LABORATÓRIO NACIONAL DE COMPUTAÇÃO CIENTÍFICA
PROGRAMA DE PÓS-GRADUAÇÃO DO LABORATÓRIO NACIONAL DE COMPUTAÇÃO CIENTÍFICA
CURSO DE MESTRADO EM MODELAGEM COMPUTACIONAL

GA018 - MÉTODOS NUMÉRICOS

Trabalho do Módulo II

Discente: Wellington Silva
Docente: Marcio Borges

Petrópolis - RJ
Novembro - 2020

Sumário

1	**Introdução**	**2**
2	**Método de Newton na Otimização**	**6**
2.1	Deduzir o método de Newton para o caso bidimensional	6
2.2	Código computacional	7
2.3	Validação	10
2.4	Conclusão	10
3	**Sistema Mecânico**	**12**
3.1	Newton	12
3.1.1	Resultados do Método de Newton	14
3.2	Quasi-Newton	14
3.2.1	Resultado do Método Quasi-Newton	17
3.3	Conclusão	17
	REFERÊNCIAS	**19**

1 Introdução

As Bibliotecas utilizadas para o trabalhos são relacionadas ao projeto do Sympy[1]. O uso da biblioteca relacionado ao Sympy está representada pelo Algoritmo 1 e os métodos para solução linear são implementados pelo Algoritmo 2, tendo por exemplo a Matriz Hessiana, a Matriz Jacobiana e a Eliminação de Gauss (FRANCO, 2006; GOLUB, 2013). Além disso, as classes Log e Error para auxiliar na implementação do métodos numéricos (SPERANDIO; MENDES; SILVA, 2003). O primeiro, empregado para gerar o arquivo de log do algoritmo executado e informação do tempo de execução do método estudado, vide Algoritmo 3. Por fim, usado para o cálculo de erro absoluto, relativo e também do cálculo da norma da matriz, vide o Algoritmo 3. Portanto, o código fonte e os dados gerados pelos métodos estão disponível no Github [2].

```
1 from sympy import lambdify, diff, hessian, jacobi, cos, sin, pprint
2 from sympy.matrices import Matrix
3 from sympy.abc import x,y,w,z
4
5 from Error import *
6 from Log import *
```

Listing 1 – Biblioteca Sympy

[1] https://www.sympy.org/
[2] https://github.com/sswellington/GA018

```
def hessiana(function, c):
    '''  Hessian: init and set  '''
    h = hessian(function, c)
    return (lambdify(c, h))


def hessiana_inverse(function, c):
    '''  Hessian: init and set  '''
    h = (hessian(function, c)).inv()
    return (lambdify(c, h))


def jacobiana(function, c):
    '''  Jacobian: init and set  '''
    j = (Matrix([function]).jacobian(c))
    return (lambdify(c, j))


def jacobian_inversa(function, c):
    '''  Jacobian: init and set  '''
    j = (Matrix(function).jacobian(c)).inv()
    return (lambdify(c, j))


def jacobian_transpose(function, c):
    '''  Jacobian: init and set  '''
    j = (Matrix([function]).jacobian(c)).transpose()
    return (lambdify(c, j))


def gauss_jordan(matrix_a, matrix_b ):
    '''  Gauss: init and set  '''
    sol, params = matrix_a.gauss_jordan_solve(matrix_b)
    taus_zeroes = { tau:0 for tau in params }
    return sol.xreplace(taus_zeroes)
```

Listing 2 – Sistema Lineares

```
import csv
import sys
import time
class Log(object):
    _header = None
    def __init__(self):
        self._list = []
        self._start = time.time()
    def __repr__(self):
        return self._list
    def p_time(self):
        end = time.time()
        end -= self._start
        return end
    def time(self, path):
        end = time.time()
        end -= self._start
        f = open(path+'.txt', 'a')
        f.write(str(end))
        f.write('\n')
        f.close()
        return (end)
    def set_header(self, header):
        self._header = header
    def append(self, value):
        self._list.append( value)
    def list2file(self, path):
        with open(path+'.csv', 'w', newline='') as csvfile:
            spamwriter = csv.writer(csvfile,
                                    delimiter=',',
                                    quotechar='|',
                                    quoting=csv.QUOTE_MINIMAL)
            spamwriter.writerow(self._header)
            for l in (self._list) :
                spamwriter.writerow(l)
```

Listing 3 – Classe Log

```python
from sympy.matrices import Matrix


class Error(object):
    _absolute = None
    _relative = None
    _norm = None


    def __init__(self):
        _absolute = -10.987654321
        _relative = -10.987654321
        _norm = -10.987654321


    def absolute(self, current, previous) :
        self._absolute = (abs(current - previous))

        return self._absolute


    def relative(self, current, previous) :
        if (current == 0):
            return ('Error 16: Current is equal 0.0')
            breakpoint
        self._relative = abs( 1 -  (previous/current))

        return self._relative


    def matrix_norm(self, a, b) :
        self._norm = abs(Matrix(a).norm(1) - Matrix(b).norm(1))

        return self._norm
```

Listing 4 – Classe Error

2 Método de Newton na Otimização

$$f(x,y) = 1 - e^{\left(-\frac{(x-1)^2}{2*(\frac{3}{4})^2} + \frac{(y-1)^2}{2*(\frac{1}{2})^2}\right)} + \frac{1}{25} * \left((x-1)^2 + (y-2)^2\right) \tag{1}$$

Neste capítulo será aplicado Método de Newton para encontrar o mínimo da Equação 1.

2.1 Deduzir o método de Newton para o caso bidimensional

O método de Newton é utilizado para encontrar as raízes de uma função através da fórmula iterativas e aproximações. Esse método é útil quando não for possível encontrar as raízes de uma determinada função analiticamente. Para usar o método na função de uma variável, basta encontrar os pontos em que $f(x) = 0$, através da raiz de uma aproximação linear de F em um dado ponto inicial x_0. Ou seja, se r é a reta tangente a função no ponto x_0, então o próximo ponto x_1 do método seria o ponto em que $r(x1) = 0$ pois é onde a aproximação linear se anula. E esse processo ocorre até se encontrar o melhor ponto para a raiz de F. A reta tangente no ponto $(x_0, f(x_0))$ pode dar uma aproximação linear da função f no ponto $x - 0$, vide a Equação 2(RUGGIERO; LOPES, 2000)

Seja $F : \mathbb{R}^2 \to \mathbb{R}^2, F = (f_1, ..., f_n$, que possui objetivo de encontrar as soluções aproximadas para $F(x,y) = 0$ De uma forma equivalente a que foi realizado para funções de uma variável, considere $(x_0, y_0) \in \mathbb{R}^2$ um ponto inicial uma aproximação linear de F em torno de (x_0, y_0). A partir do momento em que é encontrado (x_1, y_1), é obtida a aproximação linear de F no ponto (x_1, y_1). O ponto seguinte (x_2, y_2) será aquele que anula esta aproximação linear, e assim será repetido o processo iterativamente. Portanto, dado $i \in \mathbb{R}$ e um ponto $(x_i, y_i) \in \mathbb{R}^2$ o novo iterando $(x_{i+1}, y_{i+1}) \in \mathbb{R}^2$ obtido pelo método de Newton. Sendo J a matriz jacobiana (RUGGIERO; LOPES, 2000).

$$\begin{aligned} x_1 &= x_1 - \frac{f(x_0)}{f'(x_0)} \\ x_1 &= x_1 - J(x_0)^{-1} F(x_0) \\ J(x)(x_1 - x_0) &= -F(x_0) \end{aligned} \tag{2}$$

2.2 Código computacional

```
MAX = 20
PATH = 'log/newton-opt/'
TOLERANCE = 0.00000001 # 10**(-8)
F = 1 - (exp(
            - ((((x-1)**2) / (2*(0.75**2)))
            + (( (y-2)**2) / (2*(0.5**2))))))
            + ( 0.04 * (((x-1)**2) + ((y-2)**2)))


if __name__ == "__main__" :
    seed = Matrix([[0.0],[0.0]])
    for i in range(1):
        n = optimization(F, seed, TOLERANCE, MAX)
        t = optimization_inv(F, seed, TOLERANCE, MAX)
```

Listing 5 – Execução do Método Newton para Otimização

```
def optimization_inv(f, xy, tol, nmax) :

    l = Log()
    e = Error()
    h = hessiana_inverse(f)
    j = jacobian_transpose(f)
    previous = xy

    for n in range(nmax

        hh = (h(float(xy[0]), float(xy[1])))
        jj = (j(float(xy[0]), float(xy[1])))

        xy = xy - (Matrix(hh) * Matrix(jj))

        e.matrix_norm(xy, previous)

        l.append([float(xy[0]), float(xy[1]),
                  float(previous[0]), float(previous[1]),
                  float(e._norm)])

        if (e._norm < tol) :
            l.set_header(['X axes'  ,'Y axes',  'W axes'  ,'Z axes',
                          'X-1 axes','Y-1 axes','W-1 axes','Z-1 axes',
                          'Matrix Norm'])

            l.list2file((PATH+'main-inversa'))
            l.time(PATH+'time-n-opt-inversa')

            return list(xy)
            breakpoint

        previous = xy

    return (xy)
```

Listing 6 – Método Newton para Otimização usando a Matriz Inversa

```
def optimization(f, xy, tol, nmax) :
    l = Log()
    e = Error()
    h = hessiana(f,[x,y])
    j = jacobian_transpose(f,[x,y])
    previous = xy

    for n in range(nmax) :
        hh = h(float(xy[0]), float(xy[1]))
        jj = j(float(xy[0]), float(xy[1]))

        L, U, _ = Matrix(hh).LUdecomposition()

        gauss = gauss_jordan(Matrix(L), Matrix(-jj))
        xy = xy + gauss_jordan(Matrix(U), Matrix(gauss))

        e.matrix_norm(xy, previous)
        l.append([float(xy[0]), float(xy[1]),
                  float(previous[0]), float(previous[1]),
                  float(e._norm)])

        if (e._norm < tol) :
            l.set_header(['X axes'  ,'Y axes',  'W axes'  ,'Z axes',
                          'X-1 axes','Y-1 axes','W-1 axes','Z-1 axes',
                          'Matrix Norm'])
            l.list2file((PATH+'main-inversa'))
            l.time(PATH+'time-n-opt-inversa')

            return xy
            breakpoint

        previous = xy
    return False
```

Listing 7 – Método Newton para Otimização usando a Decomposição LU

2.3 Validação

O método de Newton utilizado para encontrar o mínimo da Equação 1, tendo o ponto mínimo $\bar{X} = (1,2)$. Então para obter tal resultado foi empregado duas variações do método de Newton (RUGGIERO; LOPES, 2000; QUARTERONI; SACCO; SALERI, 2010). A primeira, calcula a inversa da matriz Hessiana, vide o Algoritmo 6. Já, a segundo utiliza a decomposição LU, disponível no Algoritmo 7 em alternativa da matriz inversa, que possui um alto custo computacional para a realizar o seu cálculo. Portanto, ao analisar os valores do tempo de execução de um método contra o outro é nítida a vantagem da decomposição LU em comparação a Matriz Inversa. O Algoritmo 6 que utiliza a Matriz Inversa [3] possui a média de execução de $5,774522781$ segundos e o desvio padrão de $0,4805941876$. Finalmente, o resultado do teste aplicado aos algoritmos, em suma o teste é a média e o desvio padrão do tempo de execução dos algoritmos, no qual é repetido 100 vezes para ser obter a média sem viés ou acaso, a fim de obter o melhor método. o Algoritmo 7 que usa a decomposição de LU [4] tem a média de execução de $0,0916364193$ segundos e o desvio padrão de $0,04318986438$. Logo, o Método de Newton com a decomposição de LU, vide o Algoritmo 7, é o ideal para encontrar o ponto mínimo dessa função.

2.4 Conclusão

O Método de Newton com decomposição LU aplicado a Equação 1, tendo a precisão de 10^{-8} e ponto de partida o ponto $x_0 = (0,0)$ o resultado está disponível na Tabela 1, no qual representa todas iterações do Método de Newton com decomposição LU. A Figura 1 ilustra a evolução do processo iterativo pela Tabela 1. Por fim a Figura 2 é a análise do erro obtido em cada iteração.

Tabela 1 – Método de Newton com decomposição LU - Evolução do processo iterativo

Iteração	**Eixo X**	**Eixo Y**	**Eixo X-1**	**Eixo Y-1**	**Norma da matriz**
1	1,061376269	2,275142385	0	0	3,336518654
2	0,9741326823	1,880732943	1,061376269	2,275142385	0,4816530295
3	1,001522677	2,007197686	0,9741326823	1,880732943	0,1538547379
4	0,999999692	1,999998508	1,001522677	2,007197686	0,0087221625
5	1	2	0,999999692	1,999998508	0,000001799718643
6	1	2	1	2	0

[3] https://github.com/sswellington/GA018/blob/main/log/newton-opt/time-n-sys-inversa.txt
[4] https://github.com/sswellington/GA018/blob/main/log/newton-opt/time-n-opt-lu.txt

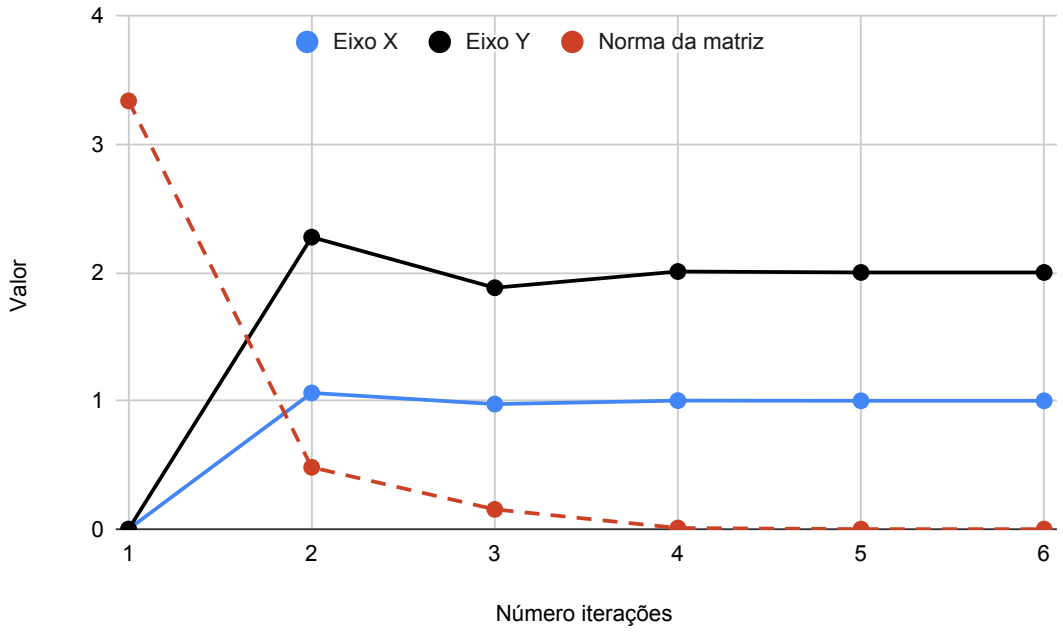

Figura 1 – Método de Newton com decomposição LU - Evolução do processo iterativo

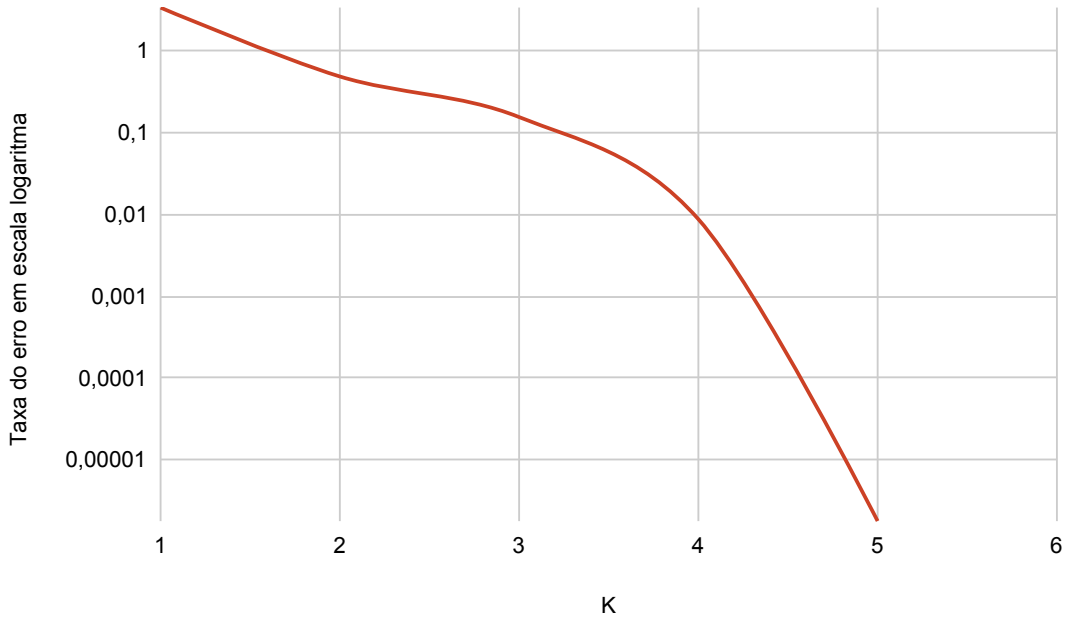

Figura 2 – Método de Newton com decomposição LU - Taxa de convergência obtido em cada iteração

3 Sistema Mecânico

Um sistema mecânico é constituído de quatro segmentos, todos com comprimento L, unidos entre si e a uma parede por articulações, vide a Figura 3. O momento em cada articulação é proporcional à deflexão com constante de proporcionalidade κ. Os segmentos são feitos de material homogêneo de peso P. A condição de equilíbrio pode ser expressa em termos dos ângulos $\theta_1,\ \theta_2,\ \theta_3,\ \theta_4$ conforme à Equação 3.

$$\begin{aligned}
\kappa(\theta_1) &= \frac{7PL}{2}\cos(\theta_1) + \kappa(\theta_2 - \theta_1), \\
\kappa(\theta_2 - \theta_1) &= \frac{5PL}{2}\cos(\theta_2) + \kappa(\theta_3 - \theta_2), \\
\kappa(\theta_3 - \theta_2) &= \frac{3PL}{2}\cos(\theta_3) + \kappa(\theta_4 - \theta_3), \\
\kappa(\theta_4 - \theta_3) &= \frac{1PL}{2}\cos(\theta_4)
\end{aligned} \tag{3}$$

Onde, $P = 15N,\ L = 1m$ e $\kappa = 100Nm/rad$.

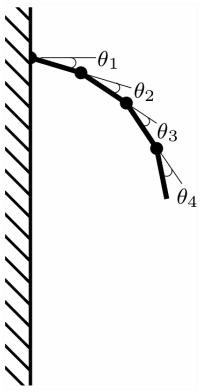

Figura 3 – Sistema mecânico

3.1 Newton

Código computacional para aproximar $\theta_1,\ \theta_2,\ \theta_3,\ \theta_4$ usando o Método de Newton.

```
def newton(fn, point, tol, nmax) :
    l = Log()
    e = Error()
    j = jacobiana(fn,[x,y,w,z])
    f = lambdify([x,y,w,z], fn)
    previous = point

    for n in range(nmax) :
        jj = j(float(point[0]),float(point[1]),
               float(point[2]),float(point[3]))
        ff = f(float(point[0]),float(point[1]),
               float(point[2]),float(point[3]))
        L, U, _ = Matrix(jj).LUdecomposition()
        g = gauss_jordan(Matrix(L), Matrix(-ff))
        point = point + gauss_jordan(Matrix(U), Matrix(g))
        e.matrix_norm(point, previous)

        l.append([float(point[0]), float(point[1]),
                  float(point[2]), float(point[3]),
                  float(previous[0]),float(previous[1]),
                  float(previous[2]),float(previous[3]),
                  float(e._norm)])

        if (e._norm < tol) :
            l.set_header(['X axes'  ,'Y axes',  'W axes'  ,'Z axes',
                          'X-1 axes','Y-1 axes','W-1 axes','Z-1 axes',
                          'Matrix Norm'])
            l.list2file((PATH_N+'main-lu-pr'))
            l.time(PATH_N+'time-n-sys-lu-pr')
            return point
            breakpoint
        previous = point
    return False
```

Listing 8 – Método Newton usando Decomposição LU

3.1.1 Resultados do Método de Newton

O Método de Newton aplicado a Equação 3, tendo a precisão de 10^{-8} e ponto de partida o ponto $x_0 = (1,1,1,1)$ o resultado está disponível na Tabela 2, no qual representa todas as iterações do Método Newton até a convergência do método.

Tabela 2 – Método de Newton para aproximação dos ângulos $\theta_1,\ \theta_2,\ \theta_3,\ \theta_4$

K	**Ângulo** θ_1	**Ângulo** θ_2	**Ângulo** θ_3	**Ângulo** θ_4	**Norma da Matriz**
1	1	1	1	1	0,09247623274
2	0,7203879844	1,033592424	1,154783643	1,183712181	0,027472394
3	0,7123207365	1,027283876	1,14831363	1,177085596	0,00003998152952
4	0,7123108904	1,027273674	1,148303632	1,177075661	0

3.2 Quasi-Newton

Código computacional para aproximar $\theta_1,\ \theta_2,\ \theta_3,\ \theta_4$ Métodos Quasi-Newton. Existem diversos Métodos Quasi-Newton, neste trabalho será utilizado o Método Quasi-Newton Broyden.

```
def quasi_newton(fn, point, tol, nmax) :
    l = Log()
    e = Error()
    j = jacobiana(fn,[x,y,w,z])
    f = lambdify([x,y,w,z], fn)
    previous = point

    jj = j(float(point[0]),float(point[1]),
          float(point[2]),float(point[3]))
    L, U, _ = Matrix(jj).LUdecomposition()

    for n in range(nmax) :
        ff = f(float(point[0]),float(point[1]),
              float(point[2]),float(point[3]))
        g = gauss_jordan(Matrix(L), Matrix(-ff))
        point = point + gauss_jordan(Matrix(U), Matrix(g))
        e.matrix_norm(point, previous)

        l.append([float(point[0]), float(point[1]),
                  float(point[2]), float(point[3]),
                  float(previous[0]),float(previous[1]),
                  float(previous[2]),float(previous[3]),
                  float(e._norm)])

        if (e._norm < tol) :
            l.set_header(['X axes'  ,'Y axes',  'W axes'  ,'Z axes',
                          'X-1 axes','Y-1 axes','W-1 axes','Z-1 axes',
                          'Matrix Norm'])
            l.list2file((PATH_Q+'main-jab-pr'))
            l.time(PATH_Q+'time-qn-sys_pr')
            return point
            breakpoint
        previous = point
    pprint(point)
    return False
```

Listing 9 – Método Quasi-Newton Broyden

```
MAX = 200
PATH_N = 'log/newton-sys/'
PATH_Q = 'log/quasi-newton/'
TOLERANCE = 0.00000001 # 10**(-8)

PL = 15      # weight * lenght
K = 100      # proportionality

M = Matrix([[K * (x    ) - ((7 * PL) / 2 ) * cos(x) - K * (y - x)],
            [K * (y - x) - ((5 * PL) / 2 ) * cos(y) - K * (w - y)],
            [K * (w - y) - ((3 * PL) / 2 ) * cos(w) - K * (z - w)],
            [K * (z - w) - ((    PL) / 2 ) * cos(z)]])


if __name__ == "__main__" :

    seed_pr = Matrix([[1],[1],
                      [1],[1]])

    seed = Matrix   ([[1],[2],
                      [3],[4]])

    for i in range(101):
        q = quasi_newton(M, seed_pr, TOLERANCE, MAX)
        n = newton(M, seed_pr, TOLERANCE, MAX)

        q = quasi_newton(M, seed, TOLERANCE, MAX)
        n = newton(M, seed, TOLERANCE, MAX)
```

Listing 10 – Execução do Método Newton e Quasi-Newton Broyden

3.2.1 Resultado do Método Quasi-Newton

O Método Quasi-Newton aplicado a Equação 3, tendo a precisão de 10^{-8} e ponto de partida o ponto $x_0 = (1,1,1,1)$ o resultado está disponível na Tabela 3, no qual representa todas iterações do Método.

Tabela 3 – Método Quasi-Newton para aproximação dos ângulos $\theta_1,\ \theta_2,\ \theta_3,\ \theta_4$

K	**Ângulo** θ_1	**Ângulo** θ_2	**Ângulo** θ_3	**Ângulo** θ_4	**Norma da Matriz**
1	1	1	1	1	0,09247623274
2	0,7203879844	1,033592424	1,154783643	1,183712181	0,02700406424
3	0,712694052	1,027425471	1,148309993	1,177042652	0,000453052468
4	0,7123318101	1,02728699	1,148314367	1,17708595	0,00005325202152
5	0,7123119537	1,027274207	1,148303874	1,177075829	0,000001875314228
6	0,7123109468	1,027273707	1,148303654	1,177075681	0,0000001255222459
7	0,7123108933	1,027273676	1,148303632	1,177075661	0,000000005706309203

3.3 Conclusão

A comparação entre o Método de Newton e o Método Quasi-Newton envolve diversos fatores. Em geral, o Método Quasi-Newton exige menor custo computacional em relação ao Método de Newton, em visto que o Quasi-Newton Boyde não utiliza derivada para obter a aproximação. Entretanto, o Método Quasi-Newton possui mais iterações, além de ser mais dependente do ponto inicial que o Método de Newton, podendo assim demorar mais ou mesmo não convergir. Ao realizar os teste para o ponto mais distante da raiz o Método Quasi-Newton realiza mais que o dobro de iterações que o Método de Newton vide a Tabela 4 em comparação a Tabela 5

Tabela 4 – Método de Newton para aproximação dos ângulos $\theta_1,\ \theta_2,\ \theta_3,\ \theta_4$, tendo ponto inicial $= (1,2,3,4)$

K	**Ângulo** θ_1	**Ângulo** θ_2	**Ângulo** θ_3	**Ângulo** θ_4	**Norma da Matriz**
1	1	2	3	4	8,800845906
2	0,4743766868	0,4328888603	0,01309229993	-0,2787962474	4,716893296
3	0,8899125749	1,41244724	1,748667692	1,865019884	1,825360327
4	0,7181529221	1,035762302	1,154749038	1,182022803	0,02568589428
5	0,7123182123	1,02728409	1,148313579	1,177085289	0,00003731266287
6	0,7123108904	1,027273674	1,148303632	1,177075661	0
7	0,7123108933	1,027273676	1,148303632	1,177075661	0,000000005706309203

Tabela 5 – Método Quasi-Newton para aproximação dos ângulos $\theta_1,\ \theta_2,\ \theta_3,\ \theta_4$, tendo ponto inicial = $(1,2,3,4)$

K	**Ângulo θ_1**	**Ângulo θ_2**	**Ângulo θ_3**	**Ângulo θ_4**	**Norma da Matriz**
1	1	2	3	4	8,800845906
2	0,4743766868	0,4328888603	0,01309229993	-0,2787962474	4,123504199
3	0,773266324	1,211545553	1,574927102	1,762919314	2,312279199
4	0,6407698264	0,8472990284	0,7977852189	0,7245250207	1,743217113
5	0,7556849842	1,141277512	1,378859023	1,477774688	1,229316534
6	0,6762538242	0,9353211357	0,9679975809	0,9447071324	0,9192274459
7	0,7366821104	1,090600278	1,274881539	1,341343192	0,6689558152
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
68	0,7123108901	1,027273674	1,14830363	1,177075659	0,000000008481138458

Portanto, neste caso o Método de Newton demandou menos recurso computacional que o O Método Quasi-Newton. Tal hipótese se confirma ao analisar o tempo de execução de cada método, o Método Quasi-Newton obteve a média $0,7413711548$ segundos e o desvio padrão $0,08539475997$. Já, o Método de Newton obteve a média $0,1109697819$ segundos e o desvio padrão $0,03354966481$. Além disso, com o ponto inicial $(1,1,1,1)$ o Método de Newton foi superior ao Método Quasi-Newto, obtendo à média $0,08055830002$ segundos e o desvio padrão $0,03348963636$ contra a média $0,09879040718$ segundos e o $0,03627166764$ desvio padrão.

Logo, confirma que neste caso o método de Newton se destacou em relação ao Método Quasi-Newton Broyden, sendo desvantagem do Método Quasi-Newton Broyden realizar aproximação por apenas um lado da reta em vista que a mantém a matriz Jacobina Fixa desda primeira iteração, de forma a iterar apenas a Função (RUGGIERO; LOPES, 2000).

Referências

FRANCO, N. B. *Cálculo numérico*. [S.l.]: Pearson, 2006.

GOLUB, C. F. V. L. G. H. *Matrix Computations*. [S.l.]: The Johns Hopkins University Press, 2013.

QUARTERONI, A.; SACCO, R.; SALERI, F. *Numerical mathematics*. [S.l.]: Springer Science & Business Media, 2010. v. 37.

RUGGIERO, M. A. G.; LOPES, V. L. d. R. *Cálculo numérico: aspectos teóricos e computacionais*. [S.l.]: Pearson, 2000.

SPERANDIO, D.; MENDES, J. T.; SILVA, L. H. M. e. *Cálculo numérico: características matemáticas e computacionais dos métodos numéricos*. [S.l.]: Prentice Hall, 2003.