**UNIVERSIDADE DO ESTADO** 

**DO RIO DE JANEIRO**

****

**INSTITUTO POLITÉCNICO**

**GRADUAÇÃO EM**

**ENGENHARIA DA COMPUTAÇÃO**

Vitor Saraiva

Gabriel Calheias

**Métodos Numéricos para Equações Diferenciais**

**Trabalho 1**

Nova Friburgo

2022

# As etapas do trabalho referentes às modelagem computacionais dos métodos de **Runge-Kutta de 3ª Ordem** e de Problema de Contorno, utilizando-se de **Jacobi**, foram executadas na linguagem de programação **Python**.

As bibliotecas utilizadas neste trabalho são **matplotlib**para o plot de gráficos além de **math** e **numpy** para manipulações aritméticas.

# 

# Parte 1

A parte 1 do trabalho foi dedicada ao encontro de aproximações numéricas para as equações de concentração de três reagentes. Em **Python**, para a realização destas codificações com as seguintes fórmulas:

= −  *,*

= −  *,*

= −  −

Com as condições iniciais pré-estabelecidas nas orientações do trabalho para = e = , inicial, até um máximo.

Aplicando valores facultativos, seguindo determinada regra, para as demais variáveis pode-se utilizar do método de **Runge-Kutta de 3ª Ordem** para obter as aproximações numéricas.

Para , e , cujos valores são facultativos em torno de uma dezena, foram escolhidos inicialmente 11, 12 e 13, respectivamente. Já, para os valores iniciais de e , facultativos por volta de algumas dezenas, escolheu-se 64 e 76, respectivamente. Para o máximo, foi escolhido 2 segundos, com o ∆ = 0.001.

import matplotlib.pyplot as plt

import numpy

import math

fCa = []

fCb = []

fCc = []

tempo = []

# Por volta de uma dezena

alfa = 11

beta = 12

gamma = 13

# Por volta de algumas dezenas

A = 64

B = 0

C = 76

t = 0

# Por volta de algumas unidades

tf = 2

# Passo de Tempo

# Max Calculado = 0.0014 ----- 0.0015 os valores de concentração se tornam Nan

Δt = 0.001

def dca(ca, cb, cc): return -alfa\*ca\*cc + cb

def dcb(ca, cb, cc): return beta\*ca\*cc - cb

def dcc(ca, cb, cc): return -gamma\*ca\*cc + cb - 2\*cc

def RungeKutta(A, B, C, t, tf, Δt):

fa = A

fb = B

fc = C

while (t < tf):

print("\nt = " + str(t))

ka1 = dca(fa, fb, fc)

kb1 = dcb(fa, fb, fc)

kc1 = dcc(fa, fb, fc)

ka2 = dca(fa + (ka1\*(Δt/2)), fb + (kb1\*(Δt/2)), fc + (kc1\*(Δt/2)))

kb2 = dcb(fa + (ka1\*(Δt/2)), fb + (kb1\*(Δt/2)), fc + (kc1\*(Δt/2)))

kc2 = dcc(fa + (ka1\*(Δt/2)), fb + (kb1\*(Δt/2)), fc + (kc1\*(Δt/2)))

ka3 = dca(fa - (ka1\*Δt) + (2\*ka2\*Δt), fb - (kb1\*Δt) + (2\*kb2\*Δt), fc - (kc1\*Δt) + (2\*kc2\*Δt))

kb3 = dcb(fa - (ka1\*Δt) + (2\*ka2\*Δt), fb - (kb1\*Δt) + (2\*kb2\*Δt), fc - (kc1\*Δt) + (2\*kc2\*Δt))

kc3 = dcc(fa - (ka1\*Δt) + (2\*ka2\*Δt), fb - (kb1\*Δt) + (2\*kb2\*Δt), fc - (kc1\*Δt) + (2\*kc2\*Δt))

ka = (Δt\*(ka1+(4\*ka2)+ka3))/6

kb = (Δt\*(kb1+(4\*kb2)+kb3))/6

kc = (Δt\*(kc1+(4\*kc2)+kc3))/6

fa = fa + ka

fb = fb + kb

fc = fc + kc

print("RK3 dCa = " + str(fa))

print("RK3 dCb = " + str(fb))

print("RK3 dCc = " + str(fc))

fCa.append(fa)

fCb.append(fb)

fCc.append(fc)

tempo.append(t)

t = t + Δt

print("Alpha = " + str(alfa))

print("Beta = " + str(beta))

print("Gamma = " + str(gamma))

print("\nA = " + str(A))

print("B = " + str(B))

print("C = " + str(C))

print("Δt = " + str(Δt))

RungeKutta(A, B, C, t, tf, Δt)

plt.plot(tempo, fCa, color = 'red', marker = 'o', label='dCa/dt = -αCaCc + Cb')

plt.plot(tempo, fCb, color = 'green', marker = 'D', label='dCb/dt = βCaCc - Cb')

plt.plot(tempo, fCc, color = 'blue', marker = '^', label='dCc/dt = -γCaCc + Cb - 2Cc')

plt.xlabel('t')

plt.ylabel('f(t)')

plt.grid(True)

plt.title('Δt = ' + str(Δt))

plt.legend()

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 