

---

# Implementação de Controle com Redução Modal

---

Ataias Pereira Reis  
Emanuel Pereira Barroso Neto

2 de junho 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>](mailto:Rafael.Simões@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):

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

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

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

na qual  $\mu$  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:

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

## 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	$\rho_{bar}$	191	kg/m <sup>3</sup>

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 <sup>3</sup>
Área da seção transversal	$A_b$	$\pi r_b^2$	m <sup>2</sup>

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

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

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

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

$m_{fb} \ll m_b$  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  $\tau$  e  $\tau'$ , que substituem o termo  $\frac{\mu}{m} \left| \frac{\partial \Upsilon}{\partial t} \right|$  para o barbante e para a bolinha, respectivamente. Essas constantes são definidas de acordo com a trajetória prevista, uma vez que a velocidade média depende dessa trajetória. Já levando em conta um valor médio para  $\left| \frac{\partial \Upsilon}{\partial t} \right|$ , tem-se

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

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  $\Upsilon$  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

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

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  $\Upsilon_{k-2}$ ? 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

$$a = -\frac{EJ}{ml^4} \quad (2.8)$$

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

$$c = \frac{g}{2l} \quad (2.10)$$

$$d_k = b_k - c, \quad k \geq 2 \quad (2.11)$$

$$e_k = b_k + c, \quad k \geq 2 \quad (2.12)$$

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

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

$$u = \Upsilon(L, t) = \Upsilon_7 \quad (2.14)$$

$$y = \Upsilon(0, t) = \Upsilon_1 \quad (2.15)$$

e as equações são

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

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

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

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

$$\begin{aligned}\ddot{\Upsilon}_6 &= b_6 (\Upsilon_5 - 2\Upsilon_6 + u) + c(-\Upsilon_5 + u) - \tau \dot{\Upsilon}_6 \\ &= d_6 \Upsilon_5 - 2b_6 \Upsilon_6 + e_6 u - \tau \dot{\Upsilon}_6\end{aligned}\quad (2.20)$$

A equação para a posição da carga  $\Upsilon_1$  leva em conta a massa da bolinha e a força de Morison:

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

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

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

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  $\tau$  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  $\tau'$ :

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

com

$$b_1 = \frac{m_b g}{m' l} = \frac{g}{l}. \quad (2.24)$$

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

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

que pode ser representado concisamente como

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{0}_{6 \times 6} & \mathbf{I}_{6 \times 6} \\ \mathbf{M}_{6 \times 6} & \mathbf{L}_{6 \times 6} \end{bmatrix} \mathbf{x} + \begin{bmatrix} \mathbf{0}_{11 \times 1} \\ e_6 \end{bmatrix} u \quad (2.26)$$

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

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

$$y = \begin{bmatrix} 1 & \mathbf{0}_{1 \times 2N-1} \end{bmatrix} \mathbf{x} \quad (2.28)$$

### 2.3 CÓDIGO PARA CALCULAR MATRIZES

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

```

1 module ModalReduction
2 export generateA, generateB, generateC
3 export generateABC, getABC_M, getABCD_R
4 export manuscript_p48, simulation
5 export generateMATLABSimulationScript
6
7
8 function simulation(n,original=false)
9     A, B, C = generateABC(n)
10    A_M, B_M, C_M = getABC_M(n, A, B, C)
11    A_R, B_R, C_R, D_R = getABCD_R(n, A_M, B_M, C_M)
12    if original
13        generateMATLABSimulationScriptToCompare("simulacaoN" * string(n)
14            * "Compare.m", A, B, C, A_R, B_R, C_R, D_R)
15        generateMATLABSimulationScript(n, "simulacaoN" * string(n) *
16            "Reduced.m", A_R, B_R, C_R, D_R)

```

```

15     else
16         generateMATLABSimulationScript(n, "simulacaoN" * string(n) *
            "Reduced.m", A_R, B_R, C_R, D_R)
17     end
18 end
19
20 #Gera A, B, C to sistema completo
21 function generateABC(n)
22     tau = 0.2426      # tau do barbante (1/s) para excursão de 30cm
23     tau_l = 0.1133    # tau da bolinha (1/s) para excursão de 30cm
24     ms = 0.0006       # massa linear do barbante (kg/m)
25     mb = 0.00015     # massa da bolinha (kg)
26     g = 9.80665      # aceleração da gravidade (m/s^2)
27     L = 0.82         # Comprimento total do barbante (m)
28     l = L/n          # distância entre dois pontos de discretização (m)
29     T0 = mb*g        # Tração no ponto 0 (logo acima da bolinha) -
        considerando peso da bolinha (N)
30
31     b = zeros(n)
32     c = g/(2*l)
33     d = zeros(n)
34     e = zeros(n)
35
36     b[1] = g/l
37     for k = 2:n
38         b[k] = (T0 + ms*g*(k-1)*l)/(ms*l^2)
39         d[k] = b[k] - c
40         e[k] = b[k] + c
41     end
42
43     A = generateA(n, b, d, e, tau, tau_l)
44     B = generateB(n,e[n])
45     C = generateC(n)
46
47     return A, B, C
48 end
49
50 function generateA(n, b, d, e, tau, tau_l)
51     M = zeros(n,n)
52     #Primeira linha de M
53     M[1,1] = -b[1]
54     M[1,2] = b[1]
55
56     #Linhas 2 ate n-1
57     for i = 2:n-1
58         M[i,i-1] = d[i]
59         M[i,i] = -2*b[i]
60         M[i,i+1] = e[i]
61     end

```

```

62
63 #Linha n
64 M[n,n-1] = d[n]
65 M[n,n] = -2*b[n]
66
67 L = eye(n)
68 for i = 1:n
69     L[i,i] = i == 1 ? -tau1 : -tau
70 end
71
72 #Concatenar matrizes, gerando matriz (2n,2n)
73 A = [[zeros(n,n) eye(n)]; [M L]]
74
75 return A
76 end
77
78 function generateB(n, eN)
79     B = zeros(2*n)
80     B[2*n] = eN
81
82     return B
83 end
84
85 function generateC(n)
86     C = zeros(1,2*n)
87     C[1,1] = 1
88
89     return C
90 end
91
92 function getT(n, A)
93     eig_A = eigvals(A)
94     Tcomplex = eigvecs(A)
95
96     T = zeros(2*n, 2*n)
97     i = 1
98     while i <= 2*n #será que tem algo errado? tem de testar
99         if abs(imag(eig_A[i])) > 1e-10
100             T[:,i] = real(Tcomplex[:,i])
101             T[:,i+1] = -imag(Tcomplex[:,i])
102             i = i + 2
103         else
104             T[:,i] = Tcomplex[:,i]
105             i = i + 1
106         end
107     end
108
109     return T
110 end

```



```

111
112 function getABC_M(n, A, B, C)
113     T = getT(n,A)
114
115     A_M = T \ A * T
116     B_M = T \ B
117     C_M = C * T
118
119     return A_M, B_M, C_M
120 end
121
122 function getABCD_R(n, A_M, B_M, C_M)
123     C_M_diag = diagm(vec(C_M)) #matriz diagonal
124     G = C_M_diag / A_M * B_M #ganhos
125     subsystems = getSubsystems(n, eigvals(A_M), G)
126     n_out = 4
127     A_R = zeros(n_out,n_out)
128     B_R = zeros(n_out)
129     C_R = zeros(1,n_out)
130
131     #Construir matrizes
132     i = 1
133     j = 1
134     while i <= n_out
135         index = subsystems[j][2]
136         if length(index) == 2 #complexo
137             A_R[i:i+1,i:i+1] = A_M[index, index]
138             B_R[[i,i+1]] = B_M[index]
139             C_R[1,[i,i+1]] = C_M[index]
140             i = i + 2
141         else
142             A_R[i,i] = A_M[index[1],index[1]]
143             B_R[i] = B_M[index[1]]
144             C_R[1,i] = C_M[index[1]]
145             i = i + 1
146         end
147         j = j + 1
148     end
149
150     D_R = C_M / A_M * B_M - C_R / A_R * B_R
151     return A_R, B_R, C_R, D_R
152 end
153
154 function getSubsystems(n, eig_A, G)
155     i = 1
156     subsystems = []
157     while i <= 2*n
158         if abs(imag(eig_A[i])) > 1e-10 #complexo

```

```

159     gain = abs(G[i] + G[i+1]) #considera sinal na soma como aqui
    ou soma os módulos?
160     append!(subsystems, [(gain, [i,i+1])])
161     i = i + 2
162     else #real
163         gain = abs(G[i])
164         append!(subsystems, [(gain, [i])])
165         i = i + 1
166     end
167 end
168 #ordena pelo primeiro elemento da tupla
169 #ordem decendente
170 sort!(subsystems, rev=true)
171 return subsystems
172 end
173
174 function manuscript_p48()
175     n = 2
176     M = [[-1 1]; [1 -3]]
177     L = -2 * eye(n)
178     A = [[zeros(n,n) eye(n)]; [M L]]
179     B = generateB(2,2)
180     C = generateC(n)
181
182     A_M, B_M, C_M = getABC_M(n,A,B,C)
183     A_R, B_R, C_R, D_R = getABCD_R(n, A_M, B_M, C_M)
184
185     return A_R, B_R, C_R, D_R
186 end
187
188 function generateMATLABSimulationScript(n, filename, A, B, C, D)
189     output = "A = " * string(A) * ";\n\n"
190     output = output * "B = " * string(B) * "';\n\n"
191     output = output * "C = " * string(C) * ";\n\n"
192     output = output * "D = " * string(D) * ";\n\n"
193     output = output * "sys = ss(A, B, C, D);\n"
194     output = output * "opt = stepDataOptions;"
195     output = output * "opt.InputOffset = 0;\n"
196     output = output * "opt.StepAmplitude = 0.3;\n"
197     output = output * "t = (0:0.01:50)';\n"
198     output = output * "y = step(sys, t, opt);"
199
200     output = output * "fig = figure;\n"
201     output = output * "hold on;\n"
202     output = output * "plot(t,y,'k-');\n"
203     output = output * "xlabel('Time (s)'), ylabel('Position (m)');\n"
204     output = output * "title('Sistema original N = " * string(n) * ",
    simulacao reduzida para ordem 4');\n"

```

```

205 output = output * "print('SimulationReduceN" * string(n) * "',
    '-dpng', '-r300');\n"
206 output = output * "close(fig);\n"
207
208 file = open(filename, "w")
209 write(file, output)
210 close(file)
211 end
212
213 function generateMATLABSimulationScriptToCompare(filename, A, B, C,
    A_R, B_R, C_R, D_R)
214 output = "A_R = " * string(A_R) * ";\n\n"
215 output = output * "B_R = " * string(B_R) * "';\n\n"
216 output = output * "C_R = " * string(C_R) * ";\n\n"
217 output = output * "D_R = " * string(D_R) * ";\n\n"
218 output = output * "sysR = ss(A_R, B_R, C_R, D_R);\n\n"
219
220 output = output * "A = " * string(A) * ";\n\n"
221 output = output * "B = " * string(B) * "';\n\n"
222 output = output * "C = " * string(C) * ";\n\n"
223 output = output * "sys0 = ss(A, B, C, [0]);\n\n"
224
225 output = output * "opt = stepDataOptions;"
226 output = output * "opt.InputOffset = 0;\n"
227 output = output * "opt.StepAmplitude = 0.3;\n\n"
228
229 output = output * "t = (0:0.01:50)';\n"
230 output = output * "yR = step(sysR, t, opt);\n"
231 output = output * "y0 = step(sys0, t, opt);\n\n"
232
233 output = output * "fig = figure;\n"
234 output = output * "hold on;\n"
235 output = output * "plot(t,yR,'k--');\n"
236 output = output * "plot(t,y0,'k');\n"
237 output = output * "legend('Reduzido', 'Original');\n"
238 output = output * "xlabel('Time (s)'), ylabel('Position (m)');\n"
239 n = round(Int,size(A)[1]/2)
240 output = output * "title('Simulacao com sistema original N = " *
    string(n) * " e reduzido N = 4');\n"
241 output = output * "print('SimulationN" * string(n) * "', '-dpng',
    '-r300');\n"
242 output = output * "close(fig);\n"
243
244 file = open(filename, "w")
245 write(file, output)
246 close(file)
247 end
248
249

```

### 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:

$$\mathbf{T} = (\mathbf{v}_1 \mid \mathbf{v}_2 \mid \dots \mid \mathbf{v}_{2N})_{1 \times 2N} \quad (3.1)$$

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:

$$\mathbf{A}_M = \tilde{\mathbf{T}}^{-1} \mathbf{A} \tilde{\mathbf{T}}, \quad (3.2)$$

$$\mathbf{x}_M = \tilde{\mathbf{T}}^{-1} \mathbf{x}, \quad (3.3)$$

$$\mathbf{B}_M = \tilde{\mathbf{T}}^{-1} \mathbf{B}, \text{ e} \quad (3.4)$$

$$\mathbf{C}_M = \mathbf{C} \tilde{\mathbf{T}}. \quad (3.5)$$

O sistema transformado, denotado pelo subscrito  $\mathbf{M}$ , é mais adequado à análise.  $\mathbf{A}_{\mathbf{M}}$  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

$$\dot{\mathbf{z}} = \mathbf{A}_{\mathbf{R}}\mathbf{z} + \mathbf{B}_{\mathbf{R}}u \quad (3.6)$$

$$y = \mathbf{C}_{\mathbf{R}}\mathbf{z} + \mathbf{D}_{\mathbf{R}}u \quad (3.7)$$

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  $\mathbf{D}_{\mathbf{R}}$ , que é a diferença dos ganhos dos sistemas original e reduzido:

$$\mathbf{D}_{\mathbf{R}} = \mathbf{C}\mathbf{A}^{-1}\mathbf{B} - \mathbf{C}_{\mathbf{R}}\mathbf{A}_{\mathbf{R}}^{-1}\mathbf{B}_{\mathbf{R}} \quad (3.8)$$

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

A matriz de transferência direta  $\mathbf{D}_{\mathbf{R}}$  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  $\epsilon$  que minimiza a transferência direta e garante dinâmica nula para  $t < \epsilon$ :

$$\begin{aligned} \dot{\mathbf{z}} &= \mathbf{A}_{\mathbf{R}}\mathbf{z} + \mathbf{B}_{\mathbf{D}}u(t - \epsilon) \\ y &= \mathbf{C}_{\mathbf{R}}\mathbf{z} + \mathbf{D}_{\mathbf{D}}u(t - \epsilon) \end{aligned} \quad (3.9)$$

sendo

$$\mathbf{B}_{\mathbf{D}} = \mathbf{A}_{\mathbf{M}}(e^{\epsilon\mathbf{A}_{\mathbf{M}}})\mathbf{A}_{\mathbf{M}}^{-1}\mathbf{B}_{\mathbf{M}} \quad (3.10)$$

$$\mathbf{D}_{\mathbf{D}} = \mathbf{C}_{\mathbf{M}}(e^{\epsilon\mathbf{A}_{\mathbf{M}}} - \mathbf{I})\mathbf{A}_{\mathbf{M}}^{-1}\mathbf{B}_{\mathbf{M}} + \mathbf{D}_{\mathbf{M}} \quad (3.11)$$

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  $\epsilon$  pode ser visto como uma aproximação para o atraso natural de propagação da estrutura.