Universidade Federal do Rio Grande do Norte
Centro de Tecnologia
Departamento de Engenharia da Computação e Automação
DCA0304 – Métodos Computacionais em Engenharia

Comparação entre Julia e outras linguagens de programação na eficiência de execução do método de Newton-Raphson para solução de sistema de equações não-lineares

André Rodrigues Bezerra Madruga
Bruno Matias de Sousa
José Ricardo Bezerra de Araújo
Levy Gabriel da Silva Galvão

Novembro
2018

Universidade Federal do Rio Grande do Norte
Centro de Tecnologia
Departamento de Engenharia da Computação e Automação
DCA0304 – Métodos Computacionais em Engenharia

Comparação entre Julia e outras linguagens de programação na eficiência de execução do método de Newton-Raphson para solução de sistema de equações não-lineares

Relatório técnico referente à execução prática dos métodos numéricos para a solução de sistemas de equações não-lineares realizado na disciplina de Métodos Computacionais em Engenharia, como requisito parcial para avaliação da terceira unidade da discplina antes mencionada.

Orientador: Profº. Drº. Paulo Sergio da Motta Pires

Novembro
2018

Sumário

Resumo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque et gravida mauris. Phasellus at ipsum in nisl iaculis consequat. Fusce vulputate nisl ipsum, quis egestas justo accumsan et. Morbi consequat tellus a eros eleifend congue. Aenean laoreet mattis nunc, at iaculis orci imperdiet in. Donec a diam in sem auctor fringilla. Aenean euismod odio vel arcu pretium, in vehicula urna ultricies.

Palavras-chaves: métodos. computacionais. engenharia.

1 Introdução

Muitos problemas da ciência da computação e de outras ciências podem serem abstraídos por meio de fórmulas e equações provenientes de uma linguagem matemática. Algumas dessas equações podem ser solucionadas de forma analítica utilizando a conceituação da literatura da área. Porém muitos outros problemas não possuem solução fechada por um método analítico, assim sendo necessário recorrer aos métodos iterativos, na maioria dos casos.

Ao longo dos anos os métodos iterativos solucionam problemas em diversas áreas, tais como: economia, engenharia, física, biologia, etc. Sua aplicação se baseia na aproximações para a solução do problema que melhoram em precisão de acordo com que aumentam o número de iterações. A extensa literatura na área de métodos iterativos só fortalece a importância de estudar a área.

A natureza repetitiva dos métodos iterativos sugere a sua execuação em recursos computacionais. Assim, a atividade "manufaturada"de realizar os cálculos é transferida para um computador, capaz de executá-las mais rapidamente.

Dessa forma, surgiu com o tempo uma tendência cada vez maior – de acordo com que a tecnologia se desenvolvia – de aliar a solução de problemas matemáticos aos métodos computacionais. Principalmente aqueles cuja solução analítica é difícil ou impossível. Um exemplo são os problemas de sistemas de equações não lineares que serão abordados no presente trabalho.

Para a obtenção das soluções desejadas foram utilizadas as linguagens de programação para realizar a comunicação entra a linguagem humana e matemática e a linguagem binária de máquina. Existem diversas linguagens com os mais diversos propósitos. Umas aplicadas ao gerenciamento de bancos de dados e outras com recursos dedicados aos métodos numéricos.

Com a pluralidade de escolhas, basta ao profissional escolher aquela linguagem que mais se adequa às suas necessidades. O mais procurado nos dias atuais na solução de problemas por métodos numéricos é a linguagem que seja mais rápida, que utilize menos recurso computacional e ofereça a resposta mais precisa.

Uma linguagem tida como forte candidata à preferida no cálculo numérico é a Julia. Uma linguagem bastante recente, cujo desenvolvimento começou em 2009 e teve a primeira versão de código aberto lançada em 2012.

2 Desenvolvimento

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque et gravida mauris. Phasellus at ipsum in nisl iaculis consequat. Fusce vulputate nisl ipsum, quis egestas justo accumsan et. Morbi consequat tellus a eros eleifend congue. Aenean laoreet mattis nunc, at iaculis orci imperdiet in. Donec a diam in sem auctor fringilla. Aenean euismod odio vel arcu pretium, in vehicula urna ultricies.

Vivamus ut pharetra diam. Aliquam metus sem, tristique ac dignissim eu, pretium id velit. Donec id tincidunt odio. Fusce vehicula ac est quis convallis. Nullam sollicitudin euismod dolor, eget blandit turpis hendrerit at. In bibendum suscipit odio, at consequat erat laoreet id. Donec in tellus at nulla gravida egestas. Suspendisse non elementum leo. Nam viverra sapien sed velit tempor scelerisque. Nunc accumsan odio eget mi vehicula, vel interdum libero gravida. Pellentesque vitae molestie diam, quis vestibulum libero. Aenean finibus sapien diam, ac sagittis magna placerat nec. Ut maximus eros felis, vel egestas diam convallis sit amet. Integer interdum elementum turpis, sit amet luctus nisi scelerisque at. Donec eleifend arcu dictum metus ullamcorper tempor. Aenean placerat, arcu id suscipit vehicula, velit ipsum viverra lorem, et pretium velit tellus quis libero.

2.1 Newton-Raphson para sistema de equações não-lineares

Nunc accumsan odio eget mi vehicula, vel interdum libero gravida. Pellentesque vitae molestie diam, quis vestibulum libero. Aenean finibus sapien diam, ac sagittis magna placerat nec. Ut maximus eros felis, vel egestas diam convallis sit amet. Integer interdum elementum turpis, sit amet luctus nisi scelerisque at. Donec eleifend arcu dictum metus ullamcorper tempor. Aenean placerat, arcu id suscipit vehicula, velit ipsum viverra lorem, et pretium velit tellus quis libero.

2.2 SENL propostos

Nunc accumsan odio eget mi vehicula, vel interdum libero gravida. Pellentesque vitae molestie diam, quis vestibulum libero. Aenean finibus sapien diam, ac sagittis magna placerat nec. Ut maximus eros felis, vel egestas diam convallis sit amet. Integer interdum elementum turpis, sit amet luctus nisi scelerisque at. Donec eleifend arcu dictum metus ullamcorper tempor. Aenean placerat, arcu id suscipit vehicula, velit ipsum viverra lorem, et pretium velit tellus quis libero.

2.3 Fluxograma do pseudocódigo

Nunc accumsan odio eget mi vehicula, vel interdum libero gravida. Pellentesque vitae molestie diam, quis vestibulum libero. Aenean finibus sapien diam, ac sagittis magna placerat nec. Ut maximus eros felis, vel egestas diam convallis sit amet. Integer interdum elementum turpis, sit amet luctus nisi scelerisque at. Donec eleifend arcu dictum metus ullamcorper tempor. Aenean placerat, arcu id suscipit vehicula, velit ipsum viverra lorem, et pretium velit tellus quis libero.

3 Resultados

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque et gravida mauris. Phasellus at ipsum in nisl iaculis consequat. Fusce vulputate nisl ipsum, quis egestas justo accumsan et. Morbi consequat tellus a eros eleifend congue. Aenean laoreet mattis nunc, at iaculis orci imperdiet in. Donec a diam in sem auctor fringilla. Aenean euismod odio vel arcu pretium, in vehicula urna ultricies.

Vivamus ut pharetra diam. Aliquam metus sem, tristique ac dignissim eu, pretium id velit. Donec id tincidunt odio. Fusce vehicula ac est quis convallis. Nullam sollicitudin euismod dolor, eget blandit turpis hendrerit at. In bibendum suscipit odio, at consequat erat laoreet id. Donec in tellus at nulla gravida egestas. Suspendisse non elementum leo. Nam viverra sapien sed velit tempor scelerisque. Nunc accumsan odio eget mi vehicula, vel interdum libero gravida. Pellentesque vitae molestie diam, quis vestibulum libero. Aenean finibus sapien diam, ac sagittis magna placerat nec. Ut maximus eros felis, vel egestas diam convallis sit amet. Integer interdum elementum turpis, sit amet luctus nisi scelerisque at. Donec eleifend arcu dictum metus ullamcorper tempor. Aenean placerat, arcu id suscipit vehicula, velit ipsum viverra lorem, et pretium velit tellus quis libero.

3.1 Raízes

Nunc accumsan odio eget mi vehicula, vel interdum libero gravida. Pellentesque vitae molestie diam, quis vestibulum libero. Aenean finibus sapien diam, ac sagittis magna placerat nec. Ut maximus eros felis, vel egestas diam convallis sit amet. Integer interdum elementum turpis, sit amet luctus nisi scelerisque at. Donec eleifend arcu dictum metus ullamcorper tempor. Aenean placerat, arcu id suscipit vehicula, velit ipsum viverra lorem, et pretium velit tellus quis libero.

3.2 Eficiência na execução

Nunc accumsan odio eget mi vehicula, vel interdum libero gravida. Pellentesque vitae molestie diam, quis vestibulum libero. Aenean finibus sapien diam, ac sagittis magna placerat nec. Ut maximus eros felis, vel egestas diam convallis sit amet. Integer interdum elementum turpis, sit amet luctus nisi scelerisque at. Donec eleifend arcu dictum metus ullamcorper tempor. Aenean placerat, arcu id suscipit vehicula, velit ipsum viverra lorem, et pretium velit tellus quis libero.

4 Conclusões

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque et gravida mauris. Phasellus at ipsum in nisl iaculis consequat. Fusce vulputate nisl ipsum, quis egestas justo accumsan et. Morbi consequat tellus a eros eleifend congue. Aenean laoreet mattis nunc, at iaculis orci imperdiet in. Donec a diam in sem auctor fringilla. Aenean euismod odio vel arcu pretium, in vehicula urna ultricies.

Vivamus ut pharetra diam. Aliquam metus sem, tristique ac dignissim eu, pretium id velit. Donec id tincidunt odio. Fusce vehicula ac est quis convallis. Nullam sollicitudin euismod dolor, eget blandit turpis hendrerit at. In bibendum suscipit odio, at consequat erat laoreet id. Donec in tellus at nulla gravida egestas. Suspendisse non elementum leo. Nam viverra sapien sed velit tempor scelerisque. Nunc accumsan odio eget mi vehicula, vel interdum libero gravida. Pellentesque vitae molestie diam, quis vestibulum libero. Aenean finibus sapien diam, ac sagittis magna placerat nec. Ut maximus eros felis, vel egestas diam convallis sit amet. Integer interdum elementum turpis, sit amet luctus nisi scelerisque at. Donec eleifend arcu dictum metus ullamcorper tempor. Aenean placerat, arcu id suscipit vehicula, velit ipsum viverra lorem, et pretium velit tellus quis libero.

Referências

Nenhuma citação no texto.

A. B. de Normas Técnicas. *NBR 10719: Apresentação de relatórios técnico-científicos.* 1989. Nenhuma citação no texto.

M. d. A. Marconi and E. M. Lakatos. *Fundamentos de metodologia científica.* 5. ed.-São Paulo: Atlas, 2003. Nenhuma citação no texto.

5 Apêndice

5.1 Fortran

```
program main
    implicit none
    double precision :: x0(3), x(size(x0)), tol, iter
    !Chute inicial
    x0(1) = 0.5
    x0(2) = 0.5
    x0(3) = 0.5
    iter = 100 !Numero de iteracoes
    tol = 1e-12 !Tolerancia

    x = newton_raph(x0, tol, .true., iter)
    write(*,*) x(1), x(2), x(3) !Solucao



contains

    !DECLARACAO DAS FUNCOES
    function f(p) result(res)
        IMPLICIT NONE
        double precision :: p(:), res(size(p))

        !Sistema de equacoes

        !eq1: 1/2*sin(x1*x2)-(x2/(4*pi))-(x1/2),
        !eq2: (1-1/(4*pi))*((exp(2*x1))-exp(1))-((exp(1)*x2)/pi)
           -2*exp(1)*x1)

        !eq3: x2+x3-exp(-x1)
        !eq4: x1+x3-exp(-x3)
        !eq5: x1+x2-exp(-x3)

        res(1) = p(2) + p(3) - exp(-p(1))
        res(2) = p(1) + p(3) - exp(-p(3))
        res(3) = p(1) + p(2) - exp(-p(3))
    end function f
    !MATRIZ JACOBIANA
```

```
    function jac(x) result(jc)
        IMPLICIT NONE
        double precision :: dx=1e-12, x(:), jc(size(x), size(x)),
            xn(size(x)), dy(size(x)), dif(size(x))
        integer :: n, i, k

        n = size(x)
        do k = 1,n !Calculo numerico da matriz jacobiana
            xn = x
            xn(k) = xn(k)+dx
            dy = f(xn) - f(x)
            dif = dy/dx
            do i = 1,n
                jc(i, k) = dif(i)
            end do
        end do
    end function jac
    ! METODO DA FATORACAO LU
    function LU(M, b) result(x)
        IMPLICIT NONE
        double precision :: b(:), M(:, :), x(size(b)), y(size(b))
            , U(size(b), size(b)), L(size(b), size(b)), c
        integer :: n, i, j, k

        n = size(b)

        do i = 1,n
            do j = 1,n
                L(i, j) = 0
                U(i, j) = 0
            end do
        end do
        do i = 1,n
            x(i) = 0
            y(i) = 0
            L(i, i) = 1 !Preenche L com a Matriz Identidade
        end do

        U = M !Atribui a Matriz J a Matriz U

        do i = 1,(n-1)
```

```
            do k = (i+1),n
                c = U(i, k) / U(i, i)
                L(k, i) = c !Armazena o multiplicador
                do j = 1,n
                    U(k, j) = U(k, j) - c*U(i, j) !Multiplica com o pivo da linha e subtrai
end
                end do
            end do
            do k = (i+1),n
                U(k, i) = 0
            end do
        end do

        !Resolve o Sistema Ly=b
        do i = 1,n
            y(i) = b(i) / L(i, i)
            do k = 1,(i-1)
                y(i) = y(i) - y(k)*L(i, k)
            end do
        end do

        x = y
        !Resolve o Sistema Ux=y
        do i = n,1,-1
            do k = (i+1),n
                x(i) = x(i) - x(k)*U(i, k)
            end do
            x(i) = x(i)/U(i, i)
        end do
    end function LU
    !METODO NEWTON_RAPHSON
    function newton_raph(x0, tol, iter, n_tot) result(x)
        IMPLICIT NONE
        double precision :: x0(:), x(size(x0)), tol, e, func(size(x0)), S(size(x0)), jc(size(x0), size(x0))
        integer :: n_tot, n0, n=0, i
        logical :: iter

        n0 = size(x0)

```

```
        do i = 1,n0
            x(i) = x0(i)
        end do
        e = maxval(abs(f(x)))

        do while ((e>tol).and.(n<=n_tot))
            n = n+1
            func = f(x)
            e = maxval(abs(func))
            jc = jac(x)
            if (size(func)==1) then
                x = x - (func(1)/jc(1, 1))
            else
                S = LU(jc, -func)
                x = x+S
            end if
        end do

        if (iter .eqv. .true.) then
            write(*,*) "Total de iteracoes: ", n
        end if
        if (n>=n_tot) then
            write(*,*) "Processo parou, numero de iteracoes
               limite atingido", n
        end if

    end function newton_raph

end program main
```

5.2 Julia

```
# METODO DA FATORACAO LU
function LU(matriz, vetor_b)
    n = length(vetor_b)
    x = zeros(n)
    L = zeros(n, n)
    for i = 1:n  #Preenche L com a Matriz Identidade
        L[i, i] = 1
    end
    U=zeros(n,n)
    U = matriz #Atribui a Matriz J a Matriz U

    for i = 1:n-1
        for k = i+1:n
            c = U[k, i] / U[i, i]
            L[k, i] = c #Armazena o multiplicador
            for j = 1:n
                U[k, j] = U[k, j] - c*U[i, j] # Multiplica com o
                   pivo da linha e subtrai
            end
        end
        for k = i+1:n
            U[k, i] = 0
        end
    end

    # Resolve o Sistema Ly=b
    y = zeros(n)
    for i = 1:n
        y[i] = vetor_b[i] / L[i, i]

        for k = 1:i-1
            y[i] = y[i]  - y[k]*L[i, k]
        end
    end
    n = length(y)

    # Resolve o Sistema Ux=y
    x = copy(y)
    for i = (n:-1:1)
        for k = 1+i:n
```

```
            x[i] = x[i] - x[k]*U[i, k]
        end
        x[i] = x[i]/U[i, i]
    end

    return x
end


#DECLARACAO DAS FUNCOES
function f(p)
    #Sistema de equacoes

    #eq1: 1/2*sin(x1*x2)-(x2/(4*pi))-(x1/2),
    #eq2: (1-1/(4*pi))*((exp(2*x1))-exp(1))-((exp(1)*x2)/pi)-2*
       exp(1)*x1)

    #eq3: x2+x3-exp(-x1)
    #eq4: x1+x3-exp(-x3)
    #eq5: x1+x2-exp(-x3)
    a = p[2] + p[3] - exp(-p[1])
    b = p[1] + p[3] - exp(-p[3])
    c = p[1] + p[2] - exp(-p[3])
    return [a c b]
end
#MATRIZ JACOBIANA
function jac(x, dx=1e-10)
    n = length(x)
    J = zeros(n, n)
    for j = 1:n #Calculo numerico da matriz jacobiana
        xn = copy(x)
        xn[j] = xn[j] +dx
        dy = f(xn) - f(x)
        dif = dy/dx
        for i = 1:n
            J[i, j] = dif[i]
        end
    end
    return J
end
#METODO NEWTON_RAPHSON
```

```
function newton_raph(x0, tol, iter, n_tot)
    #DECLARACAO DAS VARIAVEIS
    tol = abs.(tol) #Modulos de tol e iter
    n_tot = abs.(n_tot)
    x = convert(Array{Float64}, x0) #Cria o vetor x
    e = maximum(abs.(f(x))) #Erro
    n = 0 #Atribue zero a variavel de interacoes totais

    while (e>tol)&(n<=n_tot)
        n += 1
        F = f(x)
        e = maximum(abs.(F))
        J = jac(x)
        if length(F) == 1
            x = x - F / J
        else
            S = LU(J, -F)
            x = x+S
        end
    end
    if iter==true
        print("Total de Iteracoes: ", string(n))
    end
    if n>=n_tot  #Condicao de parada
        print("Processo parou, numero de iteracoes limite
           atingido")
    else
        return x
    end
end


    x0 = [0.5, 1, 5] # Chute inicial
    iter  = 100 # Numero de iteracoes
    tol = 1e-12 # Tolerancia
    @timev x = newton_raph(x0, tol, true, iter)
    print("\nSolucao= ",x, "\n")
```

5.3 Python

```
import numpy as np
import time
from numpy import sin, exp
from math import pi

#DECLARACAO DAS FUNCOES
def func(p):
    '''
    Sistema de equacoes

        eq1: 1/2*sin(x1*x2)-(x2/(4*pi))-(x1/2),
        eq2: (1-1/(4*pi))*((exp(2*x1))-exp(1))-((exp(1)*x2)/pi)
            -2*exp(1)*x1)

        eq3: x2+x3-exp(-x1)
        eq4: x1+x3-exp(-x3)
        eq5: x1+x2-exp(-x3)
    '''

    x1, x2 = p

    return np.array((1/2*sin(x1*x2)-(x2/(4*pi))-(x1/2),
                    (1 - 1 / (4 * pi)) * ((exp(2 * x1)) - exp(1)
                        ) - ((exp(1) * x2) / pi) - 2 * exp(1) * x1
                        ))


#MATRIZ JACOBIANA
def jac(f, x, dx=1e-10):
    x = np.array(x)
    n = len(x)
    J = np.zeros((n, n))
    for j in range(n):          #Calculo numerico da matriz
       jacobiana
        xn = np.copy(x)
        xn[j] = xn[j] + dx
        dy = np.array(f(xn)) - np.array(f(x))
        dif = dy / dx
        for i in range(n):
            J[i, j] = dif[i]
```

```
    return J


# METODO DA FATORACAO LU
def LU(matriz, vetor_b):

    n = len(matriz)

    L = [[0 for i in range(n)] for i in range(n)]
    for i in range(n):
        L[i][i] = 1         #Preenche L com a Matriz Identidade

    U = [[0 for i in range(n)] for i in range(n)]
    for i in range(n):
        for j in range(n):
            U[i][j] = matriz[i][j]       #Atribui a Matriz J a
               Matriz U


    # Encontra as Matrizes U e L
    for i in range(n-1):
        for k in range(i + 1, n):
            c = U[k][i] / U[i][i]
            L[k][i] = c  # Armazena o multiplicador
            for j in range(n):
                U[k][j] -= c * U[i][j]  # Multiplica com o pivo
                   da linha e  subtrai


        for k in range(i + 1, n):
            U[k][i] = 0


    # Resolve o Sistema Ly=b
    y = [0 for i in range(n)]

    for i in range(0, n, 1):
        y[i] = vetor_b[i] / L[i][i]
        for k in range(0, i, 1):
            y[i] -= y[k] * L[i][k]

```

```

    # Resolve o Sistema Ux=y
    x = [y[i] for i in range(n)]

    for i in range(n-1, -1, -1):
        for k in range(i+1, n):
            x[i] -= x[k] * U[i][k]
        x[i] /= U[i][i]
    return x

#METODO NEWTON_RAPHSON
def newton_raph(func, x0, tol, iter):

    #DECLARACAO DAS VARIAVEIS
    tol = abs(tol)      #Modulos de tol e iter
    iter = abs(iter)
    x = (np.array(x0)).astype(np.float)    #Cria o vetor x
    e = max(abs(np.array(func(x))))  #Erro
    n = 0      #Atribue zero a variavel de interacoes totais

    #Print da interacao 0
    if iter == 0:
        print('\nTotal de Iteracoes: ' + str(n))
        return x

    else:

        while (e > tol and n <= iter):
            n += 1
            F = np.array(func(x))
            e = max(abs(F))
            J = jac(func, x)
            if len(F) == 1:
                x = x - F / J
            else:

                S = LU(J, -F)
                x = x + S

        print('\nTotal de Iteracoes: ' + str(n))
        if n >= iter:  #Condicao de parada
```

```
            print("PROCESSO PAROU, numero de iteracoes limite
               atingido!")
            return x
        else:
            return x


if __name__ == "__main__":

    inicio = time.time()
    x0 = [0.5, -2]  #Chute Inicial
    iter = 100      #Quantidade de Interacoes maximas
    tol = 1e-12     #tolerancia

    x = newton_raph(func, x0, tol, iter)   #Chama o metodo de
       Newton_Raphson

    print('Solucao: {} ' .format(x))

    fim = time.time()
    print('Tempo gasto: {}'.format(fim - inicio))
```