Revista Eletrônica Paulista de Matemática

ISSN 2316-9664
Volume 16, dez. 2019

Solução numérica de sistemas de equações não lineares: combinando uma técnica de busca não exaustiva com um método do tipo Quasi-Newton

Numerical solution of non-linear equation systems: combining a non-exhaustive search technique with a Quasi-Newton type method

Francisco José Alves de Aquino
Instituto Federal do Ceará, Campus Fortaleza
fcoalves_aq@ifce.edu.br

Resumo

Neste artigo apresentamos uma proposta de fácil implementação computacional para o valor inicial (raiz) da solução de um sistema de equações não lineares. Para refinar a solução do sistema usamos um método Quasi-Newton bastante intuitivo: calculamos as derivadas necessárias por uma aproximação numérica que resulta no final em uma taxa de convergência quadrática. Apresentamos também alguns resultados numéricos que ilustram o método proposto e a sua eficácia.
Palavras-chave: Métodos numéricos. Sistema não lineares. Método Quasi-Newton.

Abstract

In this paper we present a proposal of easy computational implementation for the initial value (root) of the solution of a system of nonlinear equations. To refine the solution of the system we use a Quasi-Newton method quite intuitive: we calculate the necessary derivatives by a numerical approximation that results in the end in a quadratic convergence rate. We also present some numerical results that illustrate the proposed method and its effectiveness.
Keywords: Numerical methods. Nonlinear systems. Quasi-Newton method.

Artigo recebido em jul. 2019 e aceito em dez. 2019.

1 Introdução

Diversos problemas oriundos da matemática, física, química e engenharia podem ser equacionados como um conjunto de equações não lineares. Vejamos um exemplo simples originado de uma equação polinomial de grau três. As raízes de um polinômio de grau três com coeficientes constantes do tipo $f(x) = x^3 + Ax^2 + Bx + C = (x + x_1)(x + x_2)(x + x_3)$ podem ser encontradas pela solução do seguinte sistema de equações não lineares:

$$\mathbf{F} = \begin{cases} x_1 + x_2 + x_3 - A &= 0 \\ x_1x_2 + x_1x_3 + x_2x_3 - B &= 0 \\ x_1x_2x_3 - C &= 0 \end{cases}$$

Em geral, podemos encontrar a solução deste tipo de sistemas de equações usando algum método iterativo, tendo como ponto de partida uma aproximação inicial $\mathbf{x}^0 = (x_1^0, x_2^0, x_3^0)$ que seja próximo o suficiente à raiz exata do sistema. Um dos mais conhecidos e mais usados métodos ou que serve de fundamento para outros métodos para resolver esse problema é método de Newton para várias variáveis. Dependendo do sistema não linear, também podemos tentar usar o método iterativo linear (BURDEN; FAIRES, 2008). Lembramos que o método de Newton para solução de equação de uma única variável $f(x) = 0$ pode ser expresso por (CAMPOS FILHO, 2007; CHAPRA; CANALE, 2008):

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \tag{1}$$

onde $f'(x_k)$ é da derivada (ou uma estimativa da derivada) da função $f(x)$ em $x = x_k$. Em termos mais formais, podemos escrever este tipo de problema para duas ou mais variáveis como

$$\mathbf{F}(\mathbf{x}) = f(x_1, x_2, \ldots, x_n) = 0, \tag{2}$$

onde $\mathbf{F} : \Re^n \to \Re^n$ é um vetor de funções continuamente diferenciáveis, isto é,

$$\mathbf{F}(\mathbf{X}) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ \vdots & \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{bmatrix}. \tag{3}$$

Neste artigo, vamos considerar que cada função $f_i : \Re^n \to \Re$, $i = 1, 2, \ldots, n$, seja suave. Outras condições necessárias para a solução numérica do sistema são apresentadas na próxima seção. Destacamos que o sistema de equações não lineares representado em 3 pode não ter uma solução, ter apenas uma solução ou apresentar muitas soluções (NOCEDAL; WRIGHT, 2006).

Temos dois objetivos principais neste artigo:

- apresentar um método simples de busca de uma região que possivelmente apresenta uma raiz do sistema;
- apresentar um método do tipo Quasi-Newton de fácil implementação computacional que chegue à raiz em poucas iterações.

Organizamos o restante deste artigo como segue. Na seção 2 introduzimos os conceitos básicos sobre a solução numérica de sistemas não lineares. Na seção 3 propomos um algoritmo prático para a solução de sistemas não lineares de fácil implementação computacional. Na seção 4 apresentamos alguns resultados numéricos. Na sequência, temos as conclusões e um apêndice com o código implementado de um dos exemplos numéricos.

2 Sistemas não lineares: conceitos básicos

Um sistema de equações não lineares é, em geral, bem mais complicado de se resolver que um sistema linear, além de raramente existir algum método de solução analítica. Em alguma situações, nos deparamos com um sistema de equações não lineares. Por exemplo, quais são os pontos de interseção do círculo $x^2+y^2=2$ e da parábola $y-x^2=0$? O cálculo desses pontos envolve a solução do sistema de equações não lineares:

$$\begin{cases} x^2+y^2-2 &= 0 \\ -x^2+y &= 0 \end{cases}$$

No sistema descrito anteriormente é fácil encontrar a solução analítica, pois envolve apenas a resolução de uma equação de segundo grau. Já para o sistema não linear

$$\begin{cases} x^2/4+y^2/8-2 &= 0 \\ \text{sen}(x)+e^{-y^2/3}-2/3 &= 0 \end{cases}$$

não é fácil encontrar suas raízes. Naturalmente, neste artigo assumiremos que:

- o mapeamento $\mathbf{F}(\mathbf{X})$ é continuamente diferenciável em um conjunto convexo aberto D.
- Existe um $\mathbf{X}^*$ em D tal que $\mathbf{F}(\mathbf{X}^*)=0$ e a matriz jacobiana $\mathbf{J}(\mathbf{X}^*)$ é não singular.

2.1 Método de Newton

Como explicamos inicialmente, podemos adaptar o método iterativo de Newton para cálculo das raízes de um sistema não linear. Neste caso, podemos reescrever a equação 1 como

$$\mathbf{X}_{k+1}=\mathbf{X}_k-\mathbf{J}_k^{-1}\mathbf{F}_k \tag{4}$$

onde $\mathbf{X}_k$ é uma aproximação do vetor solução do sistema, $\mathbf{J}_k$ é a matriz jacobiana definida por

$$\mathbf{J}_k=\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{5}$$

e

$$\mathbf{F}_k=\begin{bmatrix} f_1(x_1^k,x_2^k,\ldots,x_n^k) \\ f_2(x_1^k,x_2^k,\ldots,x_n^k) \\ \vdots \\ f_n(x_1^k,x_2^k,\ldots,x_n^k) \end{bmatrix}. \tag{6}$$

Existem duas desvantagens principais no uso do Método de Newton:

- é necessário o conhecimento da jacobiana $\mathbf{J}$ e que deve ser calculada a cada iteração, além de não poder ser singular;

- a aproximação inicial $\mathbf{X}_0$ precisa ser suficientemente próximo de uma das raízes do sistema para que ocorra a convergência.

Para contornar a primeira desvantagem apontada acima, foram propostas várias soluções, entre elas o método de Broyden (BROYDEN, 1965). Entretanto, neste artigo optamos por simplesmente calcular a matriz jacobiana de forma aproximada usando o método da secante modificada (diferença finita centrada):

$$\frac{df(x_k)}{dx} \cong \frac{f(x_k+h)-f(x_k-h)}{2h}, \tag{7}$$

sendo h um valor pequeno (por exemplo: $h = 10^{-6}$). Dessa forma, usamos um método Quasi-Newton neste artigo. Para outras considerações mais teóricas sobre os métodos Quasi-Newton recomendamos a leitura do artigo clássico de Dennis Jr e Moré (1977).

2.2 Raiz inicial

Para o método de Newton ou, de forma geral, qualquer método Quasi-Newton, funcionar adequadamente, o valor inicial para a raiz do sistema não pode estar muito distante de uma das raízes. Entretanto, nem sempre existe uma forma geral de estimar adequadamente essa aproximação inicial. Aqui entra a nossa proposta: uma busca não exaustiva por um ponto próximo de uma região de mínimo (possivelmente global) da função de mérito (NOCEDAL; WRIGHT, 2006):

$$g(x_1,x_2,\ldots,x_n) = f_1^2(x_1,x_2,\ldots,x_n) + f_2^2(x_1,x_2,\ldots,x_n) + \cdots + f_n^2(x_1,x_2,\ldots,x_n). \tag{8}$$

Percebemos facilmente que a raiz X^* do sistema não linear também faz com que a função de mérito $g(X^*)$ seja nula. Duas vantagens evidentes desta técnica são a simplicidade e a não necessidade de conhecermos o comportamento detalhado do sistema. Descrevemos essa busca com mais detalhes na próxima seção.

3 Algoritmo proposto

O primeiro passo é localizar um ponto $(x_1^0, x_2^0, \ldots, x_n^0)$ perto de uma região que (possivelmente) tenha uma raiz do sistema. Para isso, fazemos cada variável x_i variar dentro de um intervalo fechado $(x_{inicial}, x_{final})$ determinado com um passo Δ: $x_i = x_{inicial} + k\Delta$, com $k = 0,1,2,\ldots,K$; e $x_{inicial} + K\Delta = x_{final}$. O ponto escolhido é o menor valor encontrado de $g(\mathbf{X})$, na equação 8. Em torno desse ponto inicial, esse processo de busca é repetido, mas agora com um Δ menor: $\Delta \leftarrow \Delta/2,5$. O valor "2,5" foi escolhido após alguns testes numéricos. Dessa forma encontramos um ponto ainda mais próximo do mínimo. Esse processo pode ser encerrado se $g(\mathbf{X}) < \lambda_0$ ou continuado por mais uma interação ou até que seja atingido um número máximo de interações. O valor de λ_0 é um parâmetro arbitrário pequeno, por exemplo: $\lambda_0 = 0,1$. A Figura 1 mostra um exemplo de pontos encontrados para um sistema com duas incógnitas.

Com um ponto de partida já próximo a uma das raízes do sistema não linear, podemos agora calcular o valor numérico das derivadas parciais da Jacobiana de forma aproximada por:

$$\frac{\partial f_i(x_1,x_2,\ldots,x_n)}{\partial x_1} \cong \frac{f_i(x_1+h,x_2,\ldots,x_n) - f_i(x_1-h,x_2,\ldots,x_n)}{2h}. \tag{9}$$

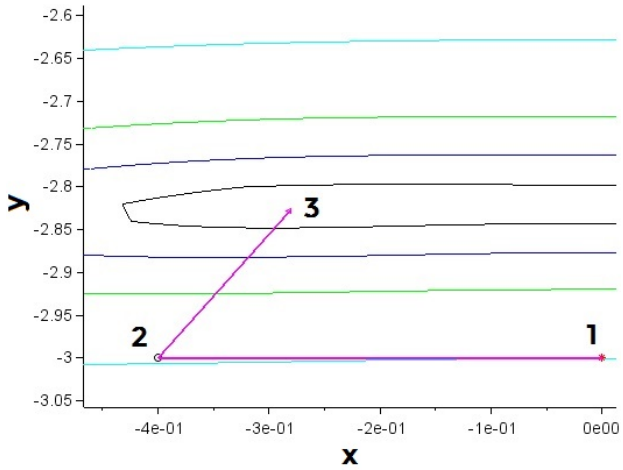

Figura 1: Exemplo de sequência de pontos encontrados em uma busca não exaustiva. O primeiro ponto encontrado foi o ponto "1". O ponto "3" já está próximo de um zero do sistema não linear.

Agora devemos calcular a inversa da Jacobiana (supondo que essa matriz não seja singular) e aplicar a forma recursiva indicada pela equação 4. Como critério de parada podemos adotar $g(\mathbf{X}) < \lambda_1$, com $\lambda_1 = 10^{-6}$, por exemplo. Também seria possível escolher $\|x_{k+1} - x_k\| < \varepsilon$ (com $\varepsilon > 0$) como critério de parada com um menor custo computacional. Naturalmente, se o número de iterações exceder um valor máximo (digamos 5 ou 6), o laço deve ser encerrado mesmo sem encontrar a raiz do sistema. Isso pode significar que o sistema não tem solução ou essa raiz não existe no local que estava sendo buscada. Resumimos os principais passos desse processo no fluxograma indicado pela Figura 2.

4 Resultados numéricos

A seguir, apresentamos o ambiente de simulação e alguns exemplos numéricos para mostrar a eficácia do algoritmo proposto.

4.1 Ambiente de cálculo

Implementamos todos os cálculos e algoritmos usando o *software* Scilab (SCILAB, 2019) em um computador portátil (*laptop*) ACER com processador Intel(R) Core(TM) i5, CPU 1,60 GHz, com 6 GB de memória RAM, sistema operacional: *Windows* 8-6.2. Escolhemos o Scilab (versão 5.5.0.1397209685) como ambiente de cálculo por ser um *software* livre, semelhante ao Matlab(R), mas bem mais leve. O *machine epsilon* é cerca de 2,2204e-16.

Podemos ainda destacar que a inversão de matrizes quadradas (essencial nos métodos de Newton e Quasi-Newton) é feita usando apenas o comando "inv()" no Scilab. Uma aviso é impresso na tela se a matriz for mal escalada ou quase singular. A função "inv()" para matrizes de números reais é baseada nas rotinas de Lapack DGETRF, DGETRI, sendo bastante eficiente computacionalmente (SCILAB, 2019). A inversão de uma matriz 100×100 leva um pouco menos de 2 ms (milisegundos), já uma matriz 1000×1000 precisa de aproximadamente 130 ms.

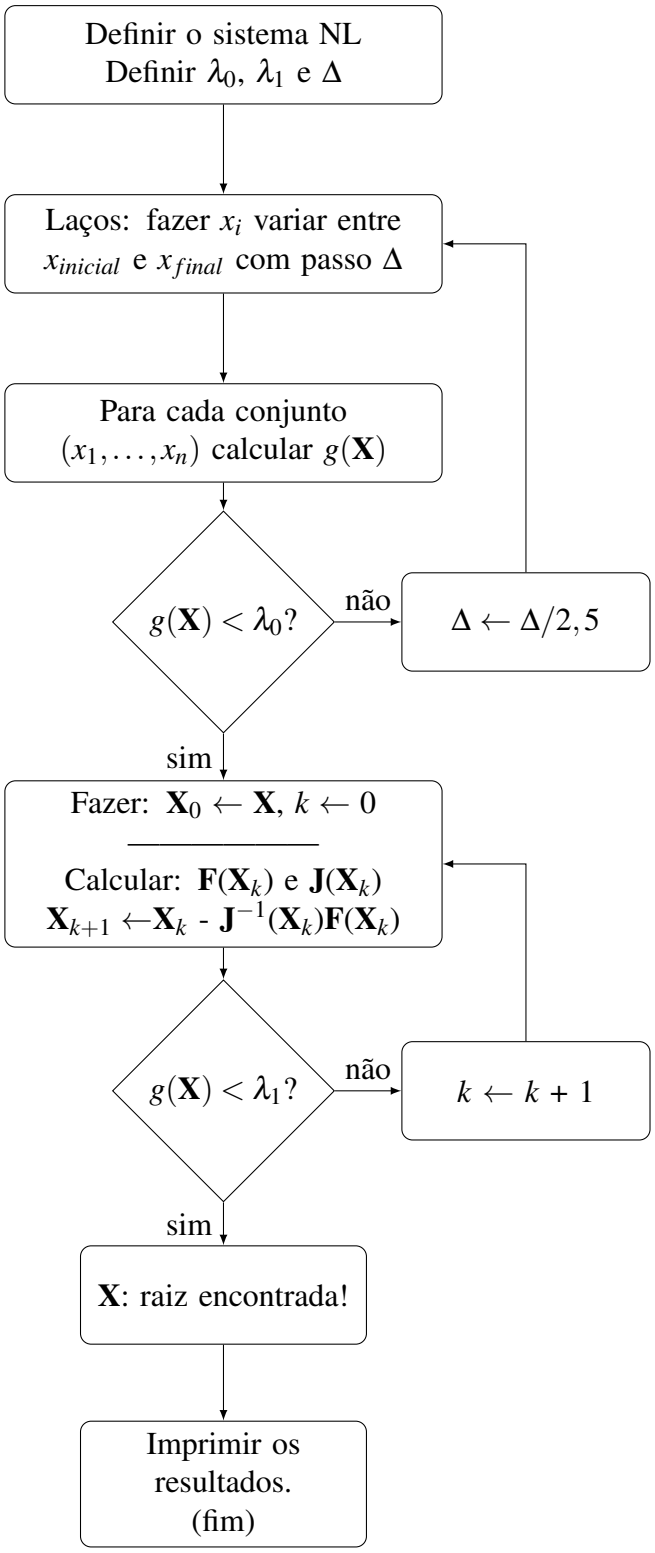

Figura 2: Fluxograma proposto - principais passos.

Logo, para os propósitos deste trabalho, o processo de inversão de matriz não é uma limitação e apresenta um baixo custo computacional.

4.2 Exemplos numéricos

Para ilustrar o método proposto, como primeiro exemplo consideramos o sistema de equações de duas variáveis

$$\mathbf{F}_1 = \begin{cases} x^2/4 + y^2/8 - 3 & = 0 \\ \text{sen}(x) + e^{-y^2/2} - 2/3 & = 0 \end{cases} \tag{10}$$

Considerando os parâmetros: $\lambda_0 = 0,05$; $\lambda_1 = 10^{-6}$; $\Delta = 1$; com $x_{inicial} = -3$ e $x_{final} = 3$. Os valores calculados são apresentados na Tabela 1 e a trajetória da solução está indicada na Figura 3. Podemos observar a rápida convergência do método Quasi-Newton, pois o valor inicial para uma das raízes desse sistema estava já bem próxima de seu valor correto. Vale observar que, para este exemplo, o resultado obtido é exatamente o mesmo que o alcançado pelo método de Newton com o cálculo das derivadas parciais analíticas de $\mathbf{J}_k$.

Tabela 1: Solução do sistema $\mathbf{F}_1$ (10)

k	x_k	y_k	$g(x_k, y_k)$
-	3,0	-2,0	0,2147649
-	2,6	-3,2	0,0219799
1	2,4233180	-3,5246083	0,0004832
2	2,4146995	-3,5126548	1,850D-09

Como segundo exemplo vamos considerar o sistema de três equações

$$\mathbf{F}_2 = \begin{cases} 3x - \cos(xy) - 1,00 & = 0 \\ x^2 - 81(y+0,1)^2 + \text{sen}(z) + 8,43 & = 0 \\ e^{-xy} + 20z + 11,30 & = 0 \end{cases} \tag{11}$$

Para este segundo exemplo, consideramos os parâmetros: $\lambda_0 = 5$; $\lambda_1 = 10^{-6}$; $\Delta = 1$; com $x_{inicial} = -3$ e $x_{final} = 3$. Os valores calculados são apresentados na Tabela 2. Novamente, podemos observar a rápida convergência do método Quasi-Newton, pois o valor inicial para uma das raízes desse sistema já estava bem próximo de seu valor correto. Entretanto, neste caso, usamos uma tolerância maior para λ_0 e foram necessárias quatro iterações na busca de um ponto próximo à raiz do sistema.

Como terceiro e último exemplo, consideremos o seguinte sistema:

$$\mathbf{F}_3 = \begin{cases} x^2 + y^2 + \sqrt{z} - 4 & = 0 \\ (x+y)^2 + (y+z)^2/25 - 5 & = 0 \\ (x-y)^2 + (y-z)^2 - 9 & = 0 \end{cases} \tag{12}$$

Para este terceiro exemplo, consideramos os parâmetros: $\lambda_0 = 0,33$; $\lambda_1 = 10^{-8}$; $\Delta = 1$; com $x_{inicial} = -3$ e $x_{final} = 3$. Os valores calculados são apresentados na Tabela 3. Os valores iniciais da Tabela 3 são iguais, pois mesmo com Δ menor não foi encontrado um ponto mais próximo do zero da função. Mais uma vez, observamos a rápida convergência do método Quasi-Newton, pois a aproximação inicial para uma das raízes desse sistema foi adequado. Neste exemplo, foram três iterações antes de entrar no método Quasi-Newton e outras três para chegar a uma solução.

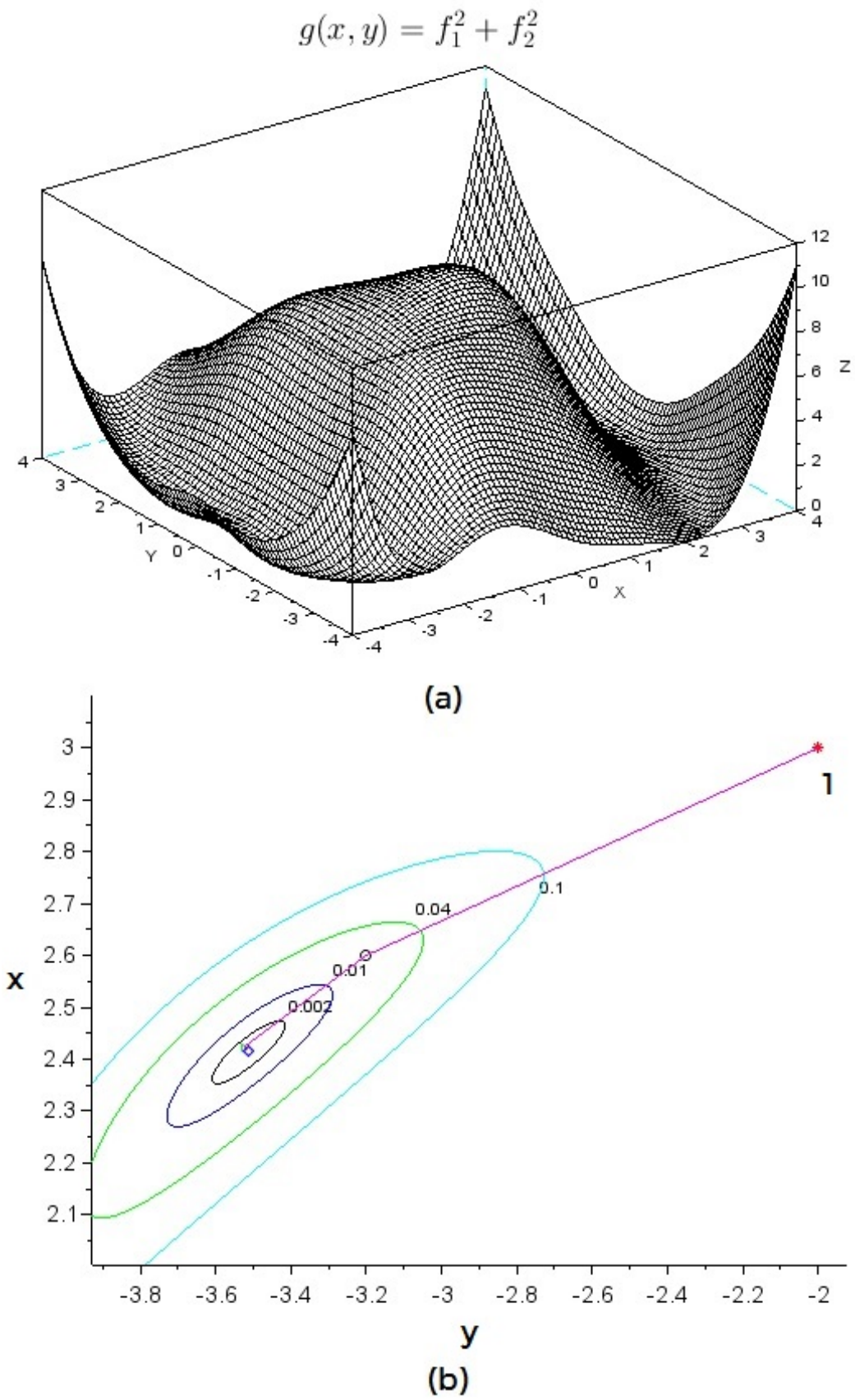

Figura 3: Convergência da solução do sistema $\mathbf{F}_1$: (a) gráfico 3D da função $g(x,y)$, (b) aproximação de uma raiz do sistema.

Tabela 2: Solução do sistema $\mathbf{F}_2$ (11)

k	x_k	y_k	z_k	$g(x_k, y_k, z_k)$
-	0.00	0.00	-1.00	109.23846
-	1.00	-0.50	-0.50	26.030271
-	1.00	0.25	-0.50	6.3291935
-	0.875	0.25	-0.625	2.2994466
1	0.6643927	0.2205994	-0.6083250	0.0006775
2	0.6631234	0.2200763	-0.6082106	4.228D-10

Tabela 3: Solução do sistema $\mathbf{F}_3$ (12)

k	x_k	y_k	z_k	$g(x_k, y_k, z_k)$
-	-1.00	-1.00	2.00	0.343146
-	-1.00	-1.00	2.00	0.343146
-	-0.75	-1.50	1.50	0.322936
1	-0.629992	-1.574791	1.300110	1.329429e-02
2	-0.624694	-1.574370	1.279402	3.283765e-07
3	-0.624583	-1.574455	1.279134	2.239418e-14

5 Conclusão

Neste artigo propomos um algoritmo de fácil implementação em *software* para inicialização de um método Quasi-Newton. A busca por um bom valor inicial para encontrar uma solução para um sistema de equações não lineares é essencial para a convergência dos métodos do tipo Newton. O método proposto é de fácil implementação, não exaustivo e consegue preservar a taxa de convergência quadrática do método de Newton, como mostram os resultados numéricos apresentados. Naturalmente, para o funcionamento do método proposto, supomos que existe uma solução dentro da região inicialmente pesquisada. A definição da área de busca inicial e dos outros parâmetros do método devem ser ajustados para cada problema, mas esse ponto ainda precisa ser melhor estudado, ficando como trabalho futuro.

6 Referências bibliográficas

BROYDEN, C. G. A class of methods for solving nonlinear simultaneous equations. **Mathematics of Computation**, v. 19, n. 92, p. 577-593, 1965.

BURDEN, R. L.; FAIRES, J. D. **Análise numérica**. São Paulo: Cengage Learning, 2008.

CAMPOS FILHO, F. F. **Algoritmos numéricos**. 2. ed. Rio de Janeiro: LTC, 2007.

CHAPRA, S. C.; CANALE, R. P. **Métodos numéricos para engenharia**. 5. ed. São Paulo: McGraw-Hill, 2008.

DENNIS JR, J. E.; MORÉ, J. J. Quasi-Newton methods, motivation and theory. **SIAM Review**, v. 19, n. 1, p. 46-89, 1977. DOI: https://doi.org/10.1137/1019005. Disponível em:

https://epubs.siam.org/doi/pdf/10.1137/1019005. Acesso em: 28 set. 2019.

NOCEDAL, J.; WRIGHT, S. J. **Numerical Optimization**. 2nd New York: Springer, 2006.

SCILAB. Open source software for numerical computation. 2019. Disponível em: http://www.scilab.org. Acesso em: 10 jul. 2019.

7 Apêndice: código Scilab para o primeiro exemplo

```
//// Código Scilab - exemplo 01.
clear; clc;      /// limpando a memória, a tela
xdel(winsid());  /// e fechando as janelas

disp(' Começando os cálculos ... ');
h = 1e-6; h2 = 2*h;  /// passo para derivada numérica

function f=fz(x, y)   /// função f1
   f = x.*x/4 +y.*y/8 - 3;
endfunction
function df=dfzx(x, y)   /// derivada parcial de f1 em relação a x
   df = (fz(x+h,y)-fz(x-h,y))/h2;
endfunction
function df=dfzy(x, y) /// derivada parcial de f1 em relação a y
   df = (fz(x,y+h)-fz(x,y-h))/h2;
endfunction

function g=fw(x, y) /// função f2
   g = sin(x) +exp(-y.*y/2)-2/3;
endfunction
function df=dfwx(x, y)  /// derivada parcial de f2 em relação a x
   df = (fw(x+h,y)-fw(x-h,y))/h2;
endfunction
function df=dfwy(x, y)  /// derivada parcial de f2 em relação a y
   df = (fw(x,y+h)-fw(x,y-h))/h2;
endfunction

///// Para fazer o gráfico 3D:
pp = -4:0.01:4;
[X,Y] = meshgrid(pp,pp);
f1 = fz(X,Y);
f2 = fw(X,Y);
Z= f1.*f1 + f2.*f2;
xtitle('$"huge g(x,y)=f˙1^2+f˙2^2$'); ps = 10;
mesh(X(1:ps:$,1:ps:$),Y(1:ps:$,1:ps:$),Z(1:ps:$,1:ps:$));

//// Curvas de nível:
```

```
zmax = 0.98*max(max(Z));
zn = [0.002, 0.01, 0.04, 0.1, zmax/4, zmax/2, 0.85*zmax, zmax];
x = pp;
y = x;
figure; contour(x,y,Z,zn);
xlabel('x'); ylabel('y');

//// Refazendo as curvas de nível:
figure; contour(x,y,Z,zn);
xlabel('x'); ylabel('y');

////////// Busca por um mínimo local por inspeção:
disp(' *** Busca não exaustiva:');
Lg = 0.05;  // valor de teste
divv = 1;   // Delta = 1
gmin = 10;  /// para iniciar o laço
pxx = [];   /// guardar os pontos x
pyy = [];   /// guardar os pontos y
x0 = 0;  y0 = 0;  kk = 1;
cor = ['r','k','m','g','b','r','k','m'];
simb = ['*','o','¡','¿','d','p','x','s'];
//// Laço de busca:
while gmin ¿ Lg do
   x = [-3,-2,-1,0,1,2,3]/divv;
   y = x;
   divv = divv * 2.5;
   for k=1:max(size(x))
        xk = x0 + x(k);
         for p=1:max(size(y))
            yp = y0 + y(p);
            fa = fz(xk,yp);
            fb = fw(xk,yp);
            g2 = fa*fa + fb*fb;
            if gmin ¿ g2 then
               gmin = g2;
               xmin = xk;
               ymin = yp;
             end;
        end
    end
    pxx = [pxx, xmin];
    pyy = [pyy, ymin];
    /// mostrando os resultados:
    mprintf(' %i & %f & %f & %f "n',kk, xmin, ymin, gmin);
    plot(ymin,xmin,cor(kk)+simb(kk));
    x0 = xmin;   /// atualizando novo x-minimo
```

```
    y0 = ymin;   /// atualizando novo y-minimo
    kk = kk + 1; /// próxima iteração
    if kk ¿ 5 then gmin = 0;
    end
end;

disp('*** Quasi-Newton:');
///// Método Quasi-Newton:
L2 = 1e-6;
kk = 1;
gmin = 1;
while gmin¿L2
   X = [x0; y0];
   J = [dfzx(x0,y0),  dfzy(x0,y0);  dfwx(x0,y0),  dfwy(x0,y0)];
   F = [fz(x0,y0);  fw(x0,y0)];
   X = X - inv(J)*F;
   x0 = X(1);
   y0 = X(2);
   pxx = [pxx, x0];
   pyy = [pyy, y0];
   fa = fz(x0,y0);
   fb = fw(x0,y0);
   gmin = fa*fa + fb*fb;
   mprintf(' %i & %f & %f & %e "n',kk, x0, y0, gmin);
   plot(y0,x0,cor(kk+1)+simb(kk+1));
   kk = kk + 1;
   if kk ¿ 5 then gmin = 0; /// saída do laço
   end
end;

//// Pontos nas curva de nível:
plot(pyy,pxx,'m');
title(simb,'fontsize',3);
```

Resultando na saída:

```
 Começando os cálculos ...

 *** Busca não exaustiva:
1 & 3.000000 & -2.000000 & 0.214765
2 & 2.600000 & -3.200000 & 0.021980

*** Quasi-Newton:
1 & 2.423318 & -3.524608 & 4.831834e-04
2 & 2.414700 & -3.512655 & 1.849926e-09
```