 estágios} \\ \text{O estágio 1 é limitado em 18 unidades tempo por semana} \\ \text{O estágio 2 é limitado em 54 unidades tempo por semana} \\ \text{Um banco de resistores necessita de 1 unidade de tempo do estágio 1 e de 2 do estágio 2.} \\ \text{Um banco de capacitores necessita de 3 unidades de tempo do estágio 1 e de 6 do estágio 2} \end{array} \right.$

O objetivo da indústria é maximizar o lucro dado as limitações de fabricação.
A equação a ser maximizada é a do lucro, que obtem-se a partir de 1.

$$z = \text{lucro}$$

$$1z = 7x_1 + 13x_2$$

O problema é maximizar essa equação, a qual está sob controle das equações obtidas a partir de 2:

$$1x_1 + 3x_2 \leq 18$$

$$5x_1 + 6x_2 \leq 54$$

$$5(18 - 3x_2) + 6x_2 = 54$$

Resolvendo o sistema de equações, tem-se: $x_1 = 18 - 3x_2$ substituindo-se: $90 - 15x_2 + 6x_2 = 54$

$$x_2 = 4$$

$$x_1 = 18 - 3(4) \quad z = 7 * 6 + 13 * 4$$

$$x_1 = 18 - 12 \Rightarrow z = 42 + 52$$

$$x_1 = 6 \quad z = 94$$

Apesar de se a solução de um problema simples de duas variáveis os problemas de programação linear muitas vezes envolvem milhares de centenas de variáveis.

Para esses problemas práticos, um algoritmo numérico bem especificado se faz necessário, o que é fornecido pelo algoritmo Simplex de Dantzig. Entretanto precisa-se considerar desenvolvimentos mais recentes que são teoricamente melhores, baseados nos trabalhos de Karmakar (1948).

10-3. 2 - MÉTODO PONTO INTERIOR – KARMAKAR

Karmakar gerou um algoritmo em princípio bem diferente do algoritmo de Dantzig. Enquanto a complexidade teórica do Método de Dantzig é exponencial em número de variáveis do problema, algumas versões do algoritmo de Karmakar têm uma complexidade a qual é da ordem do cubo de número de variáveis.

O algoritmo proposto por Karmakar trabalha com o problema da programação linear de outro modo. Este algoritmo transforma o problema em uma forma mais conveniente e então pesquisa dentro da região provável, usando uma boa direção de pesquisa através de sua superfície.

Pelo fato desse algoritmo usar pontos interiores, ele é comumente chamado de Método do Ponto Interior.

Desde sua formulação, muitos desenvolvimentos e modificações têm sido feitos a este algoritmo. Aqui vai-se tratar de uma forma que apesar de conceitualmente complexa, leva a um notavelmente simples e elegante algoritmo de programação linear, ou seja, a formulação dada por Barnes (1986). Vale observar que o algoritmo de Barnes, apesar de prover uma modificação ao algoritmo de Karmakar, preserva seus princípios fundamentais.

10- 3. 3 - MÉTODO DE BARNES

O algoritmo de Barnes pode ser aplicado a qualquer problema de programação linear, desde que ele seja convertido à forma Simplex. Entretanto, uma importante modificação inicial é necessária para assegurar que o algoritmo inicie em um ponto interior $x_0 \leq 0$.

Esta modificação é obtida introduzindo-se uma nova última coluna à matriz A, cujos elementos são o vetor b menos a soma das colunas da matriz A.

Associa-se então uma variável adicional a esta coluna adicional de modo que não se tenha uma variável supérflua na solução. Introduce-se um elemento extra no vetor c e faz-se o valor desse elemento bem grande o suficiente para fazer com que essa nova variável tenda a zero, quando o ótimo for obtido. O algoritmo pode se apresentar, como a seguir:

Passo 1. Assumindo n variáveis no problema original, faça-se:

$$a(i, n+1) = b(i) - \sum_j a(i, j) \text{ e}$$

$$c(n+1) = 10000$$

$$x^0 = [1 \ 1 \dots 1]$$

$$k = 0$$

Passo 2. $D^k = \text{diag}(x^k)$ e obtenha um ponto melhor usando a equação:

$$x^{k+1} = x^k - s(D^k)^2 (c - A^T I^k) / \text{norm}(D^k)(c - A^T I^k))$$

onde

$$I^k = (A(D^k)^2 A^T)^{-1} A(D^k)^2 c$$

o passo s é dado por:

$$s = \min \left\langle \text{norm} \left((D^k) (c - A^T I^k) \right) / \left[x_j^k (c_j - A_j^T I^k) \right] \right\rangle - a$$

onde

A_j é a j ésima coluna da matriz A e

a é o valor de uma pequena constante pré-fixada. Aqui o

mínimo é dado somente pelos valores $(c_j - A_j^T I^k) > 0$.

Nota-se também que I^k provê uma aproximação para a solução do problema.

Passo 3. Pare o processamento se os valores das funções objetivos são aproximadamente iguais.

Se não o forem, faça $k = k + 1$ e repita a partir do passo 2

11. ROTINAS DE ALGEBRA LINEAR COMPUTACIONAL

11.1 Método de Gauss-Jacobi para sistemas lineares

pmgasjob.m

```
function PMGASJOB(a,b,e)
% Metodo Iterativo de Gauss-Jacobi p/obtencao de raizes de
% equacoes de sistemas lineares
% as equacoes podem ter ate 3 variaveis: x1,x2 e x3
% a = possui a matriz com os valores das equacoes do sistema
% linear
% b = possui a matriz com os valores da aproximacao inicial
% x0
% e = tolerancia a ser respeitada
%
if nargin < 3
    error('Numero de Argumentos Insuficientes')
end
%
wfim=0;
% Vamos ver qual e' o tamanho da matriz do sist.linear
wtotcol=size(a,2); % descubro quantas colunas tem a matriz
informada
wtotlin=size(a,1); % descubro quantas linhas tem a matriz
informada
waux=wtotlin+1;
if wtotcol ~= waux
    error('Matriz do Sistema Informada nao esta com Total
Colunas = Linhas + 1 ')
    wfim=1
end
% b tem que ter somente 1 coluna, e o total de linhas = total de
linhas da matriz a
if size(b,2) ~= 1
    error('Matriz com as aprox.iniciais tem mais de uma
coluna')
    wfim=1
end
if size(b,1) ~= size(a,1)
    error('Matriz com as aprox.iniciais nao tem mesmo
num.linha da matriz do sist.linear')
    wfim=1
end
%
xk0(1,1)=b(1);
xk0(2,1)=b(2);
xk0(3,1)=b(3);
wfim=0;
wxk=b;
witer=0;
wcontador=0;
while wfim==0
    wcontador=wcontador+1;
    if wcontador > 30
        witer=wcontador;
        wxk(1,wcontador)=xk0(1,1);
        wxk(2,wcontador)=xk0(2,1);
        wxk(3,wcontador)=xk0(3,1);
        wfim=2;
        break
    end
    waux1 = (a(1,1)* 0 ) /a(1,1); % = 0
    waux2 = (a(1,2)*xk0(2,1)) /a(1,1);
    waux3 = (a(1,3)*xk0(3,1)) /a(1,1);
    waux4 = a(1,4) /a(1,1);
    xk1(1,1) = waux4 - waux1 - waux2 - waux3;
    %
    waux1 = (a(2,1)*xk0(1,1)) /a(2,2);
    waux2 = (a(2,2)* 0 ) /a(2,2); % = 0
    waux3 = (a(2,3)*xk0(3,1)) /a(2,2);
    waux4 = a(2,4) /a(2,2);
    xk1(2,1) = waux4 - waux1 - waux2 - waux3;
    %
    waux1 = (a(3,1)*xk0(1,1)) /a(3,3);
    waux2 = (a(3,2)*xk0(2,1)) /a(3,3);
    waux3 = (a(3,3)* 0 ) /a(3,3);
    waux4 = a(3,4) /a(3,3);
    xk1(3,1) = waux4 - waux1 - waux2 - waux3;
    %
    wres(1,1) = xk1(1,1) - xk0(1,1);
    wres(2,1) = xk1(2,1) - xk0(2,1);
    wres(3,1) = xk1(3,1) - xk0(3,1);
    wdistvet=max(abs(wres));
    wxmax=max(abs(xk1));
    dr=wdistvet/wxmax;
    if dr < e
        witer=wcontador;
        wxk(1,wcontador)=xk1(1,1);
        wxk(2,wcontador)=xk1(2,1);
        wxk(3,wcontador)=xk1(3,1);
        wfim=1;
        break
    end
    witer=wcontador;
    wxk(1,wcontador)=xk1(1,1);
    wxk(2,wcontador)=xk1(2,1);
    wxk(3,wcontador)=xk1(3,1);
    xk0(1,1)=xk1(1,1);
    xk0(2,1)=xk1(2,1);
    xk0(3,1)=xk1(3,1);
end
save PMGASJOB wxk witer;
disp('PMGASJOB-Pgma do Metodo Gauss-Jacobi - para
sist.lineares com 3 incognitas')
disp('Matriz do Sistema Linear informado:')
a
disp('Matriz das Aproximações Iniciais informada :')
b
disp('Precisao Informada:')
fprintf('e=%1.14f \n',e)
```



```
disp('Arquivo PMGASJOBxx.m - contem os displays da
execucao')
disp('Arquivo PMGASJOBxx.mat - contem WITER-numero
de iterações e WXX-valores aprox.de x')
```

```
disp(['Achamos o valor aproximado com ',num2str(witer),'
iteracoes'])
disp('Valores encontrados para x, a partir do valor inicial:')
wxx
```

A seguir, ilustra-se a aplicação do programa , logo acima, para os problemas:

$$\begin{cases} 10x_1 + 2x_2 + x_3 = 7 \\ x_1 + 5x_2 + x_3 = -8 \\ 2x_1 + 3x_2 + 10x_3 = 6 \end{cases} \quad e \quad x^{(0)} = \begin{cases} 0.7 \\ -1.6 \\ 0.6 \end{cases} \quad e \quad \epsilon = 0.05$$

pmgasjob01.m

PMGASJOB-Pgma do Metodo Gauss-Jacobi - para
sist.lineares com 3 incognitas

Matriz do Sistema Linear informado:

```
a = 10  2  1  7
     1  5  1 -8
     2  3 10  6
```

Matriz das Aproximações Iniciais informada :

```
b = 0.7000
    -1.6000
```

0.6000

Precisao Informada:

e=0.0500000000000000

Arquivo PMGASJOB01.m - contem os displays da execucao

Arquivo PMGASJOB01.mat - contem WITER-numero de
iteraões e WXX-valores aprox.de x

Achamos o valor aproximado com 3 iteracoes

Valores encontrados para x, a partir do valor inicial:

```
wxx = 0.9600  0.9780  0.9994
      -1.8600 -1.9800 -1.9888
      0.9400  0.9660  0.9984
```

$$\begin{cases} x_1 + 3x_2 + x_3 = -2 \\ 5x_1 + 2x_2 + 2x_3 = 36x_2 + 8x_3 = -6 \\ 6x_2 + 8x_3 = -6 \end{cases}$$

Pode-se observar que neste exercício o Critério das Linhas não foi satisfeito, e foi interrompido o programa após 30 iterações.

pmgasjob02.m

pmgasjob(a,b,e)

PMGASJOB-Pgma do Metodo Gauss-Jacobi - para
sist.lineares com 3 incognitas

Matriz do Sistema Linear informado:

```
a = 1  3  1 -2
     5  2  2  3
     0  6  8 -6
```

Matriz das Aproximações Iniciais informada :

```
b = 0
     0
     0
```

Precisao Informada:

e=0.0500000000000000

Arquivo PMGASJOB02.m - contem os displays da execucao

Arquivo PMGASJOB02.mat - contem WITER-numero de
iteraões e WXX-valores aprox.de x

Achamos o valor aproximado com 31 iteracoes

Valores encontrados para x, a partir do valor inicial:

```
wxx = 1.0e+013 *
```

Columns 1 through 7

```
-0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000
 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
-0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000
```

Columns 8 through 14

```
-0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
-0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000
```

Columns 15 through 21

```
-0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0002
0.0000 0.0000 0.0000 0.0000 0.0000 0.0001 0.0001
-0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0001
```

Columns 22 through 28

```
-0.0002 -0.0020 -0.0015 -0.0157 -0.0084 -0.1268 -0.0398
0.0007 0.0007 0.0054 0.0041 0.0433 0.0241 0.3494
-0.0001 -0.0005 -0.0005 -0.0040 -0.0031 -0.0325 -0.0181
```

Columns 29 through 31

```
-1.0303 -0.0908 -0.0908
0.1176 2.8377 2.8377
-0.2621 -0.0882 -0.0882
```

$$\begin{cases} x_1 + 3x_2 + x_3 = -2 \\ 5x_1 + 2x_2 + 2x_3 = 36x_2 + 8x_3 = -6 \\ 6x_2 + 8x_3 = -6 \end{cases}$$

Pode-se observar que nesta execução, foi permutada a segunda linha com a primeira e obtivemos o resultado em apenas 8 iterações.

pmgasjob03.m

pmgasjob(a,b,e)

PMGASJOB-Pgma do Metodo Gauss-Jacobi - para

```
a = 5 2 2 3
    1 3 1 -2
    0 6 8 -6
```

Matriz das Aproximações Iniciais informada :

Precisao Informada:

e=0.0500000000000000

Arquivo PMGASJOB03.m - contem os displays da execucao

Arquivo PMGASJOB03.mat - contem WITER-numero de iterações e WXX-valores aprox.de x

sist.lineares com 3 incognitas

Matriz do Sistema Linear informado:

```
b = 0
    0
    0
```

Achamos o valor aproximado com 8 iteracoes

Valores encontrados para x, a partir do valor inicial:

wxk =

```
Columns 1 through 7
Column 8
0.6000 1.1667 0.9467 1.1039 0.9629 1.0452
0.9754 1.0210
-0.6667 -0.6167 -0.9722 -0.8864 -1.0277 -0.9592 -
1.0220 -0.9816
-0.7500 -0.2500 -0.2875 -0.0208 -0.0852 0.0208 -
0.0306 0.0165
```

$$\begin{cases} 5x_1 + x_2 + x_3 = 5 \\ 3x_1 + 4x_2 + x_3 = 6 \quad 3x_1 + 3x_2 + 6x_3 = 0 \\ 3x_1 + 3x_2 + 6x_3 = 0 \end{cases}$$

Neste exemplo pode-se observar que o Critério das Linhas não é possível de ser satisfeito, mesmo permutando-se linhas ou colunas, e precisamos de 11 iterações para obter o resultado aproximado. Neste caso pode-se escolher um outro método iterativo que exija menos esforço computacional.

PMGASJOB-Pgma do Metodo Gauss-Jacobi - para
sist.lineares com 3 incognitas
Matriz do Sistema Linear informado:

pmgasjob04.m

pmgasjob(a,b,e)

```
a =  5   1   1   5
      3   4   1   6
      3   3   6   0
```

Matriz das Aproximações Iniciais informada :

```
b =  0
      0
      0
```

Precisao Informada:

e=0.050000000000000

Arquivo PMGASJOB04.m - contem os displays da execucao
Arquivo PMGASJOB04.mat - contem WITER-numero de
iteracoes e WXX-valores aprox.de x
wxk =

Columns 1 through 7

```
1.0000  0.7000  1.1000  0.8875  1.0675  0.9478  1.0366
1.5000  0.7500  1.2875  0.8562  1.1328  0.9173  1.0642
0 -1.2500 -0.7250 -1.1938 -0.8719 -1.1002 -0.9326
```

Columns 8 through 11

```
0.9737  1.0189  0.9865  1.0097
0.9557  1.0323  0.9770  1.0166
-1.0504 -0.9647 -1.0256 -0.9817
```

Achamos o valor aproximado com 11 iteracoes
Valores encontrados para x, a partir do valor inicial:

11.2 MÉTODO DE GAUSS-SEIDEL PARA SISTEMAS LINEARES

pmgassei.m

function PMGASSEI(a,b,e)

% Metodo Iterativo de Gauss-Seidel p/obtencao de raizes de
equacoes de sistemas lineares

% as equacoes podem ter ate 3 variaveis: x1,x2 e x3

% a = possui a matriz com os valores das equacoes do sistema
linear

% b = possui a matriz com os valores da aproximacao inicial
x0

% e = tolerancia a ser respeitada

%

if nargin < 3

error('Numero de Argumentos Insuficientes')

end

%

wfim=0;

% Vamos ver qual e' o tamanho da matriz do sist.linear

wtotcol=size(a,2); % descubro quantas colunas tem a matriz
informada

wtotlin=size(a,1); % descubro quantas linhas tem a matriz
informada

waux=wtotlin+1;

if wtotcol ~= waux

error('Matriz do Sistema Informada nao esta com Total
Colunas = Linhas + 1 ')

wfim=1;

end

% b tem que ter somente 1 coluna, e o total de linhas = total de
linhas da matriz a

if size(b,2) ~= 1

error('Matriz com as aprox.iniciais tem mais de uma
coluna')

wfim=1;

end

if size(b,1) ~= size(a,1)

error('Matriz com as aprox.iniciais nao tem mesmo
num.linha da matriz do sist.linear')

wfim=1;

```

end
%
xk0(1,1)=b(1);
xk0(2,1)=b(2);
xk0(3,1)=b(3);
wfim=0;
wxk=b;
witer=0;
wcontador=0;
while wfim==0
    wcontador=wcontador+1;
    if wcontador > 30
        witer=wcontador;
        wxk(1,wcontador)=xk1(1,1);
        wxk(2,wcontador)=xk1(2,1);
        wxk(3,wcontador)=xk1(3,1);
        wfim=2;
        break
    end
    waux1 = (a(1,1)* 0 ) /a(1,1); % = 0
    waux2 = (a(1,2)*xk0(2,1)) /a(1,1);
    waux3 = (a(1,3)*xk0(3,1)) /a(1,1);
    waux4 = a(1,4) /a(1,1);
    xk1(1,1) = waux4 - waux1 - waux2 - waux3;
    %
    waux1 = (a(2,1)*xk1(1,1)) /a(2,2);
    waux2 = (a(2,2)* 0 ) /a(2,2); % = 0
    waux3 = (a(2,3)*xk0(3,1)) /a(2,2);
    waux4 = a(2,4) /a(2,2);
    xk1(2,1) = waux4 - waux1 - waux2 - waux3;
    %
    waux1 = (a(3,1)*xk1(1,1)) /a(3,3);
    waux2 = (a(3,2)*xk1(2,1)) /a(3,3);
    waux3 = (a(3,3)* 0 ) /a(3,3);
    waux4 = a(3,4) /a(3,3);
    xk1(3,1) = waux4 - waux1 - waux2 - waux3;
    %
    wres(1,1) = xk1(1,1) - xk0(1,1);
    wres(2,1) = xk1(2,1) - xk0(2,1);

```

```

wres(3,1) = xk1(3,1) - xk0(3,1);
wdistvet=max(abs(wres));
wxmax=max(abs(xk1));
dr=wdistvet/wxmax;
if dr < e
    witer=wcontador;
    wxk(1,wcontador)=xk1(1,1);
    wxk(2,wcontador)=xk1(2,1);
    wxk(3,wcontador)=xk1(3,1);
    wfim=1;
    break
end
witer=wcontador;
wxk(1,wcontador)=xk1(1,1);
wxk(2,wcontador)=xk1(2,1);
wxk(3,wcontador)=xk1(3,1);
xk0(1,1)=xk1(1,1);
xk0(2,1)=xk1(2,1);
xk0(3,1)=xk1(3,1);
end
save PMGASSEI wxk witer;
disp('PMGASSEI-Pgma do Metodo Gauss-Seidel - para
sist.lineares com 3 incognitas')
disp('Matriz do Sistema Linear informado:')
a
disp('Matriz das Aproximações Iniciais informada :')
b
disp('Precisao Informada:')
fprintf('e=%1.14f \n',e)
disp('Arquivo PMGASSEIxx.m - contem os displays da
execucao')
disp('Arquivo PMGASSEIxx.mat - contem WITER-numero
de iterações e WXX-valores aprox.de x')
disp(['Achamos o valor aproximado com ',num2str(witer),'
iteracoes'])
disp('Valores encontrados para x, a partir do valor inicial:')
wxk

```

$$\begin{cases} 10x_1 + 2x_2 + x_3 = 7 \\ x_1 + 5x_2 + x_3 = -8 & 2x_1 + 3x_2 + 10x_3 = 6 \\ 2x_1 + 3x_2 + 10x_3 = 6 \end{cases}$$

Observa-se que o exemplo, em questão, quando resolvido utilizando-se o Método de Gauss-Jacobi necessitou de 3 iterações, enquanto que com este método, necessitou de apenas 2 iterações.

pmgassei01.m

PMGASSEI-Pgma do Metodo Gauss-Seidel - para sist.lineares
com 3 incognitas
Matriz do Sistema Linear informado:

```

a = 10  2  1  7
    1  5  1 -8
    2  3 10  6

```

Matriz das Aproximações Iniciais informada :

```

b = 0.700000000000000
   -1.600000000000000
    0.600000000000000

```

Precisao Informada:

```
e=0.050000000000000
```

Arquivo PMGASSEI01.m - contem os displays da execucao
Arquivo PMGASSEI01.mat - contem WITER-numero de
iteraões e WXX-valores aprox.de x

Achamos o valor aproximado com 2 iteracoes
Valores encontrados para x, a partir do valor inicial:

wxk =

0.9600000000000000	0.9842400000000000	-1.9120000000000000	-1.9931680000000000	0.9816000000000000	1.0011024000000000
--------------------	--------------------	---------------------	---------------------	--------------------	--------------------

$$\begin{cases} 5x_1 + x_2 + x_3 = 6 \\ 3x_1 + 4x_2 + x_3 = 6 \\ 3x_1 + 3x_2 + 6x_3 = 0 \end{cases}$$

Pode-se observar que enquanto no Método de Gauss-Jacobi nesse exemplo empregou-se 11 iterações para ser resolvido, aqui ele vai ser resolvido com apenas 3 iterações.

pmgassei02.m

PMGASSEI-Pgma do Metodo Gauss-Seidel - para sist.lineares com 3 incognitas

Matriz do Sistema Linear informado:

a =

5	1	1	5
3	4	1	6
3	3	6	0

Matriz das Aproximações Iniciais informada :

b =

0
0
0

Precisao Informada:

e=0.0500000000000000

Arquivo PMGASSEI02.m - contem os displays da execucao
Arquivo PMGASSEI02.mat - contem WITER-numero de iterações e WXX-valores aprox.de x
Achamos o valor aproximado com 3 iteracoes
Valores encontrados para x, a partir do valor inicial:

wxk =

Columns 1 through 2	Column 3
1.0000000000000000	1.0250000000000000
1.0075000000000000	0.9500000000000000
0.7500000000000000	
0.9912500000000000	
-0.8750000000000000	-0.9875000000000000
0.9993750000000000	

$$\begin{cases} 2x_1 + x_2 + 3x_3 = 9 \\ -x_2 + x_3 = 1 \\ x_1 + 3x_3 = 3 \end{cases}$$

Pode-se observar que a matriz informada ao programa para ser executado está com aparência diferente da forma em que se apresenta como exercício proposto. Não está detalhado aqui, mas submeteu-se o sistema de equações ao Critério das Linhas, o qual não foi satisfeito, permutou-se a primeira linha pela terceira, mas mesmo assim continuou não satisfazendo o Critério das Linhas. Em seguida permutou-se a primeira coluna pela terceira, quando somente então tivemos o Critério das Linhas satisfeito e o submetemos ao Método de Gauss-Seidel.

pmgassei03.m

pmgassei(a,b,e)

PMGASSEI-Pgma do Metodo Gauss-Seidel - para sist.lineares com 3 incognitas

Matriz do Sistema Linear informado:

a =

3	0	1	3
1	-1	0	1
3	1	2	9

Matriz das Aproximações Iniciais informada :

b =

0
0

0
Precisao Informada:
e=0.0500000000000000
Arquivo PMGASSEI03.m - contem os displays da execucao
Arquivo PMGASSEI03.mat - contem WITER-numero de iterações e WXX-valores aprox.de x
Achamos o valor aproximado com 6 iteracoes
Valores encontrados para x, a partir do valor inicial:

wxk =

1.0000	0	-0.6667	-1.1111	-1.4074	-1.6049
0	-1.0000	-1.6667	-2.1111	-2.4074	-2.6049
3.0000	5.0000	6.3333	7.2222	7.8148	8.2099

11.3 MÉTODO DE BARNES PARA PROGRAMAÇÃO LINEAR

pmbarnes.m

```
function [xsol,basic]=barnes(A,b,c,tol)
x2=[];x=[];[m n]=size(A)
%Set up initial problem
aplus1=b-sum(A,2)
cplus1=1000000
A=[A aplus1]
c=[c cplus1]
B=[]
n = n+1
x0=ones(1,n)'
x=x0
alpha= .0001
lambda=zeros(1,m)'
iter=0
%Main Step
while abs(c*x-lambda'*b)>tol
    x2=x.*x
    D= diag(x)
    D2=diag(x2)
    AD2= A*D2
    lambda=(AD2*A')\((AD2*c')
    dualres=c'-A'*lambda
    normres=norm(D*dualres)
    for i=1:n
        if dualres(i) > 0
            ratio(i)=normres/(x(i)*(c(i)-A(:,i)*lambda))
        else
            ratio(i)= inf
```

```
end
end
R=min(ratio)-alpha
x1=x-R*D2*dualres/normres
x=x1
basiscount=0
B=[]
basic=[]
cb=[]
for k=1:n
    if x(k)>tol
        basiscount=basiscount+1
        basic=[basic k]
    end
end
%Only used if problem non-degenerate
if basiscount==m
    for k=basic
        B=[B A(:,k)]
        cb=[cb c(k)]
    end
    primalsol=b'/B'
    xsol=primalsol
    break
end
iter=iter+1
end
objective=c*x
```

A seguir, exemplifica-se , para o caso:

$$\text{Maximizar } z = 2x_1 + x_2 + 4x_3$$

função objetivo

$$\begin{aligned} \text{Sujeita a } x_1 + x_2 + x_3 &\leq 7 \\ x_1 + 2x_2 + 3x_3 &\leq 12 \end{aligned}$$

inequações de restrição

Este problema de programação linear pode ser facilmente transformado na forma padrão:

- adicionando-se novas variáveis positivas – chamadas variáveis fracas – ao lado esquerdo das inequações de restrição;
- e trocando-se o sinal dos coeficientes da função objetivo;

de tal modo que o problema fica convertido m um problema de minimização sujeito à equações de restrição tal como:

$$\text{Mnimizar } -z = -2x_1 - x_2 - 4x_3$$

função objetivo

$$\begin{aligned} \text{Sujeita a } x_1 + x_2 + x_3 + x_4 &= 7 \\ x_1 + 2x_2 + 3x_3 + x_5 &= 12 \end{aligned}$$

equações de restrição

Variáveis fracas

As variáveis x_4 e x_5 são chamadas “variáveis fracas”, e elas representam a diferença entre os recursos disponíveis e os recursos utilizados. Se as inequações de restrição forem ≥ 0 , então subtraímos as variáveis fracas, para produzir igualdades. Para executar o método, criamos o seguinte Arquivo de Lote:

```
pm Barnes01.m
c=[-2 -1 -4 0 0];
a=[1 1 1 1 0;1 2 3 0 1];
b=[7;12];
[xsol,ind]=pm Barnes(a,b,c,0.00005);
i=1;

fprintf("\nSolution is: ");
for j=ind
    fprintf("\nx(%1.0f)=%8.4f",j,xsol(i));
    i=i+1;
end
fprintf("\nOther variables are zero\n")
```

E obtive-se o seguinte resultado:

```
pm Barnes01s.m
objective =
-19.0000

Solution is:

x(1)= 4.5000
x(3)= 2.5000
Other variables are zero
```

$$\text{Maximizar } F = x_2 - x_1$$

$$\text{Subordinada a } \begin{cases} -x_1 + 2x_2 \leq 2 \\ x_1 + x_2 \leq 4 \\ x_1 \leq 3 \end{cases} \quad \text{sendo } \begin{cases} x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$$

No Exemplo acima tínhamos duas inequações de restrição, então introduzimos 2 variáveis fracas. Neste exemplo tem-se 3 inequações de restrição e vamos introduzir três variáveis fracas.

$$\begin{cases} -x_1 + 2x_2 + x_3 = 2 \\ x_1 + x_2 + x_4 = 4 \\ x_1 + x_5 = 3 \end{cases} \quad \text{variáveis fracas}$$

Trocando o sinal da função objetivo: $F = -x_2 + x_1$, Utilizou-se o seguinte Arquivo de Lote para executar o método de Barnes:

```
pm Barnes02.m
c=[1 -1 0 0 0];
a=[-1 2 1 0 0;1 1 0 1 0;1 0 0 0 1];
b=[2;4;3];
[xsol,ind]=pm Barnes(a,b,c,0.00005);
i=1;

fprintf("\nSolution is: ");
for j=ind
    fprintf("\nx(%1.0f)=%8.4f",j,xsol(i));
    i=i+1;
end
fprintf("\nOther variables are zero\n")
```

E obtivemos o seguinte resultado:

```
pm Barnes02s.m
objective =
-1.0000

Solution is:
x(2)= 1.0000
x(4)= 3.0000
x(5)= 3.0000
Other variables are zero
```

$$\text{Minimizar } H = 2y_1 + 4y_2 + 3y_3$$

$$\text{Subordinada a } \begin{cases} y_1 - y_2 - y_3 \leq 1 \\ -2y_1 - y_2 \leq -1 \end{cases} \quad \text{sendo } y_1, y_2, y_3 \geq 0$$

Neste caso, introduzse 2 variáveis fracas nas inequações de restrição:

$$\begin{cases} y_1 - y_2 - y_3 + y_4 = 1 \\ -2y_1 - y_2 + y_5 = -1 \end{cases}$$

Criou-se o seguinte Arquivo de Lote para executar o exercício:

```
pmbarnes03.m
c=[2 4 3 0 0];
a=[1 -1 -1 1 0;-2 -1 0 0 1];
b=[1;-1];
[xsol,ind]=pmbarnes(a,b,c,0.00005);
i=1;
fprintf('\nOther variables are zero\n')
```

```
fprintf('\nSolution is: ');
for j=ind
    fprintf('\nx(%1.0f)=%8.4f',j,xsol(i));
    i=i+1;
end
```

E obtivemos o seguinte resultado:

```
pmbarnes03s.m
objective = 1.0073
```

```
x(1)= 0.5000
x(4)= 0.5000
Other variables are zero
```

Solution is:

11.4 MÉTODO DE GAUSS-JACOBI PARA SISTEMAS LINEARES

pmgasjob.m - programa em Matlab.

$$\begin{cases} x_1 + 3x_2 + x_3 = -2 \\ 5x_1 + 2x_2 + 2x_3 = 36x_2 + 8x_3 = -6 \\ 6x_2 + 8x_3 = -6 \end{cases}$$

Pode-se observar que nesta execução, foi permutada a segunda linha com a primeira e obtivemos o resultado em apenas 8 iterações.

```
pmgasjob03.m
pmgasjob(a,b,e)
PMGASJOB-Pgma do Metodo Gauss-Jacobi - para
sist.lineares com 3 incognitas
Matriz do Sistema Linear informado:
```

```
a = 5   2   2   3
     1   3   1  -2
     0   6   8  -6
```

Matriz das Aproximações Iniciais informada :

```
b = 0
     0
     0
```

Precisao Informada:
e=0.0500000000000000
Achamos o valor aproximado com 8 iteracoes
Valores encontrados para x, a partir do valor inicial:

```
wxk =
Columns      1      through      7
Column 8
    0.6000    1.1667    0.9467    1.1039    0.9629    1.0452
    0.9754    1.0210
   -0.6667   -0.6167   -0.9722   -0.8864   -1.0277   -0.9592   -
   1.0220   -0.9816
   -0.7500   -0.2500   -0.2875   -0.0208   -0.0852    0.0208   -
    0.0306    0.0165
```


11.5 MÉTODO DE GAUSS-SEIDEL PARA SISTEMAS LINEARES

pmgassei.m – programa em Matlab.

$$\begin{cases} 2x_1 + x_2 + 3x_3 = 9 \\ -x_2 + x_3 = 1 & x_1 + 3x_3 = 3 \\ x_1 + 3x_3 = 3 \end{cases}$$

Pode-se observar que a matriz informada ao programa para ser executado está com aparência diferente da forma em que se apresenta como exercício proposto.

Não está detalhado aqui, mas submeteu-se o sistema de equações ao Critério das Linhas, o qual não foi satisfeito, permutou-se a primeira linha pela terceira, mas mesmo assim continuou não satisfazendo o Critério das Linhas. Em seguida permutou-se a primeira coluna pela terceira, quando somente então tivemos o Critério das Linhas satisfeito e o submetemos ao Método de Gauss-Seidel.

pmgassei03.m

pmgassei(a,b,e)

PMGASSEI-Pgma do Metodo Gauss-Seidel - para sist.lineares

com 3 incognitas

Matriz do Sistema Linear informado:

$$a = \begin{bmatrix} 3 & 0 & 1 & 3 \\ 1 & -1 & 0 & 1 \\ 3 & 1 & 2 & 9 \end{bmatrix}$$

Matriz das Aproximações Iniciais informada :

$$b = 0$$

0

0

Precisao Informada:

$e=0.050000000000000$

Achamos o valor aproximado com 6 iteracoes

Valores encontrados para x, a partir do valor inicial:

wxk =

1.0000 0 -0.6667 -1.1111 -1.4074 -1.6049

0 -1.0000 -1.6667 -2.1111 -2.4074 -2.6049

3.0000 5.0000 6.3333 7.2222 7.8148 8.2099

11.6 MÉTODO DE BARNES PARA PROGRAMAÇÃO LINEAR

pmbarnes.m – programa em Matlab.

$$\text{Maximizar } z = 2x_1 + x_2 + 4x_3$$

função objetivo

$$\begin{aligned} \text{Sujeita a } & x_1 + x_2 + x_3 \leq 7 \\ & x_1 + 2x_2 + 3x_3 \leq 12 \end{aligned}$$

inequações de restrição

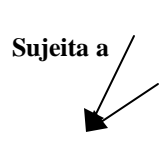
Este problema de programação linear pode ser facilmente transformado na forma padrão:

- adicionando-se novas variáveis positivas – chamadas variáveis fracas – ao lado esquerdo das inequações de restrição;
- e trocando-se o sinal dos coeficientes da função objetivo;

de tal modo que o problema fica convertido m um problema de minimização sujeito à equações de restrição tal como:

$$\text{Mnimizar } -z = -2x_1 - x_2 - 4x_3$$

função objetivo



$$x_1 + x_2 + x_3 + x_4 = 7$$

$$x_1 + 2x_2 + 3x_3 + x_5 = 12$$

equações de restrição

Variáveis fracas

As variáveis x_4 e x_5 são chamadas “variáveis fracas”, e elas representam a diferença entre os recursos disponíveis e os recursos utilizados. Note que Se as inequações de restrição forem ≥ 0 , então subtraímos as variáveis fracas, para produzir igualdade.

Para executar o método, criamos o seguinte Arquivo de Lote:

```

pmbarnes01.m
c=[-2 -1 -4 0 0 ];
a=[1 1 1 0;1 2 3 0 1];
b=[7;12];
[xsol,ind]=pmbarnes(a,b,c,0.00005);
i=1;

fprintf("\nSolution is: ");
for j=ind
    fprintf('\nx(%1.0f)=%8.4f\''j,xsol(i));
    i=i+1;
end
fprintf("\n0other variables are zero\n")

```

E obtive-se o seguinte resultado:

```

pmbarnes01s.m
objective =
-19.0000

Solution is:
x(1)= 4.5000
x(3)= 2.5000
0ther          variables          are          zero

```

12. ALGUMAS CONCLUSÕES

Apresentou-se uma revisão dos métodos mais utilizados nas aplicações de álgebra linear e não-linear computacional, em caráter introdutório. Vários exemplos foram apresentados, para o entendimento dos diversos métodos. Utilizou-se da linguagem MATLAB®, devido a facilidade e aceitação, nos dias atuais, nos diversos setores. Este texto serve como uma introdução aos alunos de pós-graduação . não exploro-se aq programação paralela que seria o próximo passo, destas notas.

13. REFERÊNCIAS BIBLIOGRÁFICAS DE SUPORTE

- GEROTTO O H, *Estudo de Técnicas de Otimização e Algumas Aplicações em Ciência da Engenharia*, Relatório de Estágio de Especialização em Matemática Aplicada, Orientador Prof. Dr. JM Balthazar UNESP-Campus de Rio Claro, SP, 2002
- CHAPRA, CS, CANALE, RP. *Numerical Methods*, McGraw-Hill, Inc., 1985
- LINDFIELD G.; PENNY J., *Numerical Methods using Matlab*, Ellis Horwood, 1995.
- BURDEN R.L.; FAIRES J.D., *Numerical Analysis*, 5ª Edição, PWS Publishing Co., 1993.
- NAYFEH A. H.; BALACHANDRAN B., *Applied Nonlinear Dynamics*, John Wiley & Sons, 1994.

- MATSUMOTO, E.Y., *Matlab 6 – Fundamentos de Programação*, Editora Érica Ltd., 2001.
- SCHEID, F., *Análise Numérica*, McGraw-Hill, 1991.
- ATKINSON, K. E., *An Introduction to Numerical Analysis*, John Wiley & Sons, 1988.
- HOUPIS, D´Azzo, *Análise e Projeto de Sistemas de Controle Lineares*, Editora Guanabara S/ A, 1988.
- HANSELMAN, D.; LITTLEFIELD B.; *Matlab 5 Guia do Usuário*, Makron Books, 1999.
- PRESS ET AL., *Numerical Recipes*, Cambridge, University Press, 1986.
- YOUNG, DM, GREGORY, RT, *A Survey of Numerical Mathematics*, Volume 1 , 2, addison wesley, 1972.
- STOER, J, Bulirsh, *Introduction to Numerical Analysis*, Springer Verlag, 1980
- Atkinson, KE, *An Introduction to Numerical analysis*, John Wiley, 1978
- Toledo, EM, Silva RD, *Introdução à Computação Paralela*, Laboratório de Computação Científica, 1997.

14. AGRADECIMENTOS

Os autores agradecem a FAPESP(Proc.2002/04130-1), CNPq e DEMAC-IGCE, pelo possibilidade de realização desta pesquisa.