Análise Numérica - T2 - Método de Newton para sistemas não lineares

Gustavo Diel[1], **Nicolas Peter Lane**[2]

[1]Departamento de Ciência da Computação – Universidade do Estado de Santa Catarina
Centro de Ciências Tecnológicas – Caixa Postal 15.064 – Joinville – SC – Brasil

{[1]gustavodiel, [2]nicolaspeterlane}@hotmail.com

Resumo. *O presente relatório tem como objetivo mostrar os resultados obtidos utilizando o método de Newton para sistemas não lineares, estudado em aula, além de demonstrar uma análise teórica sobre o teorema.*

1. Descrição do Método de Newton

Sistemas não lineares são comuns de serem encontrados tendo recorrentes aplicações em engenharias e ciências afins. Por sua vez, a necessidade de encontrar métodos capazes de resolvê-los por aproximação motivou a criação de diversos métodos (*e.g.,* ponto fixo, Newton, *etc.*) para as suas resoluções. Dentre esses métodos, destaca-se o método de Newton, capaz de obter uma convergência com complexidade de tempo de ($O(n^2)$), sendo a primeira forma adaptada de algoritmo para o fim de resolver sistemas não lineares por aproximação em computadores.

Dado uma função não linear F, composta de n funções, o objetivo é encontrar uma solução para o mesmo:

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

Sendo que:

$$F = (f_1, f_2, ..., f_n)$$

Pode-se criar um vetor tal que:

$$x = (x_0, x_1, ..., x_n)$$

Ou seja, temos como objetivo:

$$F(x) = 0$$

Há de haver a satisfação de alguns quesitos para o método de Newton, eles o são:

- Dado um ponto $x^{(k)}$, Aplica-se ele sobre a matriz $J(x^{(k)})$, conhecida como a matriz jacobiana. A matriz jacobiana é composta das derivadas parciais do sistema F, vide a equação J.

$$J(f_0, f_1, ..., f_n) = \begin{bmatrix} f_0x_0 & f_0x_1 & ... & f_0x_n \\ f_1x_0 & f_1x_1 & ... & f_1x_n \\ ... & ... & ... & ... \\ f_nx_0 & f_nx_1 & ... & f_nx_n \end{bmatrix}$$

  Sendo que quando aplicado o ponto $x^{(k)}$ na equação J há a necessidade de não resultar em nulo. Dado importante já que quando o determinante da matriz jacobiana no ponto $x^{(k)}$ é nulo, tal situação caracteriza que o sistema com a estimativa estipulada é indeterminado ou impossível.
- Após a descoberta da matriz jacobiana, é necessário resolver o sistema $J(x^{(k)}) = -F(x^{(k)})$.

O processo descrito deve ser repetido por cada iteração, o que torna-o moroso de realizar-se sem o auxílio de um computador e de complexa manipulação, já que a medida que a complexidade do sistema aumenta, proporcionalmente ao dobro aumenta a complexidade da matriz jacobiana a ser resolvida por iteração [Langtangen ,Burden et al. 2016].

2. Problema proposto

O presente trabalho aborda a resolução do sistema linear L:

$$L = \begin{cases} e^{-x_1} + e^{x_2} - x_1^2 - 2x_2^3 & (1) \\ cos(x_1 + \mathbf{p}x_2) + \mathbf{p}x_1 - x_2 - 1 & (2) \end{cases}$$

onde o coeficiente **p** representa os números referentes ao número de chamada dos integrantes do grupo. Neste caso, os números 11 e 27. O objetivo do trabalho é implementar o método de Newton para sistemas não lineares para dar um exemplo de aproximação $X^{(0)} = (x_1^{(0)}, x_2^{(0)})$ que produza sequências $\left\{X^{(k)}\right\}_{k=0}^{\infty}$ para os casos:

- Uma sequência em que X^5 aproxima uma das soluções com $||X^5 - X^4||_\infty < 10^{-5}$, de modo que leve exatamente 5 iterações para chegar na solução desejada
- Uma sequência em que X^5 aproxima uma outra solução $||X^5 - X^4||_\infty < 10^{-5}$
- Uma sequência que por algum motivo não se aproxime de nenhuma solução com erro menor que 10^{-5}, mesmo após 1000 iterações

3. Modelo Implementado

A implementação foi realizada usando a linguagem de *script Python* [Van Rossum and Drake 2011]. A implementação utiliza o pacote *Decimal*, que fornece uma biblioteca para cálculos de alta precisão, configurado para garantir precisão de 28 casas decimais.

As seguintes funções foram implementadas:

- **f_gu:** retorna o valor da função (1) em um determinado x.
- **fx_gu:** retorna o valor da derivada de (1) em relação a x_1 em um determinado x.

- **fy_gu:** retorna o valor da derivada de (1) em relação a x_2 em um determinado x.
- **g_gu:** retorna o valor da função (2), sendo $p = 11$ em um determinado x.
- **gx_gu:** retorna o valor da derivada de (2) em relação a x_1, com $p = 11$ em um determinado x.
- **gy_gu:** retorna o valor da derivada de (2) em relação a x_2, com $p = 11$ em um determinado x.
- **f_ni:** retorna o valor da função (1) em um determinado x.
- **fx_ni:** retorna o valor da derivada de (1) em relação a x_1 em um determinado x.
- **fy_ni:** retorna o valor da derivada de (1) em relação a x_2 em um determinado x.
- **g_ni:** retorna o valor da função (2), sendo $p = 27$ em um determinado x.
- **gx_ni:** retorna o valor da derivada de (2) em relação a x_1, com $p = 27$ em um determinado x.
- **gy_ni:** retorna o valor da derivada de (2) em relação a x_2, com $p = 27$ em um determinado x.
- **newton_naoLinear**: calcula as estimativas das funções aplicando o método de Newton.

4. Experimentos, Resultados e Análises

Sistemas analisados:

$$L_{11} = \begin{cases} e^{-x_1} + e^{x_2} - x_1^2 - 2x_2^3 \\ cos(x_1 + 11x_2) + 11x_1 - x_2 - 1 \end{cases}$$

$$L_{27} = \begin{cases} e^{-x_1} + e^{x_2} - x_1^2 - 2x_2^3 \\ cos(x_1 + 27x_2) + 27x_1 - x_2 - 1 \end{cases}$$

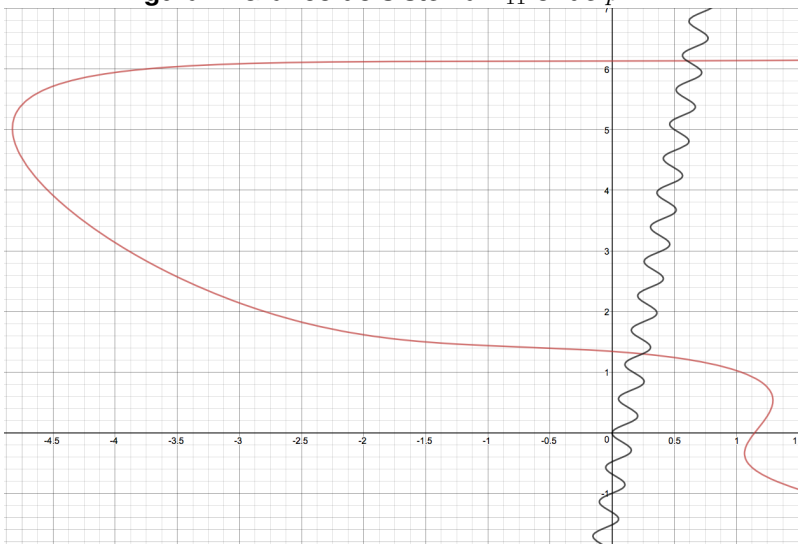
Para obter uma base visual de como o problema será resolvido, utilizou-se a ferramenta Desmos para produzir os gráficos referentes ao sistema. Ambos os gráficos estão ilustrados respectivamente na Figura 1, para o sistema L_{11} e na Figura 2, para o sistema L_{27}.

Analisando o sistema L_{11}, foram encontrados soluções para os três casos propostos. A Tabela 1 ilustra as iterações resultantes na ferramenta com $X^{(0)} = (-1.6, 1.29)$, resultando em $X^{(5)} = (0.2449, 1.2998)$, satisfazendo o caso 1. Em seguida, a Tabela 2 dispõem os resultados obtidos pela ferramenta partindo de $X^{(0)} = (0.5, 7.0)$. Esse ponto de partida ocasionou o resultado $X^{(5)} = (0.60200, 6.13457)$ em 5 iterações, conforme exigido pelo caso 2. Para a última análise do sistema L_{11}, chegou-se a Tabela 3, iniciando a função com ponto $X^{(0)} = (-50, -50)$, e mesmo após 1000 iterações, o erro absoluto não chegou a níveis aceitáveis, permanecendo sempre acima de 1.

Similarmente, as mesmas análises foram realizadas para o sistema L_{27}. Satisfazendo o caso 1, foi-se obtida a Tabela 4, $X^{(0)} = (2.19, 1.32)$, resultando em $X^{(5)} = (0.0921, 1.3287)$. A fim de obter uma outra solução e satisfazer o segundo caso, iniciou-se o problema com $X^{(0)} = (9.19, 6.2)$. O resultado obtido foi $X^{(5)} = (0.2944, 6.1325)$, com as iterações explicitadas na Tabela 5. Por fim, para atingir um ponto de partida em que aconteça uma divergência, iniciou-se o teste com $X^{(0)} = (-50, -50)$, e após 1000 iterações, o resultado continuou longe do desejado. As primeiras 5 iterações podem ser visualizadas na Tabela 6.

Em ambos os sistemas, é possível notar a rapidez do método de convergir para as soluções do sistema para quando os valores estão próximos da solução. Porém, quando o ponto inicial $X^{(0)}$ é encontrado a uma certa distância da solução, o método falha em convergir os valores para o resultado em tempo hábil.

Figura 1. Gráfico do sistema L_{11} onde $p = 11$



Fonte: Autores

Figura 2. Gráfico do sistema L_{27} onde $p = 27$

Fonte: Autores

Tabela 1. Resultados da execução do sistema L_{11} para o caso 1

Iteração (k)	X	Y	Erro
1	0.09849	1.09398	1.69849
2	0.00395	1.42020	0.32622
3	0.29358	1.30897	0.28962
4	0.24555	1.29987	0.04802
5	0.24484	1.29971	0.00070

Fonte: Os Autores.

Tabela 2. Resultados da execução do sistema L_{11} para o caso 2

Iteração (k)	X	Y	Erro
1	0.26645	6.48748	0.51260
2	0.58450	6.21280	0.31805
3	0.57375	6.13919	0.07370
4	0.60201	6.13458	0.02826
5	0.60200	6.13457	0.00001

Fonte: Os Autores.

Tabela 3. Resultados da execução do sistema L_{11} para o caso 3 começando com (-50, -50). Mesmo com 1000 iterações, o erro não atinge um valor aceitável (< 0.00001)

Iteração (k)	X	Y	Erro
1	-49.00000	-1005.18000	955.18000
2	-48.00000	-925.79500	79.38500
3	-47.00000	-1028.79000	102.99500
4	-46.00000	-952.67500	76.11500
5	-45.00000	-1005.81000	53.13500

Fonte: Os Autores.

Tabela 4. Resultados da execução do sistema L_{27} para o caso 1

Iteração (k)	X	Y	Erro
1	0.13176	1.87204	2.05824
2	-0.09641	1.49337	0.37867
3	0.04004	1.35440	0.13897
4	0.09084	1.32800	0.05080
5	0.09407	1.32665	0.00322

Fonte: Os Autores.

Tabela 5. Resultados da execução do sistema L_{27} para o caso 2

Iteração (k)	X	Y	Erro
1	-0.23760	5.7998	9.4276
2	-0.06869	6.2585	0.4587
3	0.30465	6.1441	0.37334
4	0.29573	6.13268	0.0115
5	0.29402	6.13257	0.00013

Fonte: Os Autores.

Tabela 6. Resultados da execução do sistema L_{27} para o caso 3 começando com (-50, -50). Mesmo com 1000 iterações, o erro não atinge um valor aceitável (< 0.00001)

Iteração (k)	X	Y	Erro
1	-49.00000	-99.692	49.692
2	-48.00000	-50.791	48.901
3	-47.00000	-125.84	75.049
4	-46.00000	-82.842	42.998
5	-45.00000	-36.648	46.194

Fonte: Os Autores.

5. Conclusão

Quando os pontos iniciais estão, de certa forma, próximos dos resultados esperados, o método se mostra bastante eficaz. Nos testes realizados, pontos com certa distância dos valores esperados ainda foram capazes de convergir para a solução. Porém, quando os pontos iniciais se distanciam da solução, o método demora para divergir (isso se a divergência ocorrer) Concluindo, o método se mostra eficiente para convergir valores próximos, então um meio de encontrar quais valores seriam próximos em algum sistema não linear é ideal.

Referências

Burden, R. L., Faires, J. D., and Burden, A. M. (2016). *Numerical Analysis*. Cengage Learning.

Langtangen, H. P. Solving multiple nonlinear algebraic equations.

Van Rossum, G. and Drake, F. L. (2011). *The python language reference manual*. Network Theory Ltd.