Universidade de Brasília

Implementação de Controle com Redução Modal

Ataias Pereira Reis
Emanuel Pereira Barroso Neto

15 de abril de 2016

1 Introdução

O objetivo deste documento é apresentar os procedimentos necessários para implementar o método de controle apresentado no artigo "*Modal Reduction Based Tracking Control for Installation of Subsea Equipments*", desenvolvido por Fabrício et al, em um controlador industrial da Rockwell. Nem todos os detalhes estão presentes no artigo, o que torna difícil simplesmente lê-lo e realizar o sistema. Algumas modificações no controle serão feitas com base no trabalho do aluno de mestrado Rafael Simões <rafael.domenici@hotmail.com>.

2 Equações Governantes

Para o riser, a Equação 2.1 representa o deslocamento horizontal $\Upsilon(z,t)$ do tubo — um barbante, no caso da bancada de laboratório — sob a ação de forças hidrodinâmicas externas $F_n(z,t)$ (força linear com unidade N/m) e tração $T(z)$ (unidade N):

$$m_s \frac{\partial^2 \Upsilon}{\partial t^2} = -EJ\frac{\partial^4 \Upsilon}{\partial z^4} + \frac{\partial}{\partial z}\left(T(z)\frac{\partial \Upsilon}{\partial z}\right) + F_n(z,t) \tag{2.1}$$

Antes de prosseguir, é importante definir termos desta equação:

- m_s é a massa linear do barbante (densidade linear, kg/m)
- E é o módulo de Young do barbante e ele é desconhecido

- J é o segundo momento de área e representa a resistência do barbante à flexão. O barbante não apresenta tal resistência, daí $J = 0$
- $T(z)$ é a força de tração e é dada por

$$T(z) = (m_b + zm_s)\, g,$$

sendo m_b a massa da bolinha (kg), $m_s = m_{\text{barbante,kg}}/L$, sendo L o comprimento do barbante, z a posição vertical a partir do carrinho e g é a força da gravidade.

A força externa resultante, $F_n(z,t)$, é dada por

$$F_n(z,t) = -m_{fbar}\frac{\partial^2 \Upsilon}{\partial t^2} - \mu \left|\frac{\partial \Upsilon}{\partial t}\right| \frac{\partial \Upsilon}{\partial t}, \tag{2.2}$$

na qual μ é o coeficiente de arrasto (adimensional ? deveria ser, mas há algo que não bate) e m_{fbar} é a massa do fluido adicionado, que será posteriormente pormenorizada. Fazendo $m = m_s + m_{fbar}$ e substituindo a Equação 2.2 na 2.1, obtém-se:

$$\frac{\partial^2 \Upsilon}{\partial t^2} = -\frac{EJ}{m}\frac{\partial^4 \Upsilon}{\partial z^4} + \frac{\partial}{\partial z}\left(\frac{T(z)}{m}\frac{\partial \Upsilon}{\partial z}\right) - \frac{\mu}{m}\left|\frac{\partial \Upsilon}{\partial t}\right| \frac{\partial \Upsilon}{\partial t} \tag{2.3}$$

2.1 Constantes

Significado	**Símbolo**	**Valor**	**Unidade**
Massa	m_{bar}	0.492	g
Comprimento	L	0.82	m
Massa linear	m_s	0.6	g/m
Raio	r_{bar}	1	mm
Densidade	ρ_{bar}	191	kg/m^3

Tabela 2.1: Constantes do barbante

Significado	**Símbolo**	**Valor**	**Unidade**
Massa	m_b	0.492	g
Raio	r_b	15.3	mm
Coeficiente de inércia	C_m	1.2	-
Coeficiente de arrasto	C_d	0.6	-
Volume	V_b	$\frac{4}{3}\pi r_b^3$	m^3
Área da seção transversal	A_b	πr_b^2	m^2

Tabela 2.2: Constantes da bolinha de isopor

Como visto anteriormente, $m = m_s + m_{fbar}$. A massa linear m_{fbar} do fluido adicionado ao redor do barbante é dada por

$$\begin{aligned} m_{fbar} &= 2\pi r_{bar}^2 \rho_{\text{ar}} \\ &= 0.00770 \text{ g/m}. \end{aligned} \tag{2.4}$$

Já que $m_{fbar} \ll m_s$, consideraremos $m = m_s$ nos cálculos. Em relação à massa m_{fb} do fluido adicionado ao redor da bolinha de isopor, ela é dada por

$$\begin{aligned} m_{fb} &= 1.2 V_b \rho_{\text{ar}} \\ &= 1.2 \left(\frac{4}{3} \pi r_b^3 \right) \rho_{\text{ar}} \\ &= 0.0220 \text{ g} \end{aligned} \tag{2.5}$$

$m_{fb} \ll m_b$ também, de forma que os cálculos consideraram $m' = m_b$.

Nota-se que o barbante pesa mais que o isopor, o que faz com que a tração não seja principalmente devida pela bolinha, mas sim pelo barbante. Neste caso, não se utiliza um valor médio para $T(z)$ como no artigo do Fabrício, mas ainda se pode usar um valor médio para as constantes τ e τ' que substitui o termo $\frac{\mu}{m} \left| \frac{\partial \Upsilon}{\partial t} \right|$ para o barbante e para a bolinha, respectivamente. Já levando em conta um valor médio para $\left| \frac{\partial \Upsilon}{\partial t} \right|$, tem-se

$$\frac{\partial^2 \Upsilon}{\partial t^2} = -\frac{EJ}{m} \frac{\partial^4 \Upsilon}{\partial z^4} + \frac{\partial}{\partial z} \left(\frac{T(z)}{m} \frac{\partial \Upsilon}{\partial z} \right) - \tau \frac{\partial \Upsilon}{\partial t} \tag{2.6}$$

Antes de prosseguirmos para a discretização e obtenção das matrizes em espaço de estados, é importante pensar nas condições de contorno. No topo, $z = L$, tem-se $\Upsilon(L, t) = u(t)$, ou seja, o carrinho se move conforme uma trajetória $u(t)$ definida. Neste mesmo ponto, $\frac{\partial \Upsilon}{\partial z}(L, t) = 0$. Para a ponta na qual a carga está situada, $z = 0$, tem-se $\frac{\partial \Upsilon}{\partial z}(0, t) = \frac{F_L}{T}$, sendo F_L a força aplicada pela ponta do riser na carga. **(Outra coisa que confundi, eu entendi $u(t)$ sendo uma trajetória, pois Υ é deslocamento, mas no artigo do Fabrício está escrito em uma momento que é uma força)**.

2.2 Discretização

De forma a se realizar o controle proposto, o sistema deve ter um espaço de estados finito. Para isso, aplica-se o método de diferenças finitas na coordenada z de maneira a se aproximar a EDP governante em um número finito de EDOs. No espaço discreto, a equação do k-ésimo elemento é dada por

$$\begin{aligned} \frac{d^2 \Upsilon_k}{dt^2} &= -\frac{EJ}{ml^4} \left(\Upsilon_{k-2} - 4\Upsilon_{k-1} + 6\Upsilon_k - 4\Upsilon_{k+1} + \Upsilon_{k+2} \right) \\ &\quad + \frac{T_0 + mg(k-1)l}{ml^2} \left(\Upsilon_{k-1} - 2\Upsilon_k + \Upsilon_{k+1} \right) + g \frac{-\Upsilon_{k-1} + \Upsilon_{k+1}}{2l} - \tau \frac{d\Upsilon_k}{dt}, \end{aligned} \tag{2.7}$$

sendo N o número de pontos de discretização e l a distância entre dois pontos de discretização ($l = L/N$).

Note que $k \in \mathbb{N} : 2 \leq k \leq N-1$, pois um dos extremos é a bolinha e a equação do pêndulo rege seu movimento enquanto que a outra ponta se aplica uma condição de contorno. O que aconteceria quando $k = 2$ e se precisasse de Υ_{k-2}? Para nosso experimento, $J = 0$ e esse problema não ocorre. Caso se façam testes com um valor de $J \neq 0$, teríamos de resolver esse problema primeiro.

Para simplificar, definem-se as constantes

$$a = -\frac{EJ}{ml^4} \tag{2.8}$$

$$b_k = \frac{T_0 + mg(k-1)l}{ml^2},\ k \geq 2 \tag{2.9}$$

$$c = \frac{g}{2l} \tag{2.10}$$

$$d_k = b_k - c,\ k \geq 2 \tag{2.11}$$

$$e_k = b_k + c,\ k \geq 2 \tag{2.12}$$

A meu ver, a melhor forma de se analisar como as matrizes do sistema ficarão é expandir o sistema para casos com N pequeno e ver o que está ocorrendo. Observe que $a = 0$ para o barbante, o que simplifica os próximos passos.

Para o caso $N = 6$, tem-se

$$\mathbf{x} = \left(\Upsilon_1\ \Upsilon_2\ \Upsilon_3\ \Upsilon_4\ \Upsilon_5\ \Upsilon_6\ \dot{\Upsilon}_1\ \dot{\Upsilon}_2\ \dot{\Upsilon}_3\ \dot{\Upsilon}_4\ \dot{\Upsilon}_5\ \dot{\Upsilon}_6\right)^T \tag{2.13}$$

$$u = \Upsilon(L,t) = \Upsilon_7 \tag{2.14}$$

$$y = \Upsilon(0,t) = \Upsilon_1 \tag{2.15}$$

e as equações são

$$\begin{aligned}\ddot{\Upsilon}_2 &= b_2\left(\Upsilon_1 - 2\Upsilon_2 + \Upsilon_3\right) + c(-\Upsilon_1 + \Upsilon_3) - \tau\dot{\Upsilon}_2 \\ &= d_2\Upsilon_1 - 2b_2\Upsilon_2 + e_2\Upsilon_3 - \tau\dot{\Upsilon}_2\end{aligned} \tag{2.16}$$

$$\begin{aligned}\ddot{\Upsilon}_3 &= b_3\left(\Upsilon_2 - 2\Upsilon_3 + \Upsilon_4\right) + c(-\Upsilon_2 + \Upsilon_4) - \tau\dot{\Upsilon}_3 \\ &= d_3\Upsilon_2 - 2b_3\Upsilon_3 + e_3\Upsilon_4 - \tau\dot{\Upsilon}_3\end{aligned} \tag{2.17}$$

$$\begin{aligned}\ddot{\Upsilon}_4 &= b_4\left(\Upsilon_3 - 2\Upsilon_4 + \Upsilon_5\right) + c(-\Upsilon_3 + \Upsilon_5) - \tau\dot{\Upsilon}_4 \\ &= d_4\Upsilon_3 - 2b_4\Upsilon_4 + e_4\Upsilon_5 - \tau\dot{\Upsilon}_4\end{aligned} \tag{2.18}$$

$$\begin{aligned}\ddot{\Upsilon}_5 &= b_5\left(\Upsilon_4 - 2\Upsilon_5 + \Upsilon_6\right) + c(-\Upsilon_4 + \Upsilon_6) - \tau\dot{\Upsilon}_5 \\ &= d_5\Upsilon_4 - 2b_5\Upsilon_5 + e_5\Upsilon_6 - \tau\dot{\Upsilon}_5\end{aligned} \tag{2.19}$$

$$\begin{aligned}\ddot{\Upsilon}_6 &= b_6\left(\Upsilon_5 - 2\Upsilon_6 + u\right) + c(-\Upsilon_5 + u) - \tau\dot{\Upsilon}_6 \\ &= d_6\Upsilon_5 - 2b_6\Upsilon_6 + e_6u - \tau\dot{\Upsilon}_6\end{aligned} \tag{2.20}$$

A equação para a posição da carga Υ_1 leva em conta a massa da bolinha e a força de Morison:

$$m_b\ddot{\Upsilon}_1 = \frac{m_b g}{(N-1)l}\left(\Upsilon_2 - \Upsilon_1\right) + \rho_{\text{ar}} C_m V_b \ddot{\Upsilon}_1 - \frac{1}{2}\rho_{\text{ar}} C_d A_b \dot{\Upsilon}_1 \left|\dot{\Upsilon}_1\right|, \tag{2.21}$$

e, isolando-se $\ddot{\Upsilon}_1$, tem-se

$$\ddot{\Upsilon}_1 = \frac{m_b g}{m'(N-1)l}\left(\Upsilon_2 - \Upsilon_1\right) - \frac{1}{2m'}\rho C_d A_b \dot{\Upsilon}_1 \left|\dot{\Upsilon}_1\right|. \tag{2.22}$$

Note que $m' = m_b + \rho_{\text{ar}} C_m V_b = m_b + m_{fb} \approx m_b$. Assim, assume-se $m' = m_b$ para os cálculos.

Anteriormente, foi apresentada a linearização τ para o termo $\frac{1}{2m}\rho C_d A \left|\dot{\Upsilon}_k\right|$ do cabo. Assumo que isso também seja necessário para a bola, resultando em um τ':

$$\ddot{\Upsilon}_1 = b_1\left(-\Upsilon_1 + \Upsilon_2\right) - \tau'\dot{\Upsilon}_1, \tag{2.23}$$

com

$$b_1 = \frac{m_b g}{m'(N-1)l} = \frac{g}{(N-1)l}. \tag{2.24}$$

Desta forma, pode-se definir o sistema linear em forma matricial

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -b_1 & b_1 & 0 & 0 & 0 & 0 & -\tau' & 0 & 0 & 0 & 0 & 0 \\ d_2 & -2b_2 & e_2 & 0 & 0 & 0 & 0 & -\tau & 0 & 0 & 0 & 0 \\ 0 & d_3 & -2b_3 & e_3 & 0 & 0 & 0 & 0 & -\tau & 0 & 0 & 0 \\ 0 & 0 & d_4 & -2b_4 & e_4 & 0 & 0 & 0 & 0 & -\tau & 0 & 0 \\ 0 & 0 & 0 & d_5 & -2b_5 & e_5 & 0 & 0 & 0 & 0 & -\tau & 0 \\ 0 & 0 & 0 & 0 & d_6 & -2b_6 & 0 & 0 & 0 & 0 & 0 & -\tau \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ e_6 \end{bmatrix} u \tag{2.25}$$

que pode ser representado concisamente como

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{0}_{6\times6} & \mathbf{I}_{6\times6} \\ \mathbf{M}_{6\times6} & \mathbf{L}_{6\times6} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{0}_{11\times1} \\ e_6 \end{bmatrix} u \tag{2.26}$$

Para o caso de uma discretização com N pontos, tem-se

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{0}_{N\times N} & \mathbf{I}_{N\times N} \\ \mathbf{M}_{N\times N} & \mathbf{L}_{N\times N} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{0}_{2N-1\times1} \\ e_N \end{bmatrix} u \tag{2.27}$$

$$y = \begin{bmatrix} 1 & \mathbf{0}_{1\times2N-1} \end{bmatrix} \mathbf{x} \tag{2.28}$$

2.3 Código para calcular matrizes

Listing 1: Código para gerar matrizes A, B e C

```
# -*- coding: utf-8 -*-

from numpy import *
from numpy.linalg import *

#a is a numpy matrix
def print_matriz(a):
  (I,J) = a.shape # Determina as dimensões da matriz a ser impressa
  for i in range(0,I):
    st = ''
    for j in range(0,J):
      #Números de valor absoluto menor que um threshold são
         mostrados como 0 para facilitar visualização
      st += '{:12.2}'.format((a[i,j] if (abs(a[i,j]) > 1e-10) else
         0.0))
      if (j != J - 1):
        st += ' '
    print(st)
  print()

#Returns a string with the matriz being initialized
def initializeMatlabMatrix(M, matrixName):
  (I,J) = M.shape # Determina as dimensões da matriz a ser impressa
  st = matrixName + ' = ['
  for i in range(0,I):
    for j in range(0,J):
      #Números de valor absoluto menor que um threshold são
         mostrados como 0 para facilitar visualização
      st += str(M[i,j])
      if (j != J - 1):
        st += ' '
    if (i != I - 1):
      st += ';\n'
  st += '];\n'
  return st

def generateSimulationMfile(A, B, C, D, filename):
  with open(filename, 'w') as f:
    f.write(initializeMatlabMatrix(A, 'A'))
    f.write(initializeMatlabMatrix(B, 'B'))
    f.write(initializeMatlabMatrix(C, 'C'))
    f.write(initializeMatlabMatrix(D, 'D'))
    f.write('\nsys = ss(A, B, C, D);\n')
    f.write('opt = stepDataOptions;\n')
    f.write('opt.InputOffset = 0;\n')
    f.write('opt.StepAmplitude = 0.3;\n')
```

```
44      f.write("t = (0:0.01:50)';\n")
45      f.write('y = step(sys, t, opt);')
46
47  def generateA(n, b, d, e, tau, taul):
48    L = (-tau) * eye(n)
49    L[0][0] = (-taul)
50    M = zeros((n,n))
51    for k in range(0,n):
52      M[k][k] = -2*b[k] if k != 0 else -b[k]
53      if k != n-1:
54        M[k+1][k] = d[k+1]
55        M[k][k+1] = e[k] if k != 0 else b[k]
56    A = vstack((hstack((zeros((n,n)),eye(n))),hstack((M,L))))
57    return matrix(A)
58
59  def generateB(n, e):
60    B = zeros((2*n,1))
61    B[2*n-1,0] = e[n-1]
62    return matrix(B)
63
64  def generateC(n):
65    C = zeros((1,2*n))
66    C[0,0] = 1        # Como C é uma matriz linha, as duas dimensões
        são necessárias; senão, toda a matriz C valerá 1.
67    return matrix(C)
68
69  def getReducedSystem(A,B,C,n):
70    eig_A,T = eig(A)  # eig_A são os autovalores de A, e T é a matriz
        de autovetores
71    T = matrix(T)
72    #print('Autovalores de A')
73    #print_matriz(matrix(eig_A))
74
75    print('Matriz T')
76    print_matriz(T)
77    print()
78
79    #A Matriz T não é a que deve ser utilizada para a transformação de
        similaridade!
80    #Ela tem números complexos e isso é ruim
81    #Ela deve ser convertida em uma matriz que tenha só números reais
82
83    Tnew = matrix(zeros((2*n,2*n))) #real! não é complexo
84    i = 0
85    while i < 2*n: #será que tem algo errado? tem de testar
86      if abs(imag(eig_A[i])) > 1e-10: # Procuramos algum elemento
          complexo de cada autovetor representado em T, não da matriz
          de autovalores de A
87        Tnew[:,i] = real(T[:,i])
```

```
88        Tnew[:,i+1] = -imag(T[:,i])
89        i = i + 2
90      else:
91        Tnew[:,i] = T[:,i]
92        i = i + 1
93
94    print('Matriz T nova')
95    print_matriz(Tnew)
96    print()
97
98    del T
99    T = Tnew
100
101   T_inv = matrix(inv(T))
102   A_M = T_inv * A * T
103   B_M = T_inv * B
104   C_M = C * T
105   C_M_diag = matrix(zeros((2*n,2*n), complex))
106
107
108   print('Matriz A_M')
109   print_matriz(A_M)
110   print()
111   #
112   # print('Matrix B_M')
113   # print_matriz(B_M)
114   # print()
115   #
116   # print('Matrix C_M transposta')
117   # print_matriz(C_M.transpose())
118   # print()
119
120   for i in range(0,2*n):
121     C_M_diag[i,i] = C_M[0,i]
122   A_M_inv = inv(A_M)
123   Gains = C_M_diag * A_M_inv * B_M
124   # print('Ganhos:')
125   # print_matriz(Gains)
126   # print()
127
128   gdim = (Gains.shape[0] // 2)
129   GainSum = zeros((gdim,1))
130
131   #Como todos os autovalores são complexos, cada subsistema 2x2 é
          composto
132   #de um autovalor e seu conjugado que é um outro autovalor
133   for i in range(0,Gains.shape[0],2):
134     GainSum[i//2] = real(abs(Gains[i] + Gains[i+1]))
135
```

```
136  # print('Ganhos dos subsistemas 2x2 (todos os autovalores são
         complexos):')
137  # print_matriz(GainSum)
138  # print()
139
140  #Obter maiores ganhos e índices
141  gain = array([max(GainSum)])
142  lines = array([argmax(GainSum)*2-2, argmax(GainSum)*2-1])
143  GainSum = delete(GainSum, argmax(GainSum))
144  beginning = 0
145  gain = insert(gain, beginning, max(GainSum))
146  lines = insert(lines, beginning, array([argmax(GainSum)*2-2,
         argmax(GainSum)*2-1]))
147  print('Maiores ganhos')
148  print(gain)
149  print('Índices')
150  print(lines)
151  print()
152
153  print('Autovalores')
154  print_matriz(matrix(eig_A[lines]))
155  print()
156
157  print('Matriz A_R')
158  eig1 = eig_A[lines[0]]
159  eig2 = eig_A[lines[2]]
160  A_R = matrix(array([[real(eig1), imag(eig1),  0,
                          0],
161            [-imag(eig1), real(eig1), 0,                    0],
162            [0,           0,           real(eig2),  imag(eig2)],
163            [0,           0,          -imag(eig2),  real(eig2)]]))
164  print_matriz(A_R)
165  print()
166
167  print('Polos de A_R')
168  print_matriz(matrix(eigvals(A_R)))
169  print()
170
171  print('Vetor B_R')
172  B_R = B_M[lines]
173  print_matriz(B_R)
174  print()
175
176  print('Vetor C_R')
177  C_R = C_M[0,lines]
178  print_matriz(C_R)
179  print()
180
181  print('Constante D_R')
```

```
182   D_R = C*inv(A)*B - C_R*inv(A_R)*B_R
183   print_matriz(D_R)
184   print()
185
186   generateSimulationMfile(A_R, B_R, C_R, D_R, 'simulacao.m')
187
188 def main():
189   n = 2
190   tau = 0.2426  # tau do barbante (0.2426)
191   taul = 0.1133 # tau da bolinha (0.1133)
192   ms = 0.0006   # massa linear do barbante (0.0006 kg/m)
193   mb = 0.00015  # massa da bolinha (0.00015 kg)
194   g = 9.80665   # aceleração da gravidade (9.807 m/s^2)
195   L = 0.82    # Comprimento total do barbante (0.82 m)
196   l = L/n     # distância entre dois pontos de discretização
197   T0 = mb*g   # Tração no ponto 0 (logo acima da bolinha) -
          considerando peso da bolinha (N)
198
199   b = zeros((n,1))
200   c = g/(2*l)
201   d = zeros((n,1))
202   e = zeros((n,1))
203   b[0] = g/((n-1)*l)
204   for k in range(2,n+1):
205     b[k-1] = (T0 + ms*g*(k-1)*l)/(ms*l**2)
206     d[k-1] = b[k-1] - c
207     e[k-1] = b[k-1] + c
208
209   # b = [1,2,3,4,5,6]
210   # d = [0,0.2,0.3,0.4,0.5,0.6] # Teste
211   # e = [0,12,13,14,15,16]
212
213   A = generateA(n, b, d, e, tau, taul)
214   # print('Matriz A:')
215   # print_matriz(A)
216   # print()
217   B = generateB(n,e)
218   # print('Matriz B:')
219   # print_matriz(B)
220   # print()
221   C = generateC(n)
222   # print('Matriz C:')
223   # print_matriz(C)
224   # print()
225   getReducedSystem(A,B,C,n)
226
227 def manuscript_p48():
228   n = 2
229   M = zeros((n,n))
```

```
  M[0,0] = -1
  M[0,1] = 1
  M[1,0] = 1
  M[1,1] = -3
  L = -2 * eye(n)
  A = vstack((hstack((zeros((n,n)),eye(n))),hstack((M,L))))
  B = zeros((2*n,1))
  B[2*n-1,0] = 2
  C = generateC(n)
  getReducedSystem(A, B, C, n)

# if __name__ == "__main__":
#   main()
```

3 Uma Estratégia de Redução da Ordem do Modelo

A maior parte da teoria clássica de controle lida com sistemas representados por um pequeno número de variáveis de estado. Portanto, uma forma de aplicar métodos clássicos de controle da literatura para sistemas de parâmetros distribuídos discretos é por meio de uma redução da ordem do modelo.

Tal redução do modelo será feita em duas etapas: primeiro, uma transformação modal é aplicada nas equações originais do espaço de estados, resultando em uma nova representação em variáveis modais. Nesta forma, o sistema pode ser visto como um conjunto de subsistemas dissociados em paralelo, cuja influência na saída pode ser calculada individualmente. Então, os subsistemas com os maiores ganhos estáticos são escolhidos para criar um modelo de ordem reduzida.

3.1 Decomposição Modal

Primeiro, deve-se obter os autovalores do espaço de estados do *riser*. Observa-se que eles são sempre distintos entre si, uma condição suficiente para a diagonalização da matriz do espaço de estados. Assim, calcula-se a matriz modal $\mathbf{T}$, cuja i-ésima coluna é o i-ésimo autovetor do sistema:

$$\mathbf{T} = (\ \mathbf{v_1}\ |\ \mathbf{v_2}\ |\ \ldots\ |\ \mathbf{v_{2N}}\ )_{1\times 2N} \tag{3.1}$$

Como a matriz $\mathbf{T}$ provavelmente tem valores complexos devidos a autovalores complexos. Isso é um problema para a representação em espaço de estados e sua simulação. A solução é criar uma matriz $\tilde{\mathbf{T}}$ que tenha só números reais. Antes de explicar como criá-la, lembre que os autovalores complexos sempre aparecem em pares conjugados já que a matriz $\mathbf{A}$ só tem valores reais. Quando a primeira coluna de um autovetor de um par complexo conjugado for encontrada, a coluna respectiva de $\tilde{\mathbf{T}}$ será sua parte real. A segunda coluna desse par complexo conjugado será a parte imaginária da coluna de $\mathbf{T}$.

A matriz $\tilde{\mathbf{T}}$ é utilizada para uma transformação de similaridade no sistema original:

$$\mathbf{A_M} = \tilde{\mathbf{T}}^{-1}\mathbf{A}\tilde{\mathbf{T}}, \tag{3.2}$$

$$\mathbf{x_M} = \tilde{\mathbf{T}}^{-1}\mathbf{x}, \tag{3.3}$$

$$\mathbf{B_M} = \tilde{\mathbf{T}}^{-1}\mathbf{B}, \text{ e} \tag{3.4}$$

$$\mathbf{C_M} = \mathbf{C}\tilde{\mathbf{T}}. \tag{3.5}$$

O sistema transformado, denotado pelo subscrito $\mathbf{M}$, é mais adequado à análise. $\mathbf{A_M}$ é uma matriz diagonal, com seus autovalores explícitos, e permitindo o desacoplamento do sistema original em N subsistemas de segunda ordem formados por pares de autovalores reais ou complexo-conjugados.

3.2 Redução Modal

Neste estágio, procura-se determinar quais dos subsistemas são mais adequados para aproximar o modelo original por meio do cálculo do ganho estático de cada um. Este método depende da predominância de uns poucos autovalores na resposta do sistema, já que altas frequências são muito atenuadas pelas forças hidrodinâmicas e pela suavidade da entrada.

Os subsistemas selecionados são combinados em um modelo reduzido

$$\dot{\mathbf{z}} = \mathbf{A_R}\mathbf{z} + \mathbf{B_R}u \tag{3.6}$$

$$y = \mathbf{C_R}\mathbf{z} + \mathbf{D_R}u \tag{3.7}$$

cuja ordem é escolhida considerando o custo-benefício entre a acurácia da dinâmica reduzida e a simplicidade da estrutura de controle exigida. Além disso, o sistema reduzido deve compensar o ganho estático perdido nos autovalores desconsiderados. Isto é feito por meio de uma matriz de transferência direta $\mathbf{D_R}$, que é a diferença dos ganhos dos sistemas original e reduzido:

$$\mathbf{D_R} = \mathbf{C}\mathbf{A^{-1}}\mathbf{B} - \mathbf{C_R}\mathbf{A_R^{-1}}\mathbf{B_R} \tag{3.8}$$

O subsistemas são de ordem 1 ou 2 dependendo se o autovalor é real ou um par complexo conjugado.

A matriz de transferência direta $\mathbf{D_R}$ introduz novas dinâmicas: uma saída não-nula que não leva em conta o atraso de propagação da entrada e um ganho em altas frequências. Conforme mostrado por Fortaleza (2009), podemos refinar o modelo reduzido introduzindo um atraso de entrada ϵ que minimiza a transferência direta e garante dinâmica nula para $t < \epsilon$:

$$\begin{aligned} \dot{\mathbf{z}} &= \mathbf{A_R}\mathbf{z} + \mathbf{B_D}u(t-\epsilon) \\ y &= \mathbf{C_R}\mathbf{z} + \mathbf{D_D}u(t-\epsilon) \end{aligned} \tag{3.9}$$

sendo

$$\mathbf{B_D} = \mathbf{A_M}\left(e^{\epsilon \mathbf{A_M}}\right)\mathbf{A_M^{-1}}\mathbf{B_M} \tag{3.10}$$

$$\mathbf{D_D} = \mathbf{C_M}\left(e^{\epsilon \mathbf{A_M}} - \mathbf{I}\right)\mathbf{A_M^{-1}}\mathbf{B_M} + \mathbf{D_M} \tag{3.11}$$

O novo modelo reduzido (3.9) é tal que, para uma entrada degrau no instante t', a saída mantém seu valor inicial enquanto $t < t' + \epsilon$. Para $t \geq t' + \epsilon$, ambos os modelos reduzidos produzem a mesma saída. O atraso ϵ pode ser visto como uma aproximação para o atraso natural de propagação da estrutura.