**Universidade Federal do Rio de Janeiro**

**Escola Politécnica**

               Modelagem Hidráulica

                            TP5

**Josiane de Sousa Bezerra - DRE: 110154535**

**Talita Fernandes Ferreira - DRE: 113040832**

**Questão 1**

**a)**

# Exemplo para um sistema de 5 equações e 5 incógnitas(xN):

#

# Sistema de equações:

# B(x1) + C(x2) + 0(x3) + 0(x4) + 0(x5) = d1

# A(x1) + B(x2) + C(x3) + 0(x4) + 0(x5) = d2

# 0(x1) + A(x2) + B(x3) + C(x4) + 0(x5) = d3

# 0(x1) + 0(x2) + A(x3) + B(x4) + C(x5) = d4

# 0(x1) + 0(x2) + 0(x3) + A(x4) + B(x5) = d5

#

# Sistema na forma matricial: A . X = D

# [ B. C. 0. 0. 0.] [x1] [d1]

# [ A. B. C. 0. 0.] [x2] [d2]

# [ 0. A. B. C. 0.] x [x3] = [d3]

# [ 0. 0. A. B. C.] [x4] [d4]

# [ 0. 0. 0. A. B.] [x5] [d5]

# importa as bibliotecas necessárias:

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**def** load\_abc\_vectors(tdm):

*'Retorna os vetores a, b, c da matriz tridiagonal'*

a = [] # inicial o vetor a

b = [] # inicial o vetor b

c = [] # inicial o vetor c

**for** coluna **in** range(0, len(tdm)): # carrega os vetores a, b, c percorrendo a matriz

linha = coluna # percore pela diagonal principal

b.append(float(tdm[linha, coluna])) # carrega o vetor b convertido para float

**if** coluna == 0: # primeiro elto

a.append(0.0) # primeiro elto do vetor a eh 0

a.append(float(tdm[linha + 1, coluna])) # pega o elto abaixo da diagonal principal em float

**if** coluna == (len(tdm) - 1): # último elto

c.append(float(tdm[linha - 1, coluna])) # pega o elto acima da diagonal principal em float

c.append(0.0) # último elto do vetor c é 0

**if** coluna != 0 **and** coluna != (len(tdm) - 1): # demais eltos

a.append(float(tdm[linha + 1, coluna])) # pega elto abaixo da diagonal principal em float

c.append(float(tdm[linha - 1, coluna])) # pega o elto acima da diagonal principal em float

**return** a, b, c

**def** load\_cl(a, b, c):

*'Retorna o vetor c linha a partir dos vetores a, b, c'*

cl = [0] \* (len(c) - 1) # inicializa o vetor c linha

**for** i **in** range(0, len(cl)): # percorre c linha

**if** i == 0: # primeiro elemento

cl[i] = c[i]/b[i] # calcula o primeiro c linha

**else**: # demais eltos

cl[i] = c[i]/(b[i] - (a[i] \* cl[i - 1])) # calcula os demais eltos

**return** cl

**def** load\_dl(a, b, cl, d):

*'Retorna o vetor d linha a partir dos vetores a, b, cl, d'*

dl[0] \* len(d) # inicializa o vetor d linha

**for** i **in** range(0, len(dl)): # percorre d linha

**if** i == 0: # primeiro elemento

dl[i] = float(d[i])/b[i] # calcula o primeiro d linha

**else**: # demais eltos

dl[i] = (float(d[i]) - (a[i] \* dl[i - 1]))/(b[i] - (a[i] \* cl[i - 1])) # calcula os demais eltos

**return** dl

**def** tdma\_solver(tdm, d):

*'Retorna o vetor x com as incognitas'*

x = [0] \* len(d) # inicialixa o vetor x

a, b, c = load\_abc\_vectors(tdm) # pega os valores de a, b, c, da matriz tdm de entrada

cl = load\_cl(a, b, c) # pega o vetor c linha calculado

dl = load\_dl(a, b, cl, d) # pega o vetor d linha calculado

**for** i **in** range(len(x) - 1, -1, -1): # percorre o vetor x de tras para frente

**if** i == len(x) - 1: # último elto

x[i] = dl[i] # carrega o último elto

**else**: # demais eltos

x[i] = dl[i] - (cl[i] \* x[i + 1]) # carrega demais eltos **return** x

**b)**

# Dados do trabalho:

tdm\_input = np.matrix(np.zeros((100, 100))) # inicializa a matriz tridiagonal de entrada

**for** coluna **in** range(0, len(tdm\_input)): # percorre a matriz linha = coluna # diagonal principal

tdm\_input[linha, coluna] = coluna + 1 + (coluna + 1) \* 0.001 # carrega diagonal principal

**if** coluna == 0: # primeiro elto

tdm\_input[linha + 1, coluna] = coluna + 2 + (coluna + 1) \* 0.001 # elto abaixo da diagonal principal

**if** coluna == (len(tdm\_input) - 1): # último elto tdm\_input[linha - 1, coluna] = coluna + 0 + (coluna + 1) \* 0.001 # elto acima da diagonal principal

**if** coluna != 0 **and** coluna != (len(tdm\_input) - 1): # demais eltos

tdm\_input[linha + 1, coluna] = coluna + 2 + (coluna + 1) \* 0.001 # carrega elto abaixo da diagonal principal

tdm\_input[linha - 1, coluna] = coluna + 0 + (coluna + 1) \* 0.001 # carrega elto acima da diagonal principal

vec\_d = np.matrix(np.arange(1, len(tdm\_input) + 1).reshape(len(tdm\_input), 1)) # carrega vetor d

vec\_x = tdma\_solver(tdm\_input, vec\_d) # calcula o vetor x através do algoritmo de Thomas

**c)**

BI\_COLOR = ['#4286F4', '#6BC924'] # define a cor das barras no gráfico

**def** label\_gen(x):

*'Retorna os labels do eixo X intercalados verticalmente'* labels = [] # inicialixa o vetor de labels **for** i **in** range(1, len(x) + 1): # percorre o vetor x

**if** i % 2 == 0: # se o indice do vetor x for par

labels.append('|\nX' + str(i)) # escreve o label mais em baixo

**else**: # se o índice do vetor x for ímpar

labels.append('X' + str(i)) # escreve o label na mesma linha

**return** labels

# Plot: plt.style.use('ggplot') # estilo do gráfico

n\_bars = len(vec\_x) # quantidade de barras

x\_loc = np.arange(n\_bars) # divide igualmente o eixo x pela quantidade de barras

bar\_width = len(x\_loc)/n\_bars # defino a largura das barras para q se toquem

fig, ax = plt.subplots(figsize=(20, 10), dpi=600) # crio uma figura e os eixos do grafico

bars\_rects = ax.bar(x\_loc, vec\_x, bar\_width, color=BI\_COLOR) # ploto o grafico

x\_labels = label\_gen(vec\_x) # gero os labels do eixo x

y\_min = min(vec\_x) \* 1.3 # ajusto o tamanho do eixo y

y\_max = max(vec\_x) \* 1.3 # ajusto o tamanho do eixo y

ax.set\_ylim((y\_min, y\_max)) # defino os limites do eixo y

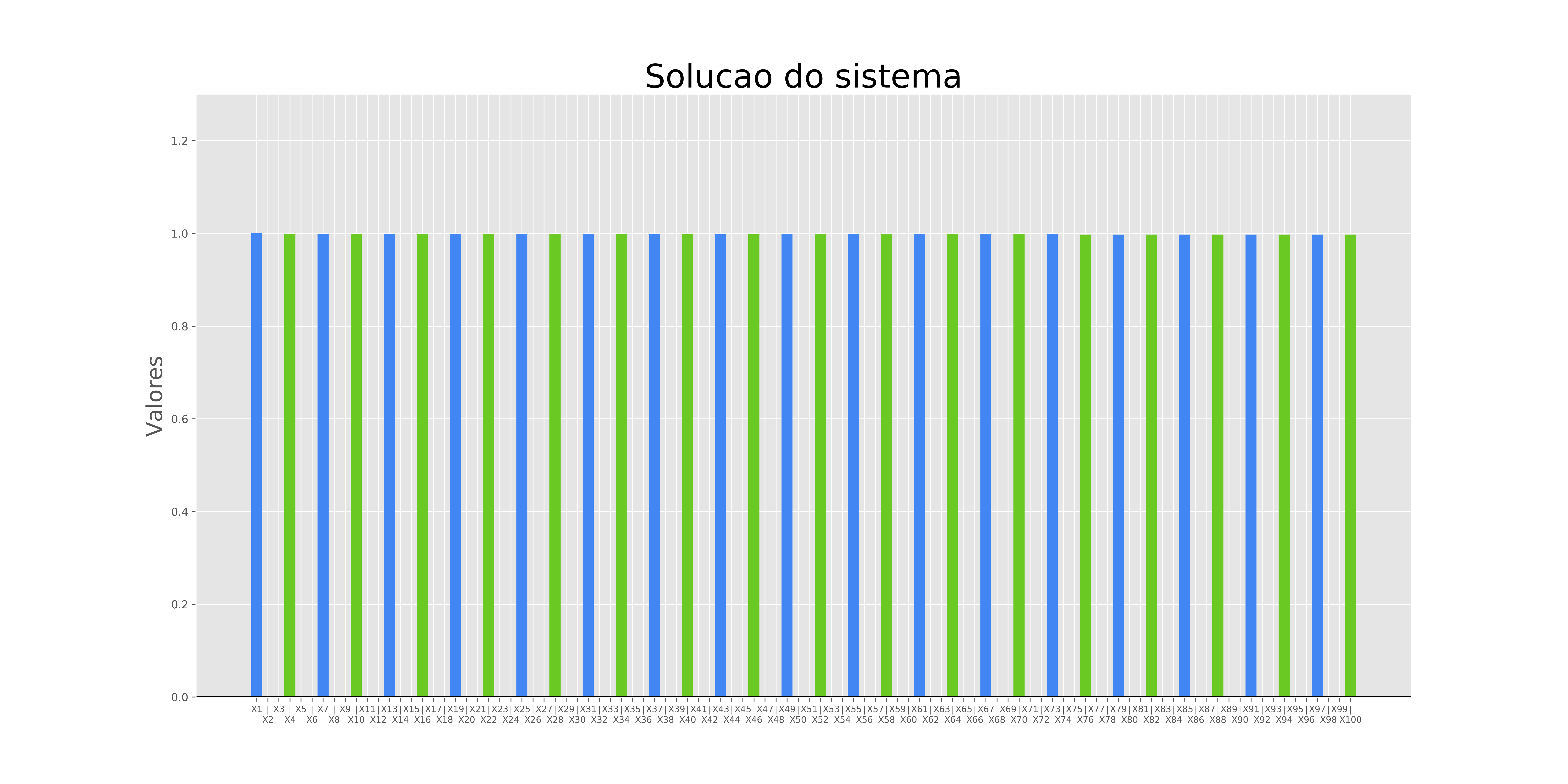
ax.set\_title('Solucao do sistema', fontsize=30) # título do gráfico

ax.set\_ylabel('Valores', fontsize=20) # título do eixo y ax.set\_xticks(x\_loc) # defino posicao que serão inseridos os labels do eixo x

ax.set\_xticklabels(x\_labels, fontsize=8) # defino quais sao os labels

plt.axhline(color='k') # ploto uma linha preta no

plt.savefig('vec\_x.png') # salvo o gráfico



**Questão 2**

Resolução dos problemas 7 e 10 do Capítulo 4 do livro “*Computational Fluid Dynamics*” (Abbott & Basco).

Problema 7

No Capítulo 4 foi abordada a questão da difusão, fenômeno que é regido pelo modelo matemático abaixo:

Onde:

**K** = coeficiente de difusão (constante)

**C**= concentração da substância que está sendo difundida.

É considerado um canal infinito de uma dimensão, com seção transversal A.

Uma fonte contínua de poluição é iniciada com uma vazão volumétrica Q (m³/s) e concentração C\* (mg/l).

Deseja-se calcular a concentração C(x,t).

1. **Esquema Explícito, Progressivo no Tempo e Centrado no Espaço.**

Discretização Progressiva no Tempo:

Discretização Centrada no Espaço:

Substituindo as duas discretizações no modelo matemático (I), temos:

Multiplicando toda a equação acima por Δt:

Sendo:

Então:

Explicitando , encontramos:

1. **Deve -se computar a concentração do soluto em seis passos no espaço e doze passos no tempo (j=6 e n=12). Selecionar o Δt que retornar a melhor acurácia.**

Dados:

A = 200 m²

Concentração do efluente lançado : C\* = 1000 mg/l

Δx = 250 m

K = 100 m²/s

Q = 1 m³/s

Velocidade do curso do canal : U = 0 m/s

j varia de 1 até 6

n varia de 0 até 12

Condição Inicial (t = 0 s)

C (x,0) = 0 mg/l

Condições de contorno

Ponto de diluição : j = 1

No limite : C = 0 para todo n

Na entrada:

Faremos a análise de estabilidade e de consistência do modelo numérico.

1. **Análise de Estabilidade**

Adotamos solução parcial padrão (harmônico genérico):

Substituímos no algoritmo (II):

Dividindo a equação acima por :

Utilizando a identidade:

Colocando em (III):

Sabendo que:

Substituindo essas relações em (IV):

O Fator de Amplificação A é:

Utilizando a equação (v):

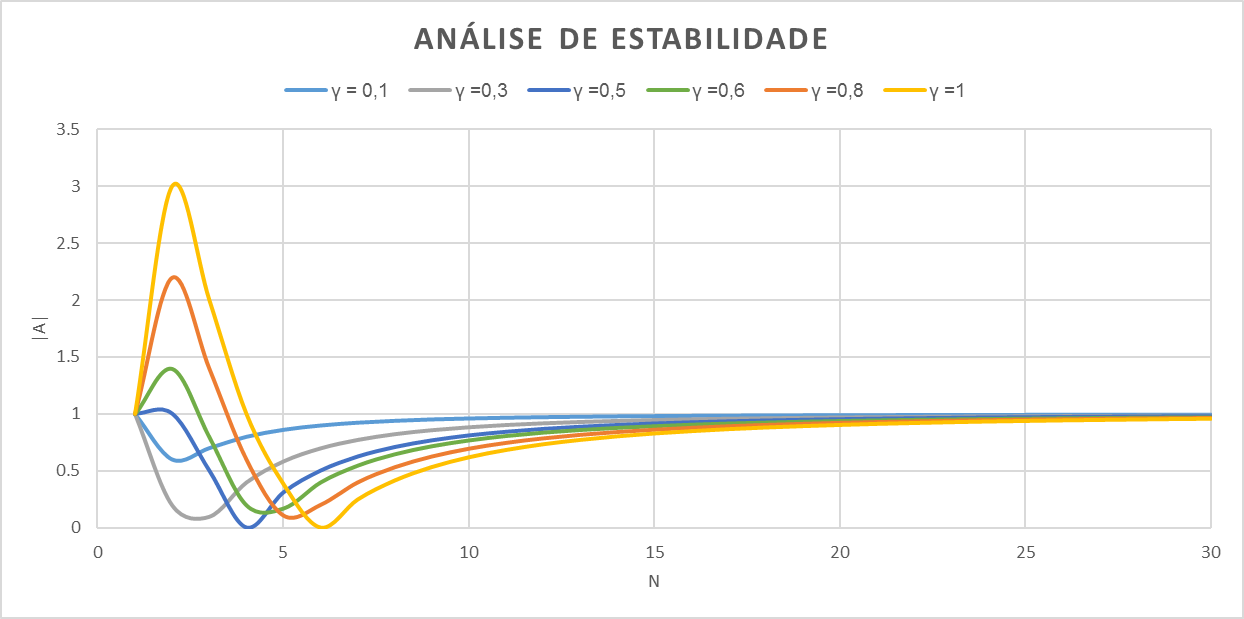
O critério para verificar a estabilidade é:

Como:

Sabendo ainda que:

Fazendo uma planilha na qual se varia N de 1 até 30, e também utilizando seis valores distintos de γ, pode-se calcular o módulo de A para cada α correspondente.

O gráfico abaixo mostra o resultado dessa análise. A condição de estabilidade |A|≤ 1 é verificada para γ ≤ ½ .



**2)Análise de Consistência – Erro de truncamento**

Utilizando a expansão por séries de Taylor a fim de verificar o erro de truncamento, encontramos:

Aplicando os termos acima nas discretizações:

Substituindo as séries no modelo matemático (i):

Os termos dentro dos colchetes são os erros de truncamento. Para que o esquema seja consistente, temos que anular tais termos:

É necessário descobrir qual valor de γ nos leva à condição acima. Rearranjando a equação acima e utilizando , temos:

Quando , só é preciso provar que:

Aplicando raiz quadrada dos dois lados, temos:

Após realizar os cálculos, chega-se a um erro de truncamento de segunda ordem. Esse resultado pode ser interpretado como uma boa acurácia para o problema.

Portanto, temos que . Substituindo esse valor, K = 100 m²/s e Δx = 250 m (dados do enunciado) na equação abaixo, encontramos Δt:

Calculando a concentração na entrada (j=1)

Aplicando as condições de contorno e inicial no esquema numérico (ii):

1. **Gráfico com os resultados obtidos**

Abaixo está representado o gráfico para o problema acima. A variação da concentração de poluente está plotada em relação ao tempo (segundos) para cada passo no espaço (j).

**Problema 10**

Pede-se na questão que seja desenvolvido um esquema implícito de diferença finita, dada a equação de onda abaixo, utilizando 3 pontos no eixo x em (n-1) e (n+1) e 3 pontos ao longo do eixo n, no ponto j.

Para resolver o problema proposto, foi utilizado um esquema centrado para o tempo e para o espaço, realizando a ponderação na parte implícita do esquema nos pontos *n+1* e *n-1.*

Sabendo que:

Discretizando :

Modificando a equação, tem-se:

Substituindo

**Questão 3**

Repetir o problema 7 com as seguintes modificações:

1. **Esquema implícito de Crank-Nicholson com θ = ½**

Esquema Implícito 🡪 se está no presente (n) olhando para o futuro (n+1). Os passos são maiores que em um esquema explícito, sendo assim um modelo mais veloz, porém também mais caro.

O modelo matemático é da difusão é:

O esquema numérico requerido é o implícito de Crank-Nicholson:

Discretizando o modelo, temos:

Multiplicando por Δt:

Substituindo

Colecionando termos em n+1 no lado esquerdo da equação:

Substituindo C da equação acima por φ e utilizando:

Temos então um sistema de equações representado abaixo:

1. **Usar a subrotina da questão 1 para computar o resultado obtido no item a**

Considerando , , e as condições de contorno e inicial da questão 7:

Condição Inicial (t = 0 s)

C (x,0) = 0 mg/l

Condições de contorno

Ponto de diluição 🡪 j = 1

No limite 🡪 C = 0 para todo n

Na entrada🡪

1. **Gráfico com os resultados obtidos**
2. **Comparar e comentar os resultados das duas versões do problema 7**

O modelo utilizado na questão 7 do livro é um esquema explícito, enquanto o utilizado nesta questão é um esquema implícito. Para comparar os dois esquemas, juntamos os resultados em um único gráfico:

É possível verificar, a partir dos resultados exibidos acima, que os valores obtidos para as concentrações ao longo do tempo para um esquema explícito e implícito são bastante próximos. Essa constatação é esperada uma vez que o mesmo fenômeno, com as mesmas condições de contorno, está sendo representado.

Os valores de concentração de poluente mostram uma tendência de uniformização ao longo do tempo dado que as concentrações em cada seção do espaço tendem a um valor constante.

O modelo implícito está centrado em n + ½ , o que faz que os resultados obtidos através dele estejam mais próximos da solução analítica. Dessa maneira, pode-se dizer que esse modelo é mais confiável que o modelo explícito.