Métodos Numéricos para Equações Diferenciais

Trabalho 1

Alunos:

Gustavo Dias de Oliveira
Matrícula: 202010078511

Letícia Bussinger das Neves
Matrícula: 202010076111

Nova Friburgo - 2022

Objetivo do Trabalho

Na primeira parte do trabalho, é solicitado que determinemos as concentrações de três reagentes representados por equações, para tempo de $t = 0$ até $t = t_{máx}$, utilizando o método de Runge-Kutta de 3ª ordem para tal finalidade, além de realizar um estudo de refinamento de malha computacional e um estudo de análise de sensibilidade quando da variação dos parâmetros do modelo.

Já na segunda parte do trabalho, é solicitado que determinemos a concentração de um componente químico ao longo de um tubo a partir da equação dada, além de realizar um estudo de refinamento de malha computacional e um estudo de análise de sensibilidade quando da variação dos parâmetros do modelo.

Desenvolvimento e Análise do Trabalho

Para a primeira parte do trabalho, determinamos o conjunto de dados a serem utilizados respeitando as especificações dadas pelo professor no PDF disponibilizado, logo, o conjunto de dados utilizado é dado por:

$$\alpha = 10 \qquad \beta = 12 \qquad \gamma = 14$$
$$A = 20 \qquad B = 0 \qquad C = 25$$
$$T = 0 \qquad T_{máx} = 1 \qquad \Delta T = 0.00025$$

Logo, as equações e as condições iniciais ficaram definidas como:

$$\frac{dC_a}{dt} = -10C_aC_c + C_b$$
$$\frac{dC_b}{dt} = 12C_aC_c - C_b$$
$$\frac{dC_c}{dt} = -14C_aC_c + C_b - 2C_c$$

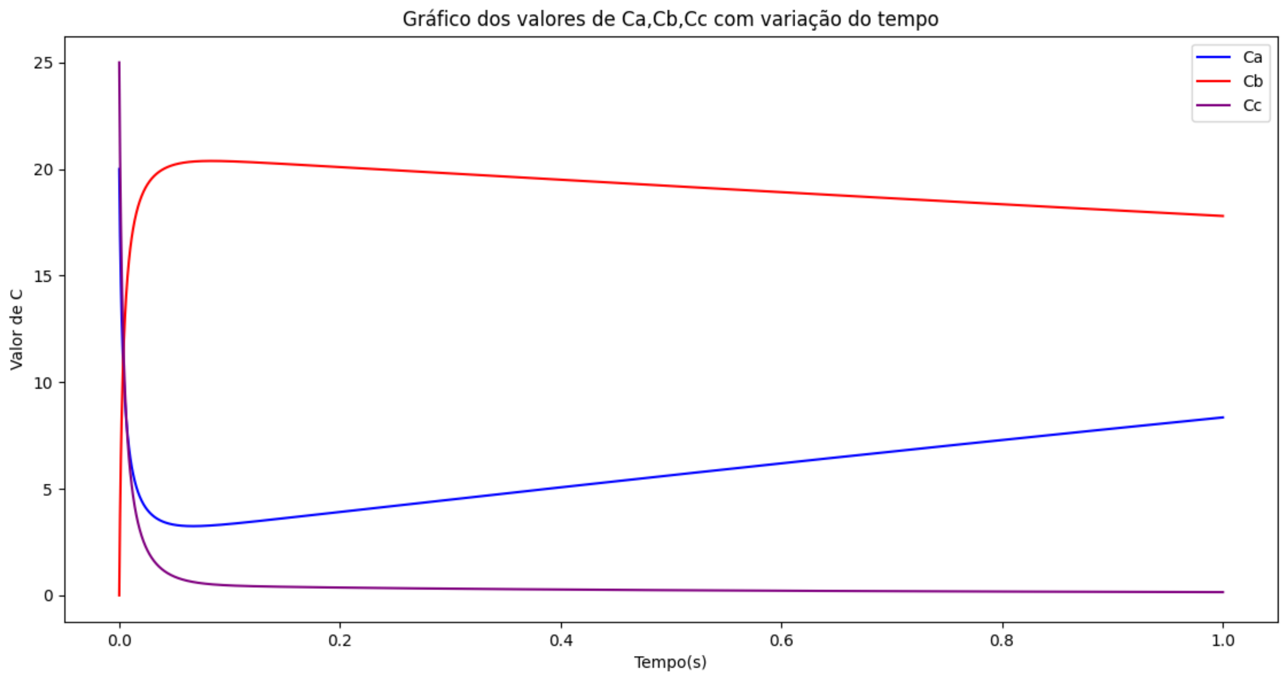
$$C_a = 20$$
$$C_b = 0$$
$$C_c = 25$$

Dessa mesma forma, para realizar o estudo de refinamento de malha computacional utilizamos da técnica de dobrar o valor de ΔT, multiplicando por 2 por 4 iterações, obtendo o vetor de valores $\boldsymbol{\Delta T}$ = **[0.00025, 0.0005, 0.001, 0.002, 0.004]**. Já para o estudo de análise de sensibilidade, utilizamos da técnica de dividir os valores por 2 a cada iteração para determinar os novos valores de α, β e γ, obtendo os vetores de valores: $\boldsymbol{\alpha}$ = **[10, 5, 2.5, 1.25, 0.625]**, $\boldsymbol{\beta}$ = **[12, 6, 3, 1.5, 0.75]**, $\boldsymbol{\gamma}$ = **[14, 7, 3.5, 1.75, 0.875]**.

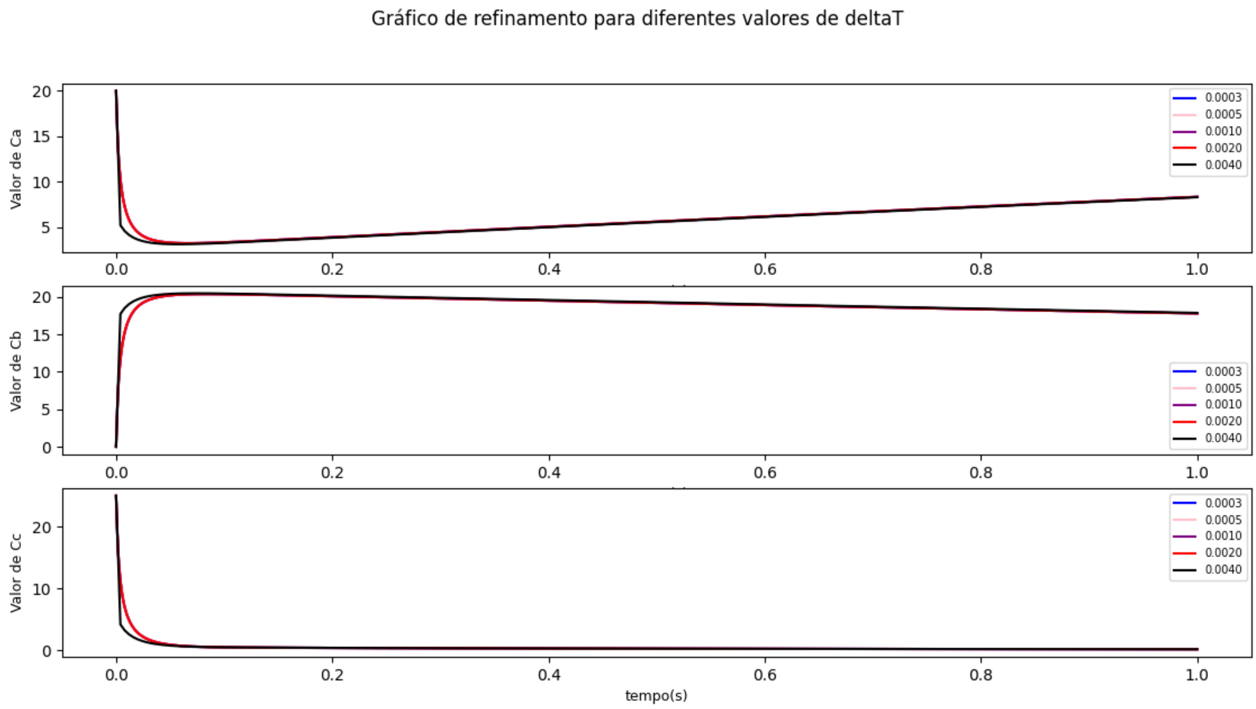
Portanto, utilizando desses valores para a resolução do problema, temos que:

$$C_a = 8.347400410344829$$
$$C_b = 17.796669700522084$$
$$C_c = 0.15070529065776084$$

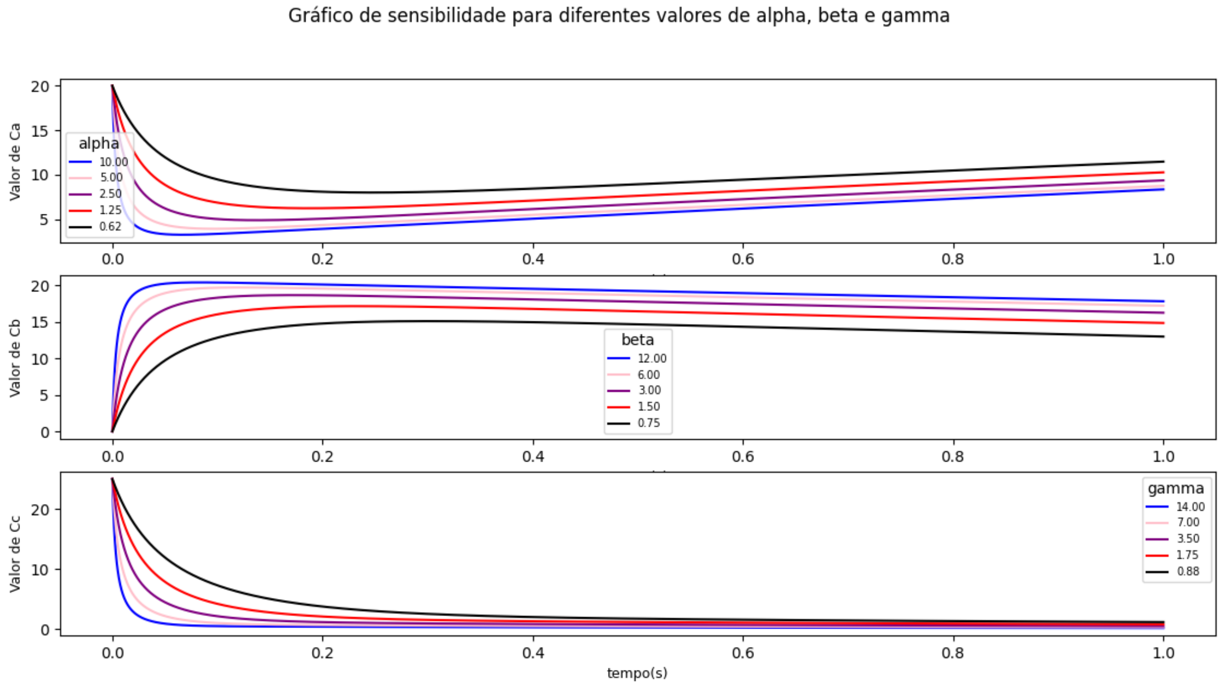
Com o gráfico sendo:



Seu refinamento de malha computacional é dado pelo gráfico:



Sua análise de sensibilidade é dada pelo gráfico:



E, por fim, abaixo tem-se o código da primeira parte do trabalho:

```
import matplotlib.pyplot as plt

# uma dezena
alpha = 10
beta = 12
gamma = 14
# algumas dezenas
Ca = 20
Cb = 0
Cc = 25
# algumas unidades
Tmax = 1
T = 0
deltaT = 0.00025


def derivada_Ca(Ca, Cb, Cc, alpha):
    return -alpha * Ca * Cc + Cb


def derivada_Cb(Ca, Cb, Cc, beta):
    return beta * Ca * Cc - Cb
```

```python
def derivada_Cc(Ca, Cb, Cc, gamma):
    return -gamma * Ca * Cc + Cb - 2 * Cc


def rungeKutta(T, deltaT, Ca, Cb, Cc, alpha, beta, gamma):
    k1 = [0, 0, 0]
    k2 = [0, 0, 0]
    k3 = [0, 0, 0]
    y = [Ca, Cb, Cc]
    vetor_Ca = [Ca]
    vetor_Cb = [Cb]
    vetor_Cc = [Cc]
    vetor_T = [T]

    while T < Tmax:

        k1[0] = derivada_Ca(Ca, Cb, Cc, alpha)
        k1[1] = derivada_Cb(Ca, Cb, Cc, beta)
        k1[2] = derivada_Cc(Ca, Cb, Cc, gamma)

        k2[0] = derivada_Ca(
            Ca + k1[0] * (deltaT / 2),
            Cb + k1[1] * (deltaT / 2),
            Cc + k1[2] * (deltaT / 2),
            alpha,
        )
        k2[1] = derivada_Cb(
            Ca + k1[0] * (deltaT / 2),
            Cb + k1[1] * (deltaT / 2),
            Cc + k1[2] * (deltaT / 2),
            beta,
        )
        k2[2] = derivada_Cc(
            Ca + k1[0] * (deltaT / 2),
            Cb + k1[1] * (deltaT / 2),
            Cc + k1[2] * (deltaT / 2),
            gamma,
        )

        k3[0] = derivada_Ca(
            Ca - (k1[0] * deltaT) + (2 * k2[0] * deltaT),
            Cb - (k1[1] * deltaT) + (2 * k2[1] * deltaT),
            Cc - (k1[2] * deltaT) + (2 * k2[2] * deltaT),
            alpha,
        )
        k3[1] = derivada_Cb(
            Ca - (k1[0] * deltaT) + (2 * k2[0] * deltaT),
            Cb - (k1[1] * deltaT) + (2 * k2[1] * deltaT),
            Cc - (k1[2] * deltaT) + (2 * k2[2] * deltaT),
```

```
            beta,
        )
        k3[2] = derivada_Cc(
            Ca - (k1[0] * deltaT) + (2 * k2[0] * deltaT),
            Cb - (k1[1] * deltaT) + (2 * k2[1] * deltaT),
            Cc - (k1[2] * deltaT) + (2 * k2[2] * deltaT),
            gamma,
        )

        y[0] = y[0] + (1 / 6) * (k1[0] + 4 * k2[0] + k3[0]) * deltaT
        y[1] = y[1] + (1 / 6) * (k1[1] + 4 * k2[1] + k3[1]) * deltaT
        y[2] = y[2] + (1 / 6) * (k1[2] + 4 * k2[2] + k3[2]) * deltaT

        Ca = y[0]
        Cb = y[1]
        Cc = y[2]

        vetor_Ca.append(Ca)
        vetor_Cb.append(Cb)
        vetor_Cc.append(Cc)
        T = T + deltaT
        vetor_T.append(T)

    return Ca, Cb, Cc, vetor_Ca, vetor_Cb, vetor_Cc, vetor_T


resultados_RK = rungeKutta(T, deltaT, Ca, Cb, Cc, alpha, beta, gamma)

print("\nDEPOIS DAS ITERAÇOES, AS APROXIMACOES FORAM:")
print(
    "\nCa = {}\nCb = {}\nCc = {}".format(
        resultados_RK[0], resultados_RK[1], resultados_RK[2]
    )
)


def grafico_RK(resultados_RK):

    plt.title("Gráfico dos valores de Ca,Cb,Cc com variação do tempo")
    plt.plot(resultados_RK[6], resultados_RK[3], color="blue",
label="Ca")
    plt.plot(resultados_RK[6], resultados_RK[4], color="red", label="Cb")
    plt.plot(resultados_RK[6], resultados_RK[5], color="purple",
label="Cc")
    plt.legend()
    plt.xlabel("Tempo(s)")
    plt.ylabel("Valor de C")
    plt.show()
```

```python
grafico_RK(resultados_RK)


def refinamento():
    vetor_deltaT = [deltaT]

    for i in range(0, 4):
        vetor_deltaT.append(vetor_deltaT[i] * 2)

    fig, (grafico_Ca, grafico_Cb, grafico_Cc) = plt.subplots(3, 1)

    for i in range(0, 5):
        resultados_RK = rungeKutta(T, vetor_deltaT[i], Ca, Cb, Cc, alpha,
beta, gamma)
        cor = ["blue", "pink", "purple", "red", "black"]

        grafico_Ca.plot(
            resultados_RK[6],
            resultados_RK[3],
            color=cor[i],
            label="%.4f" % (float(vetor_deltaT[i])),
        )
        grafico_Ca.legend(fontsize=7)
        grafico_Ca.set_xlabel("tempo(s)", fontsize=9)
        grafico_Ca.set_ylabel("Valor de Ca", fontsize=9)

        grafico_Cb.plot(
            resultados_RK[6],
            resultados_RK[4],
            color=cor[i],
            label="%.4f" % (float(vetor_deltaT[i])),
        )
        grafico_Cb.legend(fontsize=7)
        grafico_Cb.set_xlabel("tempo(s)", fontsize=9)
        grafico_Cb.set_ylabel("Valor de Cb", fontsize=9)

        grafico_Cc.plot(
            resultados_RK[6],
            resultados_RK[5],
            color=cor[i],
            label="%.4f" % (float(vetor_deltaT[i])),
        )
        grafico_Cc.legend(fontsize=7)
        grafico_Cc.set_xlabel("tempo(s)", fontsize=9)
        grafico_Cc.set_ylabel("Valor de Cc", fontsize=9)

    plt.suptitle("Gráfico de refinamento para diferentes valores de
deltaT")
    plt.show()
```

```python
refinamento()


def sensibilidade():

    sensibilidade_alpha = [alpha]
    sensibilidade_beta = [beta]
    sensibilidade_gamma = [gamma]

    i = 0
    for i in range(0, 4):
        sensibilidade_alpha.append(sensibilidade_alpha[i] / 2)
        sensibilidade_beta.append(sensibilidade_beta[i] / 2)
        sensibilidade_gamma.append(sensibilidade_gamma[i] / 2)

    fig, (grafico_alpha, grafico_beta, grafico_gamma) = plt.subplots(3,
1)

    for i in range(0, 5):
        resultados_RK = rungeKutta(
            T,
            deltaT,
            Ca,
            Cb,
            Cc,
            sensibilidade_alpha[i],
            sensibilidade_beta[i],
            sensibilidade_gamma[i],
        )
        cor = ["blue", "pink", "purple", "red", "black"]

        grafico_alpha.plot(
            resultados_RK[6],
            resultados_RK[3],
            color=cor[i],
            label="%.2f" % (sensibilidade_alpha[i]),
        )
        grafico_alpha.legend(title="alpha", fontsize=7)
        grafico_alpha.set_xlabel("tempo(s)", fontsize=9)
        grafico_alpha.set_ylabel("Valor de Ca", fontsize=9)

        grafico_beta.plot(
            resultados_RK[6],
            resultados_RK[4],
            color=cor[i],
            label="%.2f" % (sensibilidade_beta[i]),
        )
```

```python
        grafico_beta.legend(title="beta", fontsize=7)
        grafico_beta.set_xlabel("tempo(s)", fontsize=9)
        grafico_beta.set_ylabel("Valor de Cb", fontsize=9)

        grafico_gamma.plot(
            resultados_RK[6],
            resultados_RK[5],
            color=cor[i],
            label="%.2f" % (sensibilidade_gamma[i]),
        )
        grafico_gamma.legend(title="gamma", fontsize=7)
        grafico_gamma.set_xlabel("tempo(s)", fontsize=9)
        grafico_gamma.set_ylabel("Valor de Cc", fontsize=9)

    plt.suptitle(
        "Gráfico de sensibilidade para diferentes valores de alpha, beta
e gamma"
    )
    plt.show()


sensibilidade()
```

Já para a segunda parte do trabalho, determinamos o conjunto de dados a serem utilizados respeitando as especificações dadas pelo professor no PDF disponibilizado, logo, o conjunto de dados utilizado é dado por:

$$\alpha = 1$$
$$\beta = 2$$
$$k = 2 \cdot 10^{-6}$$
$$D = 1 \cdot 10^{-6}$$
$$C_e = 0{,}1$$
$$C_d = 0$$
$$L = 6$$

Para a realização desta parte do trabalho, precisamos trabalhar a equação disponibilizada, e desta forma, obtivemos o seguinte modelo:

$$-C_{i+1} + \left(2 + \frac{\Delta x^2 k}{D}\right) C_i - C_{i-1} = 0$$

Utilizando-se do valor de **4 nós internos**, temos que:

Para i=2,

$$-C_3 + \left(2 + \frac{\Delta x^2 k}{D}\right) C_2 = C_e = 0{,}1$$

Para i=3,

$$-C_4 + \left(2 + \frac{\Delta x^2 k}{D}\right) C_3 - C_2 = 0$$

Para i=4,

$$-C_5 + \left(2 + \frac{\Delta x^2 k}{D}\right) C_4 - C_3 = 0$$

Para i=5,

$$\left(2 + \frac{\Delta x^2 k}{D}\right) C_5 - C_4 = C_d = 0$$

Logo, obtemos o sistema:

$$\begin{bmatrix} 2+(\Delta x^2 k)/D & -1 & 0 & 0 \\ -1 & 2+(\Delta x^2 k)/D & -1 & 0 \\ 0 & -1 & 2+(\Delta x^2 k)/D & -1 \\ 0 & 0 & -1 & 2+(\Delta x^2 k)/D \end{bmatrix} \begin{bmatrix} C2 \\ C3 \\ C4 \\ C5 \end{bmatrix} = \begin{bmatrix} Ce \\ 0 \\ 0 \\ Cd \end{bmatrix}$$

$$\vec{A} \quad \vec{x} = \vec{b}$$

Resolvendo as incógnitas do sistema temos que:

$$2 + \frac{\Delta x^2 k}{D} = 2 + \frac{5^2 \cdot 2 \cdot 10^{-6}}{1 \cdot 10^{-6}} = 52$$

Então, temos o sistema formado por:

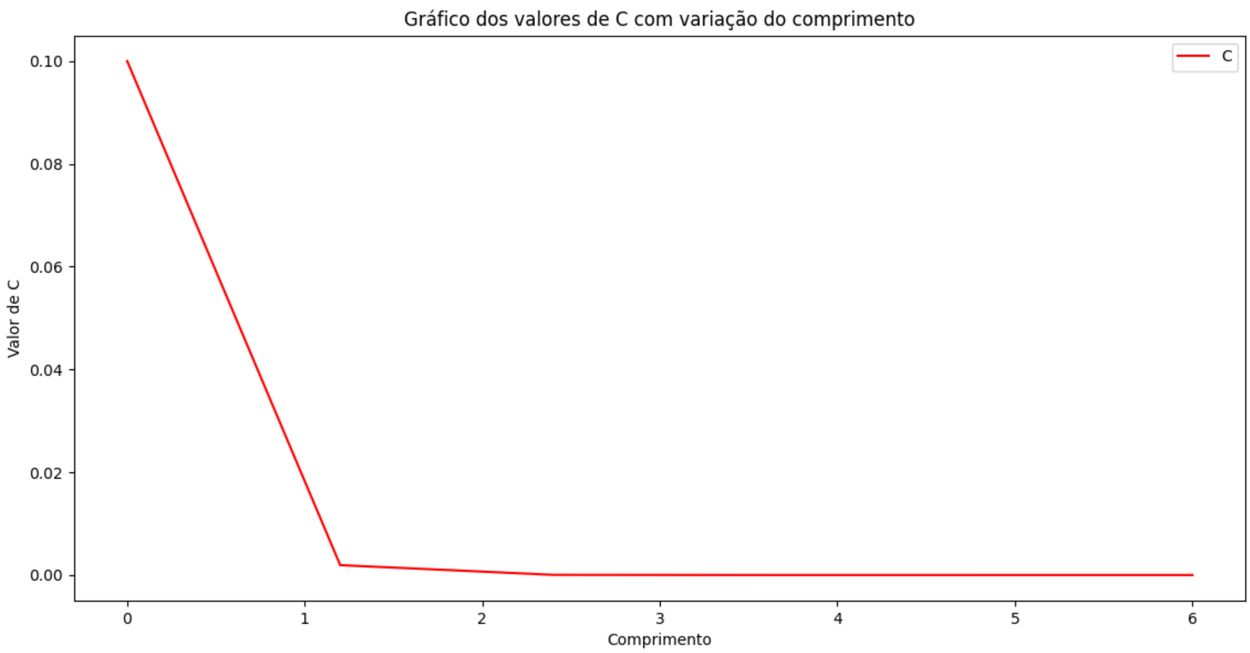
$$\begin{bmatrix} 52 & -1 & 0 & 0 \\ -1 & 52 & -1 & 0 \\ 0 & -1 & 52 & -1 \\ 0 & 0 & -1 & 52 \end{bmatrix} \begin{bmatrix} C2 \\ C3 \\ C4 \\ C5 \end{bmatrix} = \begin{bmatrix} 0{,}1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\vec{A} \quad \vec{x} = \vec{b}$$

Desta forma, a partir do programa criado, calculamos o sistema utilizando o método de Gauss Jacobi, e obtivemos o seguinte resultado:

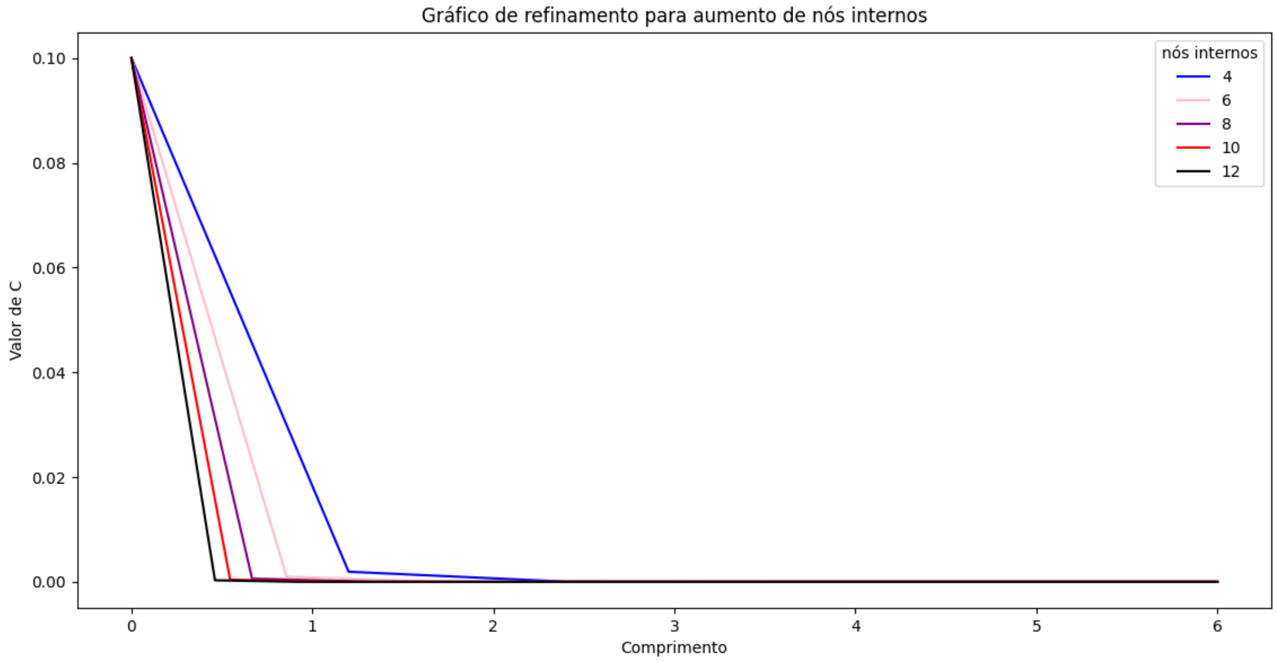
$$
\begin{aligned}
C_2 &= 0.001923788646684639 \\
C_3 &= 3.7009627570224883\text{e} - 05 \\
C_4 &= 7.119869168666427\text{e} - 07 \\
C_5 &= \ 1.3692056690114523\text{e} - 08
\end{aligned}
$$

Com o gráfico sendo:

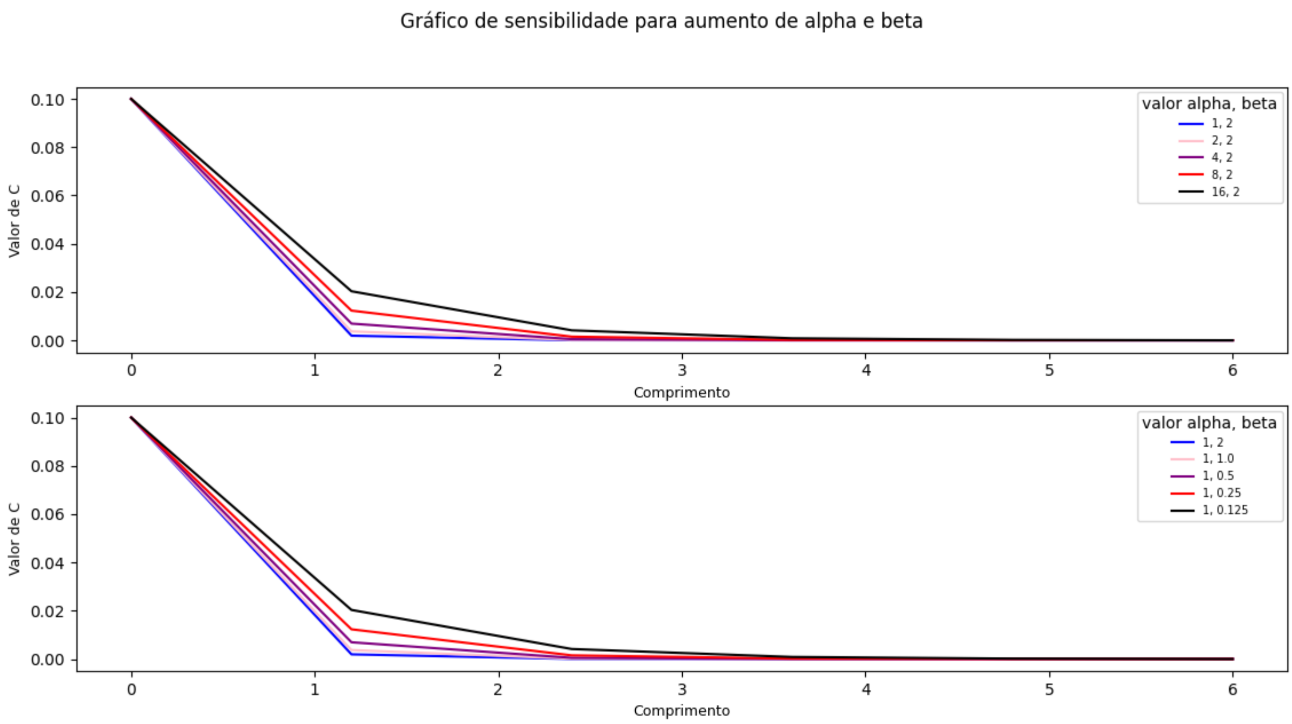


Dessa mesma forma, para realizar o estudo de refinamento de malha computacional utilizamos da técnica de aumentar o valor de nós internos somando no mesmo 2 unidades por 4 iterações, obtendo o vetor de valores $\boldsymbol{nós\ internos} =$ **[4, 6, 8, 10, 12]**. Já para o estudo de análise de sensibilidade, utilizamos técnica de aumentar o valor de α e manter o de β, multiplicando alfa por 2 por 4 iterações, obtendo o vetor de valores: $\boldsymbol{\alpha} =$ **[1, 2, 4, 8, 16],** e utilizamos técnica de diminuir o valor de β e manter o de α, dividindo beta por 2 por 4 iterações, obtendo o vetor de valores: $\boldsymbol{\beta} =$ **[2, 1, 0.5, 0.25, 0,125]**.

Seu refinamento de malha computacional é dado pelo gráfico:



Sua análise de sensibilidade é dada pelo gráfico:



E, por fim, abaixo tem-se o código da segunda parte do trabalho:

```
from pprint import pprint
from numpy import diag, diagflat, dot

alpha = 1
beta = 2 * alpha
Ce = 0.1
Cd = 0
L = 6
nos_internos = 4


def espaço_interno(nos_interior):
    return L / (nos_interior + 1)


def constante_pvc(alpha, beta, nos_interior):
    D = alpha * (10**-6)
    k = beta * (10**-6)
    return 2 + ((((nos_interior + 1) ** 2) * k) / D)


def gauss_jacobi(matriz_A, matriz_x, matriz_b):

    matriz_D = diag(matriz_A)
    matriz_R = matriz_A - diagflat(matriz_D)

    for i in range(10):
        matriz_x = (matriz_b - dot(matriz_R, matriz_x)) / matriz_D
    return matriz_x


def sistema_pvc(nos_interior, alpha_, beta_):

    linha_A = [0] * nos_interior
    matriz_A = [linha_A] * nos_interior
    matriz_b = []
    matriz_x = []
    a = -1
    b = 0

    for l in range(nos_interior):
        linha = []
        for c in range(nos_interior):
            if c == l:
                linha.append(constante_pvc(alpha_, beta_, nos_interior))
            elif (c == l - 1) | (c - 1 == l):
```

```
                linha.append(a)
            else:
                linha.append(b)
        matriz_A[l] = linha

    print("\nMatriz A:")
    pprint(matriz_A)

    for l in range(nos_interior):
        if l == 0:
            matriz_b.append(Ce)
        elif l == nos_internos - 1:
            matriz_b.append(Cd)
        else:
            matriz_b.append(0)

    print("\nMatriz b:")
    print(matriz_b)

    for l in range(nos_interior):
        matriz_x.append(1)

    print("\nMatriz x:")
    print(matriz_x)

    matriz_solucao = gauss_jacobi(matriz_A, matriz_x, matriz_b)
    print("\nMatriz solucao:")

    i = 2
    j = 0
    while i <= nos_interior + 1:
        print("C{} = {}".format(i, matriz_solucao[j]))
        i += 1
        j += 1

    vetor_solucao = [Ce]
    for i in range(nos_interior):
        vetor_solucao.append(matriz_solucao[i])
    vetor_solucao.append(Cd)

    return vetor_solucao


resultadoPVC = sistema_pvc(nos_internos, alpha, beta)

import matplotlib.pyplot as plt

vetor_comprimento = [0]
z = 0
```

```python
while z < nos_internos + 1:
    vetor_comprimento.append(vetor_comprimento[z] +
(espaço_interno(nos_internos)))
    z += 1


def grafico_PVC():

    plt.title("Gráfico dos valores de C com variação do comprimento")
    plt.plot(vetor_comprimento, resultadoPVC, color="red", label="C")
    plt.legend()
    plt.xlabel("Comprimento")
    plt.ylabel("Valor de C")
    plt.show()


grafico_PVC()


def refinamento():
    vetor_nos_internos = [nos_internos]

    for i in range(0, 4):
        vetor_nos_internos.append(vetor_nos_internos[i] + 2)

    for i in range(0, 5):
        vetor_comprimento = [0]
        z = 0
        while z < vetor_nos_internos[i] + 1:
            vetor_comprimento.append(
                vetor_comprimento[z] +
(espaço_interno(vetor_nos_internos[i]))
            )
            z += 1
        cor = ["blue", "pink", "purple", "red", "black"]
        print("\nResultados para nos internos =
{}".format(vetor_nos_internos[i]))
        resultados_PVC = sistema_pvc(vetor_nos_internos[i], alpha, beta)
        plt.title("Gráfico dos valores de C com variação do comprimento")
        plt.plot(
            vetor_comprimento, resultados_PVC, color=cor[i],
label=vetor_nos_internos[i]
        )
        plt.legend(title="nós internos")
        plt.xlabel("Comprimento")
        plt.ylabel("Valor de C")

    plt.title("Gráfico de refinamento para aumento de nós internos")
```

```
    plt.show()


refinamento()


def sensibilidade():

    vetor_alpha_normal = [alpha]
    vetor_alpha = [alpha]
    vetor_beta_normal = [beta]
    vetor_beta = [beta]

    for i in range(0, 4):
        vetor_alpha.append(vetor_alpha[i] * 2)
        vetor_alpha_normal.append(vetor_alpha_normal[i])
        vetor_beta.append(vetor_beta[i] / 2)
        vetor_beta_normal.append(vetor_beta_normal[i])

    fig, (grafico_alpha, grafico_beta) = plt.subplots(2, 1)
    for i in range(0, 5):

        cor = ["blue", "pink", "purple", "red", "black"]
        print(
            "\nResultados para alpha = {} e beta = {}".format(
                vetor_alpha[i], vetor_beta_normal[i]
            )
        )

        resultado_PVC_alpha = sistema_pvc(
            nos_internos, vetor_alpha[i], vetor_beta_normal[i]
        )

        legenda_alpha_normal = "{}, {}".format(vetor_alpha[i],
vetor_beta_normal[i])

        grafico_alpha.plot(
            vetor_comprimento,
            resultado_PVC_alpha,
            color=cor[i],
            label=legenda_alpha_normal,
        )
        grafico_alpha.legend(title="valor alpha, beta", fontsize=7)
        grafico_alpha.set_xlabel("Comprimento", fontsize=9)
        grafico_alpha.set_ylabel("Valor de C", fontsize=9)

        print(
            "\nResultados para alpha = {} e beta = {}".format(
                vetor_alpha_normal[i], vetor_beta[i]
```

```
            )
        )

        resultado_PVC_beta = sistema_pvc(
            nos_internos, vetor_alpha_normal[i], vetor_beta[i]
        )

        legenda_beta_normal = "{}, {}".format(vetor_alpha_normal[i],
vetor_beta[i])

        grafico_beta.plot(
            vetor_comprimento,
            resultado_PVC_beta,
            color=cor[i],
            label=legenda_beta_normal,
        )
        grafico_beta.legend(title="valor alpha, beta", fontsize=7)
        grafico_beta.set_xlabel("Comprimento", fontsize=9)
        grafico_beta.set_ylabel("Valor de C", fontsize=9)

    plt.suptitle("Gráfico de sensibilidade para aumento de alpha e beta")
    plt.show()


sensibilidade()
```