ALGORITMOS PARA MATRIZES POLINOMIAIS

Arlei Fonseca Barcelos

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Aprovada por:

_____
Prof. João Carlos dos Santos Basilio, D. Phil.

_____
Prof. Afonso Celso del Nero Gomes, D.Sc.

_____
Prof. Pedro Magalhães Guimarães Ferreira, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

ABRIL DE 2005

BARCELOS, ARLEI FONSECA

Algoritmos para matrizes polinomiais [Rio de Janeiro] 2005

XI, 112 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia Elétrica, 2005)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Sistemas multivariáveis

2. Matrizes polinomiais

3. Matriz de Sylvester

4. Decomposição por valores singulares

I. COPPE/UFRJ II. Título (série)

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus por manter-me firme no meu objetivo, mantendo sempre acesa a chama da esperança, mesmo com todos os obstáculos que surgiram.

Gostaria especialmente de agradecer a minha esposa e meu filho, Laura e Gabriel, por terem compreendido a minha ausência em alguns momentos da realização deste trabalho, como também pelo apoio nos momento difíceis. Aos meus Pais, Fernando e Dalila, e meus irmãos que tiveram papel fundamental na minha formação. E sogro e sogra, Paulo e Gildete, pelo apoio e compreensão.

Gostaria de agradecer aos meus atuais gerentes na CSN pelo apoio dado, Cláudio Avelino, Vilela e Kaneko, como também aos meus antigos gerentes que me ajudaram no início do mestrado, Lemos e Abdon. Aos amigos de trabalho que supriram a minha falta enquanto estava em aula, em especial, Raffaele, Benedito, Nakao e Paulo Cesar.

Aos professores da AEDB (Associação Educacional Dom Bosco), Mario Esteves, Pinto Bravo e Onofre, que me deram a oportunidade de lecionar, aumentando a vontade adquirir cada vez mais conhecimento para repassá-los aos meus alunos.

Por fim, gostaria de manifestar a minha gratidão aos professores da COPPE-UFRJ, pela forma que me acolheram, embora estivesse há 10 anos afastado da instituição. Em especial ao professor e orientador João Carlos dos Santos Basilio, que além de importante apoio técnico, foi um amigo, facilitando o desenvolvimento deste trabalho.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ALGORITMOS PARA MATRIZES POLINOMIAIS

Arlei Fonseca Barcelos

Abril/2005

Orientador: João Carlos dos Santos Basilio

Programa: Engenharia Elétrica

Matrizes polinomiais têm grande importância tanto em matemática quanto na análise e projeto de sistemas de controle multivariáveis. Contudo, apesar de atraente do ponto de vista teórico, existem poucas ferramentas para manipulação de matrizes polinomiais.

Dentro desse contexto o objetivo principal desse trabalho é coletar a partir da literatura, algoritmos robustos para operação com matrizes polinomiais e em seguida implementá-los em ambiente Matlab, levando à criação de uma "toolbox". É importante ressaltar que, embora implementados em ambiente Matlab os algoritmos são descritos de forma que possam ser implementados em qualquer linguagem de programação.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ALGORITHMS FOR POLYNOMIAL MATRICES

Arlei Fonseca Barcelos

April/2005

Advisor: João Carlos dos Santos Basilio

Department: Electrical Engineering

Polynomial matrices play an important role in both mathematics and in the analysis and design of multivariable control system. Although attractive from the theoretical point of view, the number of available tools for manipulation of polynomial matrices is quite small.

Within this content, the main objetive of this work is to collect, from the open literature, robust algorithms for operations with polynomial matrices and, in the sequel, to write Matlab functions, ending up with a polynomial matrix toolbox. It is important to stress that, although implemented in a Matlab environment, the algorithms are described such that they can be used to write routines in any programming language.

Conteúdo

Lista de Abreviaturas e Símbolos

DVS	Decomposição de Valores Singulares
DFM	Descrição por Frações de Matrizes Polinomiais
I_n	Matriz identidade de dimensão n
MDC	Máximo Divisor Comum
$\mathbb{R}^{p\times m}(s)$	Conjunto das matrizes racionais $p \times m$
$\mathbb{R}^{p\times m}[s]$	Conjunto das matrizes polinomiais $p \times m$
$\|.\|$	Determinante de uma matriz
$gr[.]$	Grau
$\rho[.]$	Posto normal de uma matriz polinomial
$\widehat{N}^T$	Transposta da matriz de coeficientes N
N^*	Conjugado complexo da matriz N
$>>$	Sinal de solicitação do Matlab

Capítulo 1

Introdução

Matrizes polinomiais têm grande importância tanto em matemática (Gohberg, Lancaster e Rodman 1982, Gantmacher 1959) quanto no estudo de sistemas de controle multivariável (Kailath 1980). Isto se deve ao fato que muitos dos conceitos para o caso monovariável só podem ser generalizados para o caso multivariável utilizando-se matrizes polinomiais, mais precisamente a chamada Descrição por Frações de Matrizes polinomiais (DFM), que podem ser definidas como:

$$H(s) = N(s)M^{-1}(s) = \tilde{M}^{-1}(s)\tilde{N}(s),$$

onde $H(s) \in \mathbb{R}^{p\times m}(s)$, $N(s), \tilde{N}(s) \in \mathbb{R}^{p\times m}[s]$, $M(s) \in \mathbb{R}^{m\times m}[s]$ e $\tilde{M}(s) \in \mathbb{R}^{p\times p}[s]$ com $\mathbb{R}^{p\times m}(s)$ denotando o conjunto das matrizes racionais $p\times m$ e $\mathbb{R}^{p\times m}[s]$ denotando o conjunto das matrizes polinomiais $p\times m$. O termo descrição por frações de matrizes pode ser também utilizado para denotar o produto de uma matriz racional pela inversa de outra matriz, também, racional. Neste trabalho serão consideradas apenas descrições por frações de matrizes polinomiais.

Apesar de atraente do ponto de vista teórico, existem poucas ferramentas para manipulação de matrizes polinomiais. Dentro desse contexto o objetivo principal deste trabalho é coletar algoritmos disponíveis na literatura com o requisito de que sejam robustos e, em seguida, implementá-los de forma que seja criada uma toolbox para o Matlab. Deve ser enfatizado que os algoritmos são descritos de forma que possam ser implementados em qualquer software de programação e não apenas no Matlab.

Atualmente, o Matlab apresenta ferramentas para muitos dos algoritmos mos-

trados aqui. Contudo, algumas vezes eles não apresentam a robustez desejada por se basearem em cálculos com variáveis simbólicas ou por utilizarem fórmulas recursivas que são mais eficientes para matrizes de baixa ordem. A vantagem dos algoritmos apresentados nesta tese é que eles têm como base a utilização de matrizes de convolução ou Sylvester e decomposição por valores singulares.

Além das ferramentas do Matlab que utilizam linguagem simbólica, há, também, no mercado uma "toolbox" desenvolvida para o Matlab encontrada em Polyx (2004).Contudo este pacote é pago, criando uma dificuldade para os usuários do meio acadêmico com baixo poder aquisitivo. Uma das premissas de pacote de ferramentas aqui desenvolvido é que será disponibilizado gratuitamente na internet para a comunidade científica.

A estrutura desta tese será a seguinte. No capítulo 2 serão discutidos os fundamentos téoricos que embasaram este tese, quais sejam:

1. Matrizes de coeficientes das matrizes polinomiais;
2. Matriz de convolução ou Sylvester;
3. Estudo da robustez da decomposição por valores singulares(DVS).

Ainda no capítulo 2, será feita uma revisão de alguns tópicos que são importantes para a manipulação de matrizes polinomiais, isto é:

1. Matrizes Unimodulares;
2. Redução por coluna;
3. Matrizes Coprimas;
4. Identidade de Bezout;
5. Bases Polinomiais Mínimas;
6. Determinação da Função de transferência a partir de uma realização de espaço de estados;

7. Inversão de Matrizes Polinomiais.

No capítulo 3 serão apresentados alguns algoritmos que foram desenvolvidos e que servirão de base para os algoritmos principais a serem discutidos no capítulo 4. O capítulo 3 é dividido em três partes:

1. Algoritmos Auxiliares. São os algoritmos que fazem manipulação de matrizes polinomiais, no sentido de mudar a sua forma de visualização ou retirar alguma informação pertinente da matriz, sem alterar as suas características. São eles.

   (a) Corta excessos de zeros á esquerda de uma matriz de coeficientes de uma matriz polinomial sem alterar o grau desta matriz;

   (b) Empilha as colunas de uma matriz polinomial formando uma matriz cujas linhas são os coeficientes dos polinômios de cada coluna;

   (c) Obtém as matrizes dos coeficientes de uma matriz polinomial a partir de um vetor cujas linhas são os coeficientes dos polinômios de cada coluna da matriz;

   (d) Calcula o grau máximo de cada coluna de uma matriz polinomial;

   (e) Forma uma matriz polinomial diagonal a partir de uma matriz cujas linhas são os coeficientes dos polinômios;

   (f) Calcula o valor de uma matriz polinomial $P(s)$ para $s = s_0 \in \mathbb{C}$;

   (g) Dadas duas matrizes $M(s)$ e $N(s)$ de mesmo número de linhas calcula as matrizes de coeficientes de $P(s) = [M(s) \quad N(s)]$;

   (h) Cria uma matriz de convolução de determinada ordem a partir das matrizes de coeficientes de uma dada matriz polinomial;

   (i) Extrai os elementos da diagonal de uma matriz polinomial.

2. Operações Básica com Matrizes polinomiais. Serão considerados os algoritmos que executam as seguintes operações básicas com as matrizes polinomiais:

   (a) Soma;

(b) Multiplicação;

3. Operações com uma matriz polinomial. Serão descritos os algoritmos que fazem operações em apenas uma matriz, quais sejam:

   (a) Transposta;

   (b) Conjugado complexo;

   (c) Traço;

No capítulo 4, serão considerados os algoritmos robustos, sendo divididos em:

1. Algoritmos Fundamentais: são os algoritmos que possuem certa complexidade no seu desenvolvimento e são fundamentais para os algoritmos principais. São eles:

   (a) Redução por colunas;

   (b) Conversão de uma Realização em Espaço de Estados em uma DFM.;

   (c) Bases polinomiais mínimas;

   (d) Posto normal de uma matriz polinomial.

2. Algoritmos principais.Serão mostrados os algoritmos que foram criados para utilização prática em sistemas de controle.

   (a) Inversa;

   (b) Determinante;

   (c) Descrição por frações de matrizes coprimas;

   (d) Máximo divisor comum de matrizes polinomiais;

   (e) Identidade de Bezout generalizada.

Finalmente, no capítulo 5 será feita uma análise dos algoritmos apresentados e uma descrição dos trabalhos futuros.

Nesta tese será adotada a utilização de notação tipográfica (máquina de escrever antiga) para comandos do Matlab, por exemplo, a função "matsplit.m" desenvolvida em Matlab quando for referida no texto o será como `Matsplit`.

Capítulo 2

Fundamentos Teóricos

Neste capítulo serão revisados os conceitos que servirão para o desenvolvimento dos algoritmos usados para manipulação das matrizes polinomiais. A estrutura desse capítulo é a seguinte: na seção 2.1 será definida a matriz formada pelas matrizes de coeficientes de matrizes polinomiais, por esta ser a forma de entrada e saída das funções Matlab a serem desenvolvidas; Na seção 2.2 será feita uma exposição de como construir as matrizes de convolução (ou de Sylvester) simples e generalizada; o conceito de matriz unimodular é relembrado na seção 2.3, enfatizando a sua importância nas operações com matrizes polinomiais, principalmente no que se refere à criação de matrizes reduzidas por coluna; na seção 2.4 será considerado o estudo de matrizes coprimas, e como verificar a coprimicidade de duas matrizes polinomiais através da Identidade de Bezout; na seção 2.7 é considerada a obtenção de realizações em espaço de estados de ordem mínima, a partir da descrição por frações de matrizes irredutíveis, isto é, aquelas cujas matrizes do numerador e denominador são coprimas; a decomposição por valores singulares(DVS) é apresentada teoricamente na seção 2.10 devida à sua importância na robustez dos algoritmos propostos e principalmente na obtenção de bases polinomiais mínimas, será discutida na seção 2.8; finalmente, considera-se a forma de Smith na seção 2.9, embora, como será visto, neste trabalho não serão apresentados algoritmos para obtenção da forma de Smith, tendo em vista que os algoritmos disponíveis na literatura, tem como base, em geral, operações de pivoteamento (Kucera 1979, Barnett 1983).

2.1 Matriz coeficiente de uma matriz polinomial

Nos algoritmos a serem considerados nesta tese, a entrada e a saída das matrizes polinomiais serão feitas através das matrizes dos coeficientes.

Definição 2.1 *Define-se matriz coeficiente de uma matriz polinomial* $p \times m$*, isto é,* $G(s) \in \mathbb{R}^{p\times m}[s]$*, onde:*

$$G(s) = G_\nu s^\nu + G_{\nu-1}s^{(\nu-1)} + \cdots + G_0, G_i \in \mathbb{R}^{p\times m}, i = 0, \ldots, \nu. \tag{2.1}$$

a matriz é obtida concatenando-se G_i *lado a lado , isto é:*

$$G = \left[\begin{array}{cccc} G_\nu & G_{\nu-1} & \cdots & G_0 \end{array} \right].$$

□

Note que $G \in \mathbb{R}^{p\times m(\nu+1)}$. Um outro ponto a ser ressaltado é que, no decorrer deste trabalho, será utilizada a seguinte notação: (i) $G(s)$ para matriz polinomial; (ii) G para matriz coeficiente associada à matriz polinomial $G(s)$; (iii) G_i para a matriz coeficiente de $G(s)$ associada à potência de s de grau i e (iv) g_{ij} elemento (i, j) da matriz G_i.

Para se ter uma idéia da vantagem de se utilizar a entrada de dados através de matrizes coeficientes, considere a soma de duas matrizes polinomiais. Para isso suponha que se tenham duas matrizes, $A(s)$ e $B(s) \in \mathbb{R}^{p\times m}(s)$, ambas de grau n, isto é:

$$A(s) = \sum_{k=0}^{n} A_{n-k}s^{n-k} \text{ e } B(s) = \sum_{k=0}^{n} B_{n-k}s^{n-k}, \tag{2.2}$$

onde $A_{n-k}, B_{n-k} \in \mathbb{R}^{p\times\nu m}$. Tem-se,portanto:

$$G(s) = A(s) + B(s) = \sum_{k=0}^{n}(A_{n-k} + B_{n-k})s^{n-k}, \tag{2.3}$$

o que ilustra a facilidade criada com a utilização da matriz de coeficientes, isto é, $G = A + B$.

2.2 Matriz de convolução ou de Sylvester

Matrizes de convolução são formadas utilizando-se as matrizes de coeficientes (G_i) das matrizes polinomiais dada na equação (2.1), e surgem naturalmente na multiplicação de duas matrizes polinomiais. Para ilustrar esse ponto, sejam $A(s) \in \mathbb{R}^{p\times n}[s]$ e $B(s) \in \mathbb{R}^{n\times m}[s]$ de graus ϕ e α, respectivamente, isto é:

$$A(s) = A_\phi s^\phi + A_{\phi-1}s^{(\phi-1)} + \cdots + A_0, A_i \in \mathbb{R}^{p\times n}, i = 0, 1, ..., \phi \tag{2.4}$$

e

$$B(s) = B_\alpha s^\alpha + B_{\alpha-1}s^{(\alpha-1)} + \cdots + B_0, B_i \in \mathbb{R}^{n\times m}, i = 0, 1, ..., \alpha, \tag{2.5}$$

e suponha que se deseje obter

$$G(s) = A(s)B(s).$$

Portanto

$$G(s) = A(s)B(s) = G_{\phi+\alpha}s^{\phi+\alpha} + G_{\phi+\alpha-1}s^{\phi+\alpha-1} + \ldots + G_0,$$

onde as matrizes de coeficientes G_i podem ser obtidas a partir da seguinte multiplicação matricial:

$$\begin{pmatrix} G_{\phi+\alpha} \\ G_{\phi+\alpha-1} \\ \vdots \\ G_0 \end{pmatrix} = \overbrace{\begin{pmatrix} A_\phi & 0 & \cdots & 0 \\ A_{\phi-1} & A_\phi & \cdots & 0 \\ \vdots & \vdots & \ddots & \cdots \\ A_0 & A_1 & \ddots & A_\phi \\ 0 & A_0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & A_1 \\ 0 & 0 & \cdots & A_0 \end{pmatrix}}^{\alpha+1 \text{ blocos colunas}} \begin{pmatrix} B_\alpha \\ B_{\alpha-1} \\ \vdots \\ B_0 \end{pmatrix} = \begin{pmatrix} A_\phi.B_\alpha \\ A_{\phi-1}.B_\alpha + A_\phi.B_{\alpha-1} \\ \vdots \\ A_0.B_0 \end{pmatrix} \tag{2.6}$$

A matriz

$$C_A^{(\alpha)} = \overbrace{\begin{pmatrix} A_\phi & 0 & \cdots & 0 \\ A_{\phi-1} & A_\phi & \cdots & 0 \\ \vdots & \vdots & \ddots & \cdots \\ A_0 & A_1 & \ddots & A_\phi \\ 0 & A_0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & A_1 \\ 0 & 0 & \cdots & A_0 \end{pmatrix}}^{\alpha+1}$$

é conhecida como matriz de convolução associada à matriz $A(s)$ de ordem α .

2.2.1 Matrizes de Sylvester Generalizadas

Muitas vezes (Chen 1999, Kaylath, Bitmead, Kung e Anderson 1978, Anderson e Jury n.d.) há a necessidade de se resolver a equação

$$D(s)A(s) + N(s)B(s) = H(s) \tag{2.7}$$

onde $A(s), B(s), D(s), N(s)$ e $H(s)$ têm dimensões compatíveis, $D(s), N(s)$ e $H(s)$ são conhecidos e $A(s)$ e $B(s)$ devem ser determinados. Denotando:

$$D(s) = D_\phi s^\phi + D_{\phi-1}s^{\phi-1} + D_{\phi-2}s^{\phi-2} + ... + D_0, \tag{2.8}$$

$$N(s) = N_\phi s^\phi + N_{\phi-1}s^{\phi-1} + N_{\phi-2}s^{\phi-2} + ... + N_0, \tag{2.9}$$

$$A(s) = A_\alpha s^\alpha + A_{\alpha-1}s^{\alpha-1} + A_{\alpha-2}s^{\alpha-2} + ... + A_0, \tag{2.10}$$

$$B(s) = B_\alpha s^\alpha + B_{\alpha-1}s^{\alpha-1} + B_{\alpha-2}s^{\alpha-2} + ... + B_0, \tag{2.11}$$

$$H(s) = H_{\phi+\alpha}s^{\phi+\alpha} + ... + H_2 s^2 + H_1 s + H_0, \tag{2.12}$$

e observando que a equação (2.7) pode ser escrita como

$$H(s) = \begin{bmatrix} D(s) & N(s) \end{bmatrix} \begin{bmatrix} A(s) \\ B(s) \end{bmatrix}, \tag{2.13}$$

então uma das formas de se resolver a equação (2.13) é através da matriz de Sylvester Generalizada, onde os coeficientes matriciais das equações (2.8) a (2.12) são colocados na seguinte forma:

$$S(D, N).\chi(A, B) = H \tag{2.14}$$

onde

$$S(D,N) = \begin{pmatrix} D_\phi & N_\phi & \ddots & 0 & 0 \\ D_{\phi-1} & N_{\phi-1} & \ddots & 0 & 0 \\ D_{\phi-2} & N_{\phi-2} & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & 0 & 0 \\ D_1 & N_1 & \ddots & 0 & 0 \\ D_0 & N_0 & \ddots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ddots & D_\phi & N_\phi \\ 0 & 0 & \ddots & D_{\phi-1} & N_{\phi-1} \\ 0 & 0 & \ddots & D_{\phi-2} & N_{\phi-2} \\ 0 & 0 & \ddots & \vdots & \vdots \\ 0 & 0 & \ddots & D_1 & N_1 \\ 0 & 0 & \ddots & D_0 & N_0 \end{pmatrix} \tag{2.15}$$

A matriz H que é o resultado da igualdade na equação (2.14) é formada empilhando-se as matrizes de coeficientes da matriz $H(s)$, isto é,

$$H = \begin{bmatrix} H_{\phi+\alpha} \\ H_{\phi+\alpha-1} \\ \vdots \\ H_0 \end{bmatrix}$$

e a matriz χ é formada empilhando-se as matrizes de coeficientes das matrizes $A(s)$ e $B(s)$ da seguinte maneira:

$$\chi = \left.\begin{bmatrix} A_\alpha \\ B_\alpha \\ A_{\alpha-1} \\ B_{\alpha-1} \\ \vdots \\ A_1 \\ B_1 \\ A_0 \\ B_0 \end{bmatrix}\right\} 2(\alpha+1)\text{blocos de linhas.}$$

À matriz $S(D,N)$, definida na equação (2.15), dá-se o nome de matriz de Sylvester Generalizada associada às matrizes $D(s)$ e $N(s)$.

2.3 Matrizes Unimodulares

Muitas vezes deseja-se realizar operações com as linhas ou colunas de matrizes polinomiais com o objetivo, por exemplo, de calcular o posto normal da matriz ou redução por coluna. Essas operações envolvem a pré-multiplicação ou pós-multiplicação por matrizes que não alteram o posto normal da matriz considerada. Isto leva à seguinte definição.

Definição 2.2 *Uma matriz polinomial é unimodular quando o seu determinante é não nulo e independe da variável complexa s.* □

As matrizes unimodulares podem, também ser caracterizadas da seguinte forma:

Fato 2.1 *Uma matriz polinomial $U(s)$ é unimodular se e somente se sua inversa $U^{-1}(s)$ é também polinomial.* □

A partir do fato 2.1, é fácil verificar que a inversa de uma matriz unimodular é uma matriz unimodular.

No presente trabalho, matrizes unimodulares serão importantes na realização de operações com matrizes polinomiais em que não se deseja alterar as características destas matrizes , como por exemplo:

- Criar novas matrizes polinomiais sem alteração de posto dessas matrizes. A redução por coluna de matrizes é um exemplo e será discutida na seção 2.6;
- Verificar se duas matrizes polinomiais são coprimas conforme será considerado na seção a seguir

2.4 DFM irredutíveis

Toda matriz $H(s) \in \mathbb{R}^{p\times m}(s)$ pode ser representada por DFM à direita ou à esquerda, da seguinte forma:

$$H(s) = N(s)M^{-1}(s) = \tilde{M}^{-1}(s)\tilde{N}(s)$$

onde $M(s)$ e $N(s)$ representam uma DFM à direita de $H(s)$ e $\tilde{M}(s)$ e $\tilde{N}(s)$ representam uma DFM à esquerda de $H(s)$, sendo $N(s), \tilde{N}(s) \in \mathbb{R}^{p\times m}[s]$, $M(s) \in \mathbb{R}^{m\times m}[s]$ e $\tilde{M}(s) \in \mathbb{R}^{p\times p}[s]$.

As Descrições por frações de matrizes não são únicas para uma mesma $H(s)$. Para tanto, note que, para uma matriz $W(s)$ não singular, pode-se definir:

$$\overline{N}(s) = N(s)W(s) \text{ e } \overline{M}(s) = M(s)W(s).$$

Desta forma:

$$H(s) = N(s)W(s)W^{-1}(s)M^{-1}(s) = \overline{N}(s)\overline{M}^{-1}(s)$$

e, portanto, $\overline{N}(s)\overline{M}^{-1}(s)$ representa uma outra DFM de $H(s)$. Definindo o grau de uma DFM como:

$$gr[\text{DFM}] = gr[|M(s)|] \text{ ou } gr[\text{DFM}] = gr[|\tilde{M}(s)|]$$

onde $|\cdot|$ denota o determinante e gr denota o grau. É fácil perceber que o grau de uma DFM pode se feito arbitrariamente grande. Além disso, é possível verificar que o menor grau do determinante de $M(s)$ de todas as DFMs de H(s), seja à esquerda ou à direita, será também igual à ordem mínima de todas as realizações em espaço de estados de $H(s)$ que será mostrado na seção 2.7. Neste ponto, seja $W(s)$ um divisor comum direita de $N(s)$ e $M(s)$. Note que,

$$gr[|\overline{M}(s)|] = gr([|M(s)|] \ + \ gr[|W(s)|]),$$

Em outras palavras, o grau de uma DFM pode ser reduzido removendo-se divisores comuns à direita (ou à esquerda) do numerador e denominador das matrizes polinomiais. Portanto, uma DFM de menor grau pode ser obtida extraindo-se um divisor comum de $M(s)$ e $N(s)$ ou $\tilde{M}(s)$ e $\tilde{N}(s)$.

Uma outra consideração importante que se pode tirar do MDC é que se $gr[|M(s)|] = gr[|\overline{M}(s)|]$, então o grau de $|W(s)|$ é zero e, portanto, $W(s)$ será uma matriz unimodular. A seguinte definição é uma conseqüência deste fato.

Definição 2.3 *:*

1. *Duas matrizes polinomiais $M(s)$ e $N(s)$ com o mesmo número de colunas são coprimas à direita se e somente se todos os seus MDC são unimodulares.*
2. *Uma DFM $H(s) = N(s)M^{-1}(s)$ é irredutível se $N(s)$ e $M(s)$ são coprimas à direita.*

□

Observação 2.1 *DFMs irredutíveis não são únicas; basta multiplicá-las por matrizes unimodulares.* □

2.5 Identidade de Bezout

Uma outra maneira de se verificar se duas matrizes polinomiais são coprimas à direita é através da **Identidade de Bezout**. Isto é garantido pelo seguinte resultado.

Fato 2.2 *Duas matrizes polinomiais $N(s)$ e $M(s)$ serão ditas coprimas à direita se e somente se existem matrizes polinomiais $X(s)$ e $Y(s)$ tal que*

$$X(s)N(s) + Y(s)M(s) = I \tag{2.16}$$

Prova: Ver Kailath (1980, página 379) □

Da equação (2.16) também pode-se tirar o seguinte lema

Lema 2.1 *$N(s)$ e $M(s)$ são coprimas à direita se e somente se $[M^T(s) \quad N^T(s)]^T$ tem posto completo para todo s, isto é, se e somente se pode ser reduzida por operações elementares de linhas e colunas a $[I \quad 0]^T$*

Prova: Ver Kailath (1980, página 379) □

Uma importante aplicação da identidade de Bezout é a obtenção de uma fatoração duplamente coprima de uma dada matriz de transferência. Este problema pode ser formulado da seguinte forma: dada uma matriz $H(s)$ tal que

$$H(s) = N(s)M^{-1}(s) = \tilde{M}^{-1}(s)\tilde{N}(s)$$

onde $N(s), M(s)$ são coprimas à direita e $\tilde{M}(s), \tilde{N}(s)$ são coprimas à esquerda. Então existirão matrizes polinomiais $X(s), Y(s), \tilde{X}(s), \tilde{Y}(s)$ tais que:

$$\begin{bmatrix} \tilde{X}(s) & -\tilde{Y}(s) \\ -\tilde{N}(s) & \tilde{M}(s) \end{bmatrix} \begin{bmatrix} M(s) & Y(s) \\ N(s) & X(s) \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \tag{2.17}$$

e

$$\begin{bmatrix} M(s) & Y(s) \\ N(s) & X(s) \end{bmatrix} \begin{bmatrix} \tilde{X}(s) & -\tilde{Y}(s) \\ -\tilde{N}(s) & \tilde{M}(s) \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}. \tag{2.18}$$

Às equações 2.17 e 2.18 acima, dá-se o nome de identidades de Bezout generalizadas.

2.6 Redução por colunas

Uma matriz $H(s) \in \mathbb{R}^{p\times m}(s)$ é própria se

$$\lim_{s\to\infty} H(s) < \infty$$

e estritamente própria se

$$\lim_{s\to\infty} H(s) = 0$$

No caso monovariável, uma função de transferência é estritamente própria se o grau do polinômio do numerador é menor do que o do denominador e própria se os graus são iguais. No caso multivariável, quando $H(s)$ é descrita por uma DFM, a situação não é tão simples, não basta apenas comparar os graus das matrizes do numerador e do denominador da DFM. Conseqüentemente, são necessários alguns conceitos para estender esta característica para matrizes polinomiais. Um destes conceitos é a Redução por Coluna ou por linha. Por uma questão de simplicidade, o estudo que será desenvolvido aqui ficará restrito à Redução por Coluna; porém usando a dualidade, é possível reproduzir este estudo para a Redução por Linha.

Definição 2.4 *Seja $M(s) \in \mathbb{R}^{m\times m}[s]$ e escreva*

$$M(s) = [\ \underline{m}_1(s) \quad \underline{m}_2(s) \quad \ldots \quad \underline{m}_m(s)\ ]$$

onde $gr[\underline{m}_i(s)] = \nu_i, i = 1, 2, ..., m$, *denota o maior grau da i-ésima coluna . Então* $M(s)$ *é reduzida por coluna se e somente se:*

$$grau\{det[M(s)]\} = \sum_{i=1}^{m} \nu_i.$$

□

Traduzindo em palavras, pode-se dizer que uma matriz é reduzida por coluna quando o grau do determinante da matriz polinomial é igual o somatório dos graus das colunas desta matriz. Isto pode ser visto num exemplo. Seja a matriz

$$M(s) = \begin{bmatrix} a_1 s^3 + a_2 s & b_1 s + b_2 \\ c_1 s^2 + c_2 s + c_3 & d_1 \end{bmatrix}. \tag{2.19}$$

Pode ser observado que para $a_1 d_1 \neq 0$ e $b_1 c_1 \neq 0$, o grau do determinante de $M(s)$ será menor ou igual 3, porém o somatório dos graus das colunas é 4. Portanto a matriz da equação (2.19) não é reduzida por coluna de acordo com a definição 2.4.

Uma outra forma de mostrar que uma matriz polinomial é reduzida por coluna Kailath (1980, página 384) é escrevendo-se a matriz $M(s) \in \mathbb{R}^{m \times m}[s]$ da seguinte forma:

$$M(s) = M_{hc} S(s) + M_{lc} \Psi(s) \tag{2.20}$$

e verificar a singularidade da matriz M_{hc}. Isto pode ser mais bem visualizado usando-se a matriz da equação (2.19) e reescrevendo-a conforme a equação (2.23), obtendo-se:

$$M(s) = \begin{bmatrix} a_1 & b_1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} s^3 & 0 \\ 0 & s \end{bmatrix} + \begin{bmatrix} a_2 s & b_2 \\ c_1 s^2 + c_2 s + c_3 & d_1 \end{bmatrix}. \tag{2.21}$$

Note que a matriz M_{hc} é singular, e isto representa uma outra forma de se verificar se uma matriz é reduzida por coluna, conforme mostra o resultado a seguir.

Lema 2.2 *Seja* $M(s)\mathbb{R}^{p \times p}[s]$ *e escreva* $M(s) = M_{hc} S(s) + M_{lc} \Psi(s)$. *Então* $M(s)$ *será reduzida por coluna se somente se* M_{hc} *for não-singular.*

Prova: Ver Kailath (1980, página 384) □

De posse do conceito de redução por coluna, pode-se estabelecer uma definição coerente sobre uma matriz racional ser própria ou não, conforme mostrado a seguir.

Lema 2.3 *Se $M(s)$ é reduzida por coluna, então*

$$H(s) = N(s)M^{-1}(s)$$

será estritamente própria (ou própria) se somente se cada coluna de $N(s)$ possui grau menor (ou igual) do que os graus da correspondente coluna de $M(s)$.

Prova: Ver Kailath (1980, página 385) □

A redução por coluna (linha) de uma matriz polinomial não-singular qualquer pode ser obtida através de operações elementares com colunas (linhas), isto é, pós-multiplicando-se a matriz original por matrizes unimodulares, é possível obter uma matriz reduzida por coluna. Na seção 4.1.1 será mostrado um algoritmo que executa esta tarefa.

2.7 Obtenção de uma Realização em Espaço de Estados a partir de uma DFM.

Nesta seção será mostrado como obter uma realização em espaço de estados na forma controladora para uma dada DFM. Este ferramenta é fundamental em muitos problemas de sistemas multivariáveis. Um pré-requisito para o estudo desenvolvido nesta seção é que a matriz racional seja estritamente própria. Se uma matriz racional $H(s)$ for própria, então ela pode ser escrita como:

$$H(s) = H_{ep}(s) + D,$$

onde $D \in \mathbb{R}^{p\times m}$ é o quociente da divisão dos polinômios do numerador de H(s) pelos seus respectivos denominadores e $H_{ep}(s)$ é a nova função racional estritamente própria. Seja, portanto,

$$H(s) = N(s)M^{-1}(s)$$

uma DFM à direita (não necessariamente coprima) de uma matriz $H(s) \in \mathbb{R}^{p\times m}[s]$ estritamente própria. O objetivo nesta seção é apresentar um procedimento para obter uma realização controlável de ordem igual ao $gr[|M(s)|]$, isto é:

$$H(s) \left\{ \begin{array}{l} \dot{x}(t) = A_c x(t) + B_c u(t). \\ y(t) = C_c x(t) \end{array} \right. \tag{2.22}$$

O primeiro passo para se obter essa realização, é escrever $M(s) \in \mathbb{R}^{m\times m}[s]$ e $N(s) \in \mathbb{R}^{p\times m}[s]$ como

$$\begin{cases} M(s) = M_{hc}S(s) + M_{lc}\Psi(s), \\ N(s) = N_{lc}\Psi(s) \end{cases} \tag{2.23}$$

onde

$$S(s) = diag\{s^{\nu_1}, ..., s^{\nu_m}\}, i = 1, 2, ..., m \tag{2.24}$$

com ν_i os graus das colunas de $M(s)$, M_{hc} é a matriz de coeficientes dos maiores graus de cada coluna de $M(s)$, M_{lc} é formada pela matriz de coeficientes dos elementos que permaneceram com retirada do M_{hc}, ou seja, os termos das colunas com os graus remanescentes da retirada de M_{hc},

$$\Psi^T \triangleq \left[\begin{array}{cccc|ccc|c|ccc} s^{\nu_1-1} & ... & s & 1 & 0 & ... & 0 & ... & 0 & ... & 0 \\ \hline 0 & ... & 0 & 0 & s^{\nu_2-1} & ... & 1 & ... & 0 & ... & 0 \\ \hline 0 & ... & 0 & 0 & 0 & ... & 0 & \ddots & 0 & ... & 0 \\ \hline 0 & ... & 0 & 0 & 0 & ... & 0 & ... & s^{\nu_m-1} & ... & 0 \end{array}\right] \tag{2.25}$$

ou

$$\Psi^T(s) = bloco\ \ diagonal\{\left(s^{\nu_1-1} \quad ..., \quad s \quad 1\right), ..., \left(s^{\nu_m-1} \quad ..., \quad 1\right)\} \tag{2.26}$$

e N_{lc} é construída em função de $\Psi(s)$. Por exemplo, seja

$$M(s) = \begin{bmatrix} a_1s^3 + a_2s & b_1s + b_2 \\ c_1s^2 + c_2s + c_3 & d_1 \end{bmatrix}.$$

Então, $M(s)$ pode ser escrita como:

$$M(s) = \underbrace{\begin{bmatrix} a_1 & b_1 \\ 0 & 0 \end{bmatrix}}_{M_{hc}} \underbrace{\begin{bmatrix} s^3 & 0 \\ 0 & s \end{bmatrix}}_{S(s)} + \underbrace{\left[\begin{array}{ccc|c} 0 & a_2 & 0 & b_2 \\ \hline c_1 & c_2 & c_3 & d_1 \end{array}\right]}_{Dlc} \underbrace{\left[\begin{array}{c|c} s^2 & 0 \\ s & 0 \\ 1 & 0 \\ \hline 0 & 1 \end{array}\right]}_{\Psi(s)}.$$

Para se chegar à realização de estados pretendida, é necessário que M_{hc} seja inversível. Portanto, $M(s)$ deverá ser reduzida por coluna. Caso a matriz $M(s)$ não seja reduzida por coluna, deve-se multiplicá-la por matrizes unimodulares, até que seja encontrada uma matriz reduzida por coluna equivalente. É importante ressaltar que, após encontrada a matriz reduzida por coluna, ela não alterará o grau do determinante de $M(s)$ original, porque conforme mostrado na seção 2.3, as matrizes

unimodulares não alteram o grau do determinante de uma matriz polinomial.

Depois de obtidas as matrizes M_{hc}, M_{lc} e N_{lc} e sendo M_{hc} inversível, é possível calcular as matrizes da realização (2.22) da seguinte forma (Kailath 1980, página 406):

$$\begin{cases} A_c = A_{c0} - B_{c0}M_{hc}^{-1}M_{lc} \\ B_c = B_{c0}M_{hc}^{-1} \\ C_c = N_{lc} \end{cases} \tag{2.27}$$

com

$$A_{co} = Blocodiagonal\{A_{co}^1, A_{co}^2, \ldots, A_{co}^m\} \tag{2.28}$$

onde

$$A_{co}^{(i)} = \left\{ \begin{bmatrix} 0 & 0 & ... & 0 & 0 \\ 1 & 0 & ... & 0 & 0 \\ 0 & 1 & ... & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & ... & 1 & 0 \end{bmatrix}, \nu_i \times \nu_i, i = 1, 2, ..., m \right\} \tag{2.29}$$

e

$$B_{co} = \begin{bmatrix} B_{co}^1 \\ B_{co}^2 \\ \vdots \\ B_{co}^m \end{bmatrix} \tag{2.30}$$

com

$$B_{c0}^{(1)} = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 \end{bmatrix}_{(\nu_1 \times m)}, B_{c0}^{(2)} = \begin{bmatrix} 0 & 1 & \ldots & 0 \\ 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 \end{bmatrix}_{(\nu_2 \times m)}, \ldots,$$

$$B_{c0}^{(m)} = \begin{bmatrix} 0 & 0 & \ldots & 1 \\ 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 \end{bmatrix}_{(\nu_m \times m)}, \text{ para } i = 1, 2, \ldots, m$$

Observação 2.2 *:*

1. *É imediato provar que o par (A_c, B_c) é controlável. Além disso, de acordo com o teste de Popov-Belevitch-Hautus, quando $N(s)$ e $M(s)$ forem coprimas à direita então o par (C_c, A_c) será observável e, portanto, A_c terá ordem mínima.*

2. Se for dada uma DFM à esquerda, isto é,

$$H(s) = \tilde{M}^{-1}(s)\tilde{N}(s),$$

então, basta aplicar o procedimento descrito nessa seção a

$$\tilde{H}(s) = H^T(s) = \tilde{N}^T(s)(\tilde{M}^T)^{-1}(s).$$

□

2.8 Base Polinomial Mínima

Definição 2.5 *Uma matriz $M(s) \in \mathbb{R}^{p\times m}(s)$, $(p \leq m)$ é denominada singular quando ela é não quadrada e/ou não inversível (Kailath 1980, página 455).* □

Pode ser observado que quando isto ocorre, então existe pelo menos um vetor $f(s) \in \mathbb{R}^m[s]$ tal que

$$M(s)\underline{f}(s) = \underline{0} \tag{2.31}$$

definindo portanto, um espaço nulo à direita de $M(s)$. Note que todos os vetores do espaço nulo serão ortogonais às linhas de $M(s)$, sendo a dimensão desse espaço nulo dada por:

$$\alpha = m - r$$

onde α é a dimensão do espaço nulo de $M(s)$ e r denota o posto normal de $M(s)$. Uma das maneiras de se calcular o espaço nulo de $M(s)$, é, inicialmente, escolher entre todos os vetores $\underline{f}(s)$ que satisfazem a equação 2.31, aquele de menor grau ($\underline{f_1}(s)$ de grau igual a ν_1); para o próximo vetor deve-se escolher, $f_2(s)$, de grau ν_2 (o menor possível) , onde $M(s)\underline{f_2}(s) = 0$, $\nu_2 \geq \nu_1$, e continuando desta forma, um conjunto de vetores linearmente independentes de dimensão α , formará a seguinte matriz $F(s) \in \mathbb{R}^{m\times\alpha}[s]$:

$$F(s) = \left[\begin{array}{cccc} \underline{f}_1(s) & \underline{f}_2(s) & \ldots & \underline{f}_\alpha(s) \end{array} \right], \tag{2.32}$$

com os graus das colunas dadas por

$$\nu_1 \leq \nu_2 \leq \ldots \leq \nu_\alpha.$$

Naturalmente, existirão diversas soluções para a equação (2.31); contudo, como mencionado em Kailath (1980, pág.456) , elas deverão possuir o mesmo grau para cada vetor $\underline{f}_i(s)$. Portanto, qualquer conjunto de soluções para o espaço nulo à direita de $M(s)$ deverá possuir o mesmo conjunto de índices (graus) mínimos para os polinômios das colunas de $F(s)$.

Definição 2.6 *Ao conjunto de vetores* $\{\underline{f}_1(s), \underline{f}_2(s), \ldots, \underline{f}_\alpha(s)\}$, $\nu_1 \leq \nu_2 \leq \ldots \leq \nu_\alpha$ *tal que* $\sum_i^\alpha \nu_i$ *é mínimo, dá-se o nome de base polinomial mínima para o espaço nulo da matriz* $M(s)$. □

A caracterização da matriz $F(s)$ é feita de acordo com o seguinte teorema.

Teorema 2.1 *Seja a matriz*

$$F(s) = \left[\begin{array}{cccc} \underline{f}_1(s) & \underline{f}_2(s) & \ldots & \underline{f}_\alpha(s) \end{array} \right]$$

tal que

$$M(s)F(s) = 0,$$

com os graus da colunas dados por

$$\nu_1 \leq \nu_2 \leq \ldots \leq \nu_\alpha.$$

Então as seguintes sentenças são equivalentes:

1. *F(s) é uma base polinomial mínima para o espaço nulo de M(s).*
2. *F(s) é reduzida por coluna e irredutível*[1]
3. *F(s) é de ordem mínima, isto é,*

$$\sum_{i=1}^{\alpha} \nu_i \text{ é mínimo.}$$

Prova: Ver Kailath (1980, teorema 6.5-10, página 458.) □

[1] Uma matriz é irredutível se tem posto completo para todo valor de s

2.9 Forma de Smith

A forma de Smith é um conceito relevante para o estudo de matrizes polinomiais. Com ela é possível achar o posto normal de uma matriz polinomial, e além disso, através da Forma de Smith McMillan, é possível achar os pólos e zeros de um sistema multivariável. Seja, portanto, uma matriz qualquer $N(s) \in \mathbb{R}^{p\times m}[s]$. É possível obter, através de operações elementares de linhas e colunas, equivalentes a pré e pós-multiplicações por matrizes unimodulares $U(s) \in \mathbb{R}^{p\times p}[s]$ e $V(s) \in \mathbb{R}^{m\times m}[s]$, tais que

$$U(s)N(s)V(s) = \Sigma_N(s)$$

onde

$$\Sigma_N(s) = \left[\begin{array}{ccc|c} \sigma_1(s) & & & \\ & \sigma_2(s) & & 0_{r\times(m-r)} \\ & & \ddots & \\ & & \sigma_r(s) & \\ \hline & 0_{(p-r)\times r} & & 0_{(p-r)\times(m-r)} \end{array}\right], r \leq \min(p,m)$$

e $\sigma_i(s)$ são polinômios mônicos e divisores de $\sigma_{i+1}(s), i = 1, \ldots, r-1$.

Definição 2.7 *À matriz Σ_N, dá-se o nome de forma de Smith de $N(s)$.* □

Observação 2.3 *:*

1. *r é denominado posto normal de $N(s)$.*

2. *Seja $\Delta_i(s)$ o MDC mônico de todos os menores de ordem i da matriz $N(s)$. Pode-se provar que,*

$$\sigma_i = \frac{\Delta_i(s)}{\Delta_{i-1}(s)},$$

onde $\Delta_0(s) = 1$, por definição.

3. *Note que, como $U(s)$ e $V(s)$ são unimodulares, então $\rho[N(s)] = \rho[\Sigma_N(s)], \ \forall s$. Além disso, $N(s)$ perderá posto para todos os valores de $s = z$ tais que $\sigma_i(z) = 0$*

□

2.10 Decomposição por Valores Singulares (DVS)

2.10.1 Cálculo da DVS

A decomposição por valores singulares é considerada como o ponto alto da fatoração de uma matriz (Strang 1988) uma vez que esta fatoração apresenta robustez na sua solução e, além disso, a matriz a ser fatorada não necessita ser quadrada.

Nessa seção será apresentada uma breve revisão de DVS. Para tanto, seja a matriz $A \in \mathbb{R}^{m \times n}$ $(m \geq n)$. Então, existirão matrizes U e V tais que

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T \tag{2.33}$$

onde $\Sigma = diag(\sigma_1, ..., \sigma_n)$ U é uma matriz $m \times m$ unitária $(U^T U = I)$ formada pelos autovetores de AA^T, V é uma matriz $n \times n$ unitária $(V^T V = I)$ formada pelos autovetores de $A^T A$, σ_i são chamados valores singulares de A, sendo dados pelos autovalores de $A^T A$. Desta forma, é fácil perceber que os valores singulares σ_i são não negativos e, por hipótese, ordenados de forma decrescente[2] , ou seja,

$$\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n.$$

À equação (2.33) dá-se o nome de DVS de A. As colunas de U são chamadas vetores singulares à esquerda ou direções principais de saída e as colunas de V são chamadas vetores singulares à direita ou direções principais de entrada.

2.10.2 Utilização da DVS na determinação do posto de uma matriz

Uma da maiores aplicações da DVS é na determinação do posto de uma matriz. Se o posto de A é $(k < n)$, então tem-se:

$$\sigma_k > 0 = \sigma_{k+1} = ... = \sigma_n.$$

Contudo, computacionalmente, em geral, tem-se que

$$\sigma_{k+1} = \xi_{k+1}, \ldots, \xi_n = \sigma_n,$$

[2]Essa hipótese é fundamentada no fato de que, em geral, algoritmos numéricos para o cálculo de DVS retornam os valores singulares ordenados de forma decrescente

onde

$$\xi_i \approx 0, \quad i = k+1, \ldots, n.$$

Assim no cálculo do posto de uma matriz deve-se estabelecer um valor ξ (tolerância), abaixo do qual pode-se considerar, $\xi_i = 0$, caracterizando-se assim uma perda de posto. Para se entender melhor este fato, considere a seguinte matriz:

$$A = \begin{bmatrix} 10 & 10 & 10 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.34}$$

Note que a matriz A é claramente singular e tem posto igual a 1. Utilizando o Matlab, vê-se que A tem os seguintes valores singulares.

$$\Sigma_A = \begin{bmatrix} 17.4929 & 0 & 0 \end{bmatrix} \tag{2.35}$$

e, portanto, neste caso, $\sigma_2 = \sigma_3 = 0$. Caso ocorra uma pequena perturbação em algum valor da matriz A, como por exemplo,

$$A'_p = \begin{bmatrix} 10 & 10 & 10 \\ 1 & 1 & 1 \\ 1 & 0.9999 & 1 \end{bmatrix}, \tag{2.36}$$

os seus respectivos valores singulares tornam-se iguais a

$$\Sigma_{A'_p} = \begin{bmatrix} 17.4928 & 0.0001 & 0 \end{bmatrix}. \tag{2.37}$$

Observando-se os valores singulares dados em (2.37), é visto que, rigorosamente, a matriz A'_p tem posto igual a 2, porém as colunas 1 e 2 de A'_p são formadas por vetores aproximadamente paralelos. Assim, se for adotado $\xi = 1 \times 10^{-3}$ então, $\sigma_2 < \xi$ e portanto, pode-se dizer que o posto de A'_p é igual a 1.

Uma outra consideração importante a ser feita, é quanto à necessidade de normalização dos valores singulares. Para esclarecer esse fato, suponha, por exemplo, que a matriz A'_p dada em (2.36) seja multiplicada, por exemplo, por um fator 10^5, obtendo-se a matriz A_p". Os seus valores singulares ficariam iguais a

$$\Sigma_{A"_p} = \begin{bmatrix} 1.7493 \times 10^6 & 8.1248 & 2.6645 \times 10^{-15} \end{bmatrix}. \tag{2.38}$$

A interpretação de quantos valores numéricos são nulos, pode ser equivocada, uma vez que $\sigma_2 \gg 0$. Contudo, as colunas 1 e 2 de A_p" são ainda aproximadamente paralelas, isto é, desalinhadas em apenas 5.6×10^{-4} graus. Portanto, A_p" deve, também, ser considerada de posto igual a 1. Esse problema pode ser evitado dividindo-se todos os valores singulares por σ_1.

Em resumo, a pesquisa do número de valores singulares nulos pode ser feita da seguinte forma:

1. Arbitrar um valor ξ de tolerância;
2. Dividir $\sigma_i, i = 1, \ldots, n$, por σ_1 obtendo-se $\overline{\sigma}_i = \sigma_i/\sigma_1$;
3. Encontrar o número de valores singulares tais que $\overline{\sigma}_i < \xi$.

Capítulo 3

Algoritmos Básicos para Operações com Matrizes Polinomiais

Conforme comentado anteriormente, o objetivo desse trabalho é apresentar algoritmos robustos utilizando como parâmetros de entrada e saída as matrizes de coeficientes das matrizes polinomiais. Contudo, para usá-los é necessário criar algoritmos básicos, que permitirão a operacionalização dos algoritmos mais complexos, como os que serão expostos no capítulo 4. Os algoritmos apresentados, neste capítulo, estão subdivididos de acordo com as seguintes funcionalidades:

1. Modificar o formato das matrizes de coeficientes, objetivando, principalmente, a facilidade de utilização. Esses algoritmos serão apresentados na seção 3.1;

2. Retirar informações relevantes das matrizes polinomiais, sendo também apresentados na seção 3.1;

3. Fazer as operações básicas de soma e multiplicação, que serão vistos na seção 3.2;

4. Fazer operações com apenas uma matriz polinomial, que serão descritos na seção 3.3;

Ainda neste capítulo, será apresentado na seção 3.4 um algoritmo que detecta o número de valores singulares nulos de forma robusta, levando em consideração o que foi discutido na seção 2.10.

3.1 Algoritmos Auxiliares

Nesta seção serão apresentados os algoritmos que modificam a estrutura das matrizes de coeficientes, para facilitar o seu uso, e os que retiram informações das matrizes polinomiais através da sua correspondente matriz de coeficientes.

3.1.1 Matsplit

Muitas vezes, há necessidade de se trabalhar com os coeficientes dos polinômios dos elementos de uma matriz polinomial individualmente. O algoritmo Matsplit forma uma matriz cujas linhas são os coeficientes dos polinômios de cada elemento de uma matriz polinomial a partir da matriz de coeficientes, representando o empilhamento das colunas dessa matriz.

Descrição do algoritmo do Matsplit

Este algoritmo é mais bem descrito utilizando-se uma matriz polinomial como exemplo. Seja

$$N(s) = \begin{bmatrix} n_{11}(s) & n_{12}(s) & \dots & n_{1m}(s) \\ n_{21}(s) & n_{22}(s) & \dots & n_{2m}(s) \\ \vdots & \vdots & \ddots & \vdots \\ n_{p1}(s) & n_{p2}(s) & \dots & n_{pm}(s) \end{bmatrix} \in \mathbb{R}^{p\times m}[s].$$

Supondo que os graus de todos os elementos polinomiais são os mesmos[1] e iguais a ν, tem-se a seguinte matriz:

$$N(s) = \left[\begin{array}{c|c|c} n_{11_\nu}s^\nu + n_{11_{\nu-1}}s^{\nu-1} + \ldots + n_{11_0} & \dots & n_{1m_\nu}s^\nu + n_{1m_{\nu-1}}s^{\nu-1} + \ldots + n_{1m_0} \\ n_{21_\nu}s^\nu + n_{21_{\nu-1}}s^{\nu-1} + \ldots + n_{21_0} & \dots & n_{2m_\nu}s^\nu + n_{2m_{\nu-1}}s^{\nu-1} + \ldots + n_{2m_0} \\ \vdots & & \vdots \\ n_{p1_\nu}s^\nu + n_{p1_{\nu-1}}s^{\nu-1} + \ldots + n_{p1_0} & \dots & n_{pm_\nu}s^\nu + n_{pm_{\nu-1}}s^{\nu-1} + \ldots + n_{pm_0} \end{array}\right] \tag{3.1}$$

Note que a matriz $N(s)$ possui a seguinte matriz coeficiente:

$$N = \left[\begin{array}{ccc|ccc|c|ccc} n_{11_\nu} & \dots & n_{1m_\nu} & n_{11_{\nu-1}} & \dots & n_{1m_{\nu-1}} & \dots & n_{11_0} & \dots & n_{1m_0} \\ n_{21_\nu} & \dots & n_{2m_\nu} & n_{21_{\nu-1}} & \dots & n_{2m_{\nu-1}} & \dots & n_{21_0} & \dots & n_{2m_0} \\ \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ n_{p1_\nu} & \dots & n_{pm_\nu} & n_{p1_{\nu-1}} & \dots & n_{pm_{\nu-1}} & \dots & n_{p1_0} & \dots & n_{pm_0} \end{array}\right] \tag{3.2}$$

[1] Quando isto não ocorre deve-se completar com zeros.

onde $N \in \mathbb{R}^{p\times m(\nu+1)}$. O algoritmo Matsplit transforma a matriz N na seguinte matriz:

$$Nvec = \left[\begin{array}{cccc} n_{11_\nu} & n_{11_{\nu-1}} & \dots & n_{11_0} \\ n_{21_\nu} & n_{21_{\nu-1}} & \dots & n_{21_0} \\ \vdots & \vdots & & \vdots \\ n_{p1_\nu} & n_{p1_{\nu-1}} & \dots & n_{p1_0} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \hline n_{1m_\nu} & n_{1m_{\nu-1}} & \dots & n_{1m_0} \\ n_{2m_\nu} & n_{2m_{\nu-1}} & \dots & n_{2m_0} \\ \vdots & \vdots & & \vdots \\ n_{pm_\nu} & n_{pm_{\nu-1}} & \dots & n_{pm_0} \end{array}\right]. \tag{3.3}$$

Note que N_{vec} é uma matriz obtida empilhando-se os coeficientes dos polinômios das colunas da matriz $N(s)$, representando, portanto, uma matriz coeficiente do seguinte vetor:

$$\underline{n} = \left[\begin{array}{c} \underline{n}_1(s) \\ \underline{n}_2(s) \\ \vdots \\ \underline{n}_p(s) \end{array}\right],$$

onde $\underline{n}_i(s)$ denota a i-ésima coluna da matriz $N(s)$.

A função `matsplit`

A função Matlab desenvolvida para realizar o algoritmo descrito nesta seção é a `matsplit.m` (apêndice A.1). Para usá-la o usuário entra com a matriz de coeficientes e o número de colunas (caso a matriz não seja quadrada) e a função retorna a matriz coeficiente do vetor obtido empilhando-se as colunas da matriz polinomial. Essa função tem a seguinte sintaxe:

$$Nvec = \texttt{matsplit}(N, m),$$

onde N é a matriz coeficiente e m o número de colunas de $N(s)$

Exemplo 3.1 *Para mostrar o funcionamento do algoritmo, seja a matriz polinomial*

$$N(s) = \begin{bmatrix} s^3+s^2+5s+3 & -s^2-3s+1 & 2s^4+s^3+2s+1 \\ -3 & -2 & s^2+5s+1 \\ s^3+5s+4 & -s^2 & 2s^4+s^3+3s^2+4s+5 \end{bmatrix}.$$

A matriz de coeficientes de $N(s)$ é dada por:

$$N = \left[\begin{array}{ccc|ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 2 & 1 & 0 & 1 & 0 & -1 & 3 & 5 & 0 & 4 & 4 & 0 & 5 \end{array}\right] \tag{3.4}$$

Para se usar a função `matsplit` *no Matlab procede-se da seguinte forma:*

```
>> N = [0  0  2  1  0  1  1  -1  0  5  -3  2   3   1  1;
        0  0  0  0  0  0  0   0  1  0   0  5  -3  -2  1;
        0  0  2  1  0  1  0  -1  3  5   0  4   4   0  5]
>> Nvec = matsplit(N, 3);
```

ou

```
>> Nvec = matsplit(N);
```

[2]

Obtendo-se:

$$N_{vec} = \left[\begin{array}{ccccc} 0 & 1 & 1 & 5 & 3 \\ 0 & 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 5 & 4 \\ \hline 0 & 0 & -1 & -3 & 1 \\ 0 & 0 & 0 & 0 & -2 \\ 0 & 0 & -1 & 0 & 0 \\ \hline 2 & 1 & 0 & 2 & 1 \\ 0 & 0 & 1 & 5 & 1 \\ 2 & 1 & 3 & 4 & 5 \end{array}\right] \tag{3.5}$$

□

Como pode ser observado na expressão (3.5), as linhas de N_{vec} correspondem aos coeficientes dos polinômios após empilhar as colunas da matriz polinomial.

3.1.2 Vecmat

Alguns algoritmos implementados retornam, após as suas operações internas, uma matriz com a estrutura daquela mostrado na equação (3.3). Contudo, é padronizado

[2]Esta possibilidade existe devido à matriz polinomial ser quadrada, não havendo, portanto, a necessidade do segundo argumento da função

neste trabalho, que os resultados dos algoritmos devem ser sempre apresentados na forma de matriz coeficiente. Logo, torna-se necessário criar um algoritmo que converta a matriz com os coeficientes dos polinômios da matriz polinomial em uma matriz coeficiente.Este algoritmo é denominado Vecmat.

Descrição do algoritmo Vecmat

Seja a matriz cujas linhas são os coeficientes dos polinômios de uma matriz polinomial, conforme a equação (3.3). O algoritmo retorna à matriz da equação (3.2). Obviamente, o algoritmo Vecmat funciona como uma operação inversa ao Matsplit, conforme comentado na seção 3.1.1.

A função `vecmat`

A função Matlab desenvolvida para realizar o algoritmo descrito nesta seção é a `vecmat.m` (apêndice A.2). Para usá-la, o usuário entra com uma matriz de coeficientes dos polinômios representativa do empilhamento das colunas da matriz polinomial, o número de linhas e o número de colunas (caso a matriz não seja quadrada) da matriz polinomial e a função retorna a matriz de coeficientes correspondente.Essa função tem a seguinte sintaxe:

$$N = \texttt{vecmat}(Nvec, p, m),$$

onde $Nvec$ é formada como mostrado na equação (3.3), p denota o número de linhas de $N(s)$ e m é o número de colunas de $N(s)$.

Exemplo 3.2 *Para mostrar o funcionamento do algoritmo, seja a vetor polinomial*

$$\underline{n}(s) = \begin{bmatrix} 5s^2 + 2s + 3 \\ -4 \\ s^2 \\ 3s + 1 \\ -2s \\ -s^2 \\ 2s^3 \\ 5s + 1 \\ 2s^3 + 4s + 5 \end{bmatrix}$$

obtido empilhando-se as colunas de uma matriz $N(s) \in \mathbb{R}^{3\times 3}[s]$. *A matriz coeficiente de* $\underline{n}(s)$ *é dada por:*

$$N_{vec} = \begin{bmatrix} 0 & 5 & 2 & 3 \\ 0 & 0 & 0 & -4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & -2 & 0 \\ 0 & -1 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 5 & 1 \\ 2 & 0 & 4 & 5 \end{bmatrix}$$

□

Para se usar a função `vecmat` no Matlab, procede-se da seguinte forma:

$$\begin{aligned} >> Nvec = [&0 \quad 5 \quad 2 \quad 3; \\ &0 \quad 0 \quad 0 \quad -4; \\ &0 \quad 1 \quad 0 \quad 0; \\ &0 \quad 0 \quad 3 \quad 1; \\ &0 \quad 0 \quad -2 \quad 0; \\ &0 \quad -1 \quad 0 \quad 0; \\ &2 \quad 0 \quad 0 \quad 0; \\ &0 \quad 0 \quad 5 \quad 1; \\ &2 \quad 0 \quad 4 \quad 5] \end{aligned}$$

$>> N =$ `vecmat`$(Nvec, 3, 3);$

ou

$>> N =$ `vecmat`$(Nvec, 3);$

Obtém-se então:

$$N = \left[\begin{array}{ccc|ccc|ccc} 0 & 0 & 2 & 5 & 0 & 0 & 2 & 3 & 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 5 & -4 & 0 & 1 \\ 0 & 0 & 2 & 1 & -1 & 0 & 0 & 0 & 4 & 0 & 0 & 5 \end{array}\right]$$

Note que é possível, também, usar apenas dois argumentos na função `vecmat`, uma vez que a matriz $N(s)$ é quadrada.

3.1.3 Cutzeros

Algumas vezes, nas operações com as matrizes de coeficientes ocorre a redução do grau. Conseqüentemente, aparecerá uma matriz ou mais matrizes de zeros para estes graus, o que não traz nenhuma informação relevante para a matriz polinomial. Para

contornar esse inconveniente, esses coeficientes devem ser retirados, fornecendo a correta informação do grau da matriz. O algoritmo utilizado para tanto é o Cutzeros.

Descrição do algoritmo Cutzeros

Para mostrar o seu funcionamento, suponha, como motivação, as seguintes matrizes polinomiais

$$N(s) = N_\nu s^\nu + N_{\nu-1}s^{\nu-1} + ... + N_1 s + N_0, N_i \in \mathbb{R}^{p\times m}, i = 0, \ldots, \nu \tag{3.6}$$

$$D(s) = D_\nu s^\nu + D_{\nu-1}s^{\nu-1} + ... + D_1 s + D_0, N_i \in \mathbb{R}^{p\times m}, i = 0, \ldots, \nu \tag{3.7}$$

onde $N_\nu = D_\nu$ e suponha que se deseje realizar a operação de subtração com as duas matrizes. Obviamente, a matriz coeficientes $A = N - D$, será:

$$A = N - D = \left[\begin{array}{ccc|ccc|ccc} 0 & \ldots & 0 & a_{11_{\nu-1}} & \ldots & a_{1m_{\nu-1}} & \ldots\ldots & a_{11_0} & \ldots & a_{1m_0} \\ 0 & \ldots & 0 & a_{21_{\nu-1}} & \ldots & a_{2m_{\nu-1}} & \ldots\ldots & a_{21_0} & \ldots & a_{2m_0} \\ \vdots & \ldots & \vdots & \vdots & \ldots & \vdots & \ddots\ddots & \vdots & \ldots & \vdots \\ 0 & \ldots & 0 & a_{p1_{\nu-1}} & \ldots & a_{pm_{\nu-1}} & \ldots\ldots & a_{p1_0} & \ldots & a_{pm_0} \end{array}\right],$$

onde $a_{ij_k} = n_{ij_k} - d_{ij_k}$. Aplicando o algoritmo Cutzeros, deve-se obter:

$$Acut = \left[\begin{array}{ccc|ccc|ccc} a_{11_{\nu-1}} & \ldots & a_{1m_{\nu-1}} & \ldots\ldots & a_{11_0} & \ldots & a_{1m_0} \\ a_{21_{\nu-1}} & \ldots & a_{2m_{\nu-1}} & \ldots\ldots & a_{21_0} & \ldots & a_{2m_0} \\ \vdots & \ldots & \vdots & \ddots\ddots & \vdots & \ldots & \vdots \\ a_{p1_{\nu-1}} & \ldots & a_{pm_{\nu-1}} & \ldots\ldots & a_{p1_0} & \ldots & a_{pm_0} \end{array}\right],$$

onde os coeficientes iguais a zero dos elementos de grau igual a ν são eliminados. É importante ressaltar que, em geral, os coeficientes a serem cortados não são, necessariamente, exatamente iguais a zero. Assim, deve-se estabelecer uma maneira sistemática de cortar todos os elementos de uma matriz coeficiente. No presente trabalho, uma matriz coeficiente deverá ser cortada se todos os vetores colunas correspondentes à matriz coeficiente em análise, tiver norma quadrática menor que ε (tolerância).

A função `cutzeros`

A função Matlab desenvolvida para realizar o algoritmo descrito nesta seção é a `cutzeros.m` (apêndice A.3). Para usá-la, o usuário entra com a matriz de coeficientes que possui zeros em excesso e com o número de colunas (caso a matriz não

for quadrada) e a função retorna uma matriz coeficiente sem zeros em excesso. Essa função tem a seguinte sintaxe:

$$A = \texttt{cutzeros}(A, m),$$

onde A é uma matriz de coeficientes de $A(s)$ e m é o número de colunas da matriz.

Exemplo 3.3 *Sejam as matrizes polinomiais*

$$N(s) = \begin{bmatrix} s^3+s^2+5s+3 & -s^2-3s+1 & 2s^4+s^3+2s+1 \\ -3 & -2 & s^2+5s+1 \\ s^3+5s+4 & -s^2 & 2s^4+s^3+3s^2+4s+5 \end{bmatrix}$$

e

$$D(s) = \begin{bmatrix} 5s^3+2s+3 & 3s+1 & 2s^4 \\ -4 & -2s & 5s+1 \\ s^3 & -s^3 & 2s^4+4s+5 \end{bmatrix},$$

cujas matrizes coeficientes dadas por

$$N = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 2 & 1 & 0 & 1 & 0 & -1 & 3 & 5 & 0 & 4 & 4 & 0 & 5 \end{array}\right]$$

e

$$D = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 5 & 0 & 0 & 0 & 0 & 0 & 2 & 3 & 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 5 & -4 & 0 & 1 \\ 0 & 0 & 2 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 0 & 0 & 5 \end{array}\right].$$

Um exemplo de utilização da função `cutzeros` *no Matlab é o seguinte:*

$$\begin{array}{l} >> N = [0 \;\; 0 \;\; 2 \;\; 5 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 2 \;\; 3 \;\; 0 \;\; 3 \;\; 1 \;\; 0; \\ \qquad\quad 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; -2 \;\; 5 \;\; -4 \;\; 0 \;\; 1; \\ \qquad\quad 0 \;\; 0 \;\; 2 \;\; 1 \;\; -1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 4 \;\; 0 \;\; 0 \;\; 5] \\ >> D = [0 \;\; 0 \;\; 2 \;\; 5 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 2 \;\; 3 \;\; 0 \;\; 3 \;\; 1 \;\; 0; \\ \qquad\quad 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; -2 \;\; 5 \;\; -4 \;\; 0 \;\; 1; \\ \qquad\quad 0 \;\; 0 \;\; 2 \;\; 1 \;\; -1 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 0 \;\; 4 \;\; 0 \;\; 0 \;\; 5] \\ >> A = N - D \end{array}$$

que resulta em

$$A = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 0 & -4 & 0 & 1 & 1 & -1 & 0 & 3 & -6 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 1 & -2 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & 3 & 5 & 0 & 0 & 4 & 0 & 0 \end{array}\right].$$

Fazendo

$>> A = \texttt{cutzeros}(A, 3)$

ou

$>> A = \texttt{cutzeros}(A)$

Obtém-se:

$$A = \left[\begin{array}{ccc|ccc|ccc|ccc} -4 & 0 & 1 & 1 & -1 & 0 & 3 & -6 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 1 & -2 & 0 \\ 0 & 1 & 1 & 0 & -1 & 3 & 5 & 0 & 0 & 4 & 0 & 0 \end{array}\right].$$

□

3.1.4 Matform

A solução da equação de Sylvester

$$A(s)D(s) + B(s)N(s) = F(s) \tag{3.8}$$

onde $A(s), B(s)$ e $F(s)$ são matrizes polinomiais conhecidas e $D(s)$ e $N(s)$ devem ser determinadas e podem ser obtidas utilizando-se a matriz de Sylvester, conforme visto na seção 2.2.1. Para tanto, deve-se escrever a equação (3.8) da seguinte forma

$$\left[\begin{array}{cc} A(s) & B(s) \end{array}\right] \left[\begin{array}{c} D(s) \\ N(s) \end{array}\right] = F(s). \tag{3.9}$$

A idéia do algoritmo Matform é produzir uma matriz coeficiente formada com a concatenação de $A(s)$ e $B(s)$, isto é, formar uma matriz $C(s)$ tal que

$$C(s) = \left[\begin{array}{cc} A(s) & B(s) \end{array}\right] \tag{3.10}$$

Descrição do algoritmo do Matform

Sejam as matrizes, $A(s)$ e $B(s)$ ambas de grau igual a ν, isto é,

$$A(s) = A_\nu s^\nu + A_{\nu-1}s^{\nu-1} + \ldots + A_1 s + A_0, A_i \in \mathbb{R}^{p\times m}, i = 0, \ldots, \nu \tag{3.11}$$

e

$$B(s) = B_\nu s^\nu + B_{\nu-1}s^{\nu-1} + \ldots + B_1 s + B_0, B_i \in \mathbb{R}^{p\times n}, i = 0, \ldots, \nu. \tag{3.12}$$

É importante ressaltar que quando os graus são diferentes, os coeficientes das potências de s da matriz que tiver menor grau devem ser feitos identicamente nulos. A matriz dos coeficientes da matriz C pode ser obtida da seguinte forma.

Algoritmo 3.1 *:*

Passo 1*: Crie as matrizes de coeficientes*

$$A = \left[\begin{array}{cccccc} A_\nu & A_{\nu-1} & \dots & A_1 & A_0 \end{array} \right]$$

e

$$B = \left[\begin{array}{cccccc} B_\nu & B_{\nu-1} & \dots & B_1 & B_0 \end{array} \right]$$

Passo 2*: Forme a matriz*

$$C = \left[\begin{array}{cccccccccc} A_\nu & B_\nu & A_{\nu-1} & B_{\nu-1} & \dots & A_1 & B_1 & A_0 & B_0 \end{array} \right] \tag{3.13}$$

□

A função `matform`

A função Matlab desenvolvida para realizar o algoritmo descrito nesta seção é o `matform.m` (apêndice A.4). Para usá-la, o usuário entra com duas matrizes de coeficientes que deseja fazer a formação dada na equação (3.13) e o número de colunas de ambas as matrizes e a função retorna uma matriz de coeficientes como a equação (3.13). Essa função tem a seguinte sintaxe:

$$[C, q] = \texttt{matform}(A, B, m, n),$$

onde A e B são as matrizes coeficientes de $A(s)$ e $B(s)$, m e n são o número de colunas de $A(s)$ e $B(s)$, respectivamente, C é a matriz coeficiente de $C(s) = [A(s) \quad B(s)]$ e $q = m + n$ é o número de colunas de $C(s)$. Se o número de colunas de $A(s)$ e $B(s)$ forem iguais, não há a necessidade do 4º argumento.

Exemplo 3.4 *Sejam as matrizes polinomiais*

$$A(s) = \begin{bmatrix} 5s+3 & -3s+1 & 2s+1 \\ -3 & -2 & 5s+1 \\ 5s+4 & -s & 4s+5 \end{bmatrix} \tag{3.14}$$

e

$$B(s) = \begin{bmatrix} 2s+3 & 3s+1 & 2s \\ -4 & -2s & 2s+1 \\ s & 2 & 3s+5 \end{bmatrix}$$

cujas matrizes coeficientes dadas por

$$A = \left[\begin{array}{ccc|ccc} 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 5 & -3 & -2 & 1 \\ 5 & -1 & 4 & 4 & 0 & 5 \end{array}\right] \tag{3.15}$$

e

$$B = \left[\begin{array}{ccc|ccc} 2 & 3 & 2 & 3 & 1 & 0 \\ 0 & -2 & 2 & -4 & 0 & 1 \\ 1 & 0 & 3 & 0 & 2 & 5 \end{array}\right].$$

Para se obter a matriz coeficiente de $C(s) = [A(s) \quad B(s)]$ *no Matlab, procede-se da seguinte forma:*

$>> A = [5 \quad -3 \quad 2 \quad 3 \quad 1 \quad 1; \quad 0 \quad 0 \quad 5 \quad -3 \quad -2 \quad 1; \quad 5 \quad -1 \quad 4 \quad 4 \quad 0 \quad 5];$

$>> B = [2 \quad 3 \quad 2 \quad 3 \quad 1 \quad 0; \quad 0 \quad -2 \quad 2 \quad -4 \quad 0 \quad 1; \quad 1 \quad 0 \quad 3 \quad 0 \quad 2 \quad 5];$

$>> C = \texttt{matform}(A, B, 3, 3);$

ou

$>> C = \texttt{matform}(A, B, 3);$

Ao final desses comandos obtém-se:

$$C = \left[\begin{array}{ccc|ccc|ccc|ccc} 5 & -3 & 2 & 2 & 3 & 2 & 3 & 1 & 1 & 3 & 1 & 0 \\ 0 & 0 & 5 & 0 & -2 & 2 & -3 & -2 & 1 & -4 & 0 & 1 \\ 5 & -1 & 4 & 1 & 0 & 3 & 4 & 0 & 5 & 0 & 2 & 5 \end{array}\right]$$

□

3.1.5 Convform

As matrizes de Sylvester possuem destacada importância no cálculo com matrizes polinomiais, conforme foi visto na seção 2.2. Na seção 4.1.4 é mostrada uma outra aplicação desta matrizes no cálculo da base polinomial mínima de uma matriz polinomial. O algoritmo Convform forma a matriz de convolução ou matriz de Sylvester para uma dada matriz $A(s)$.

Descrição do algoritmo Convform

Seja a matriz

$$A(s) = A_\phi s^\phi + A_{\phi-1} s^{(\phi-1)} + \cdots + A_0, A_i \in \mathbb{R}^{p \times n}, i = 0, 1, ..., \phi, \tag{3.16}$$

que possui a seguinte matriz coeficiente

$$A = \left[\begin{array}{cccc} A_\phi & A_{\phi-1} & A_{\phi-2} & \ldots \quad A_0 \end{array}\right].$$

Então a seguinte matriz de convolução pode ser formada a partir dos coeficientes da matriz $A(s)$:

$$C_A^l = \overbrace{\begin{pmatrix} A_\phi & 0 & \cdots & 0 \\ A_{\phi-1} & A_\phi & \cdots & 0 \\ \vdots & \vdots & \ddots & \cdots \\ A_0 & A_1 & \ddots & A_\phi \\ 0 & A_0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & A_1 \\ 0 & 0 & \cdots & A_0 \end{pmatrix}}^{l \text{ blocos colunas}}, \tag{3.17}$$

onde l depende do problema em questão. O objetivo deste algoritmo é montar a matriz dada na equação (3.17), onde o número de blocos colunas de $C_A(l)$ dependerá do comprimento desejado da matriz de convolução.

A função `convform`

A função Matlab desenvolvida para realizar o algoritmo descrito nesta seção é a `convform.m` (apêndice A.5). Para usá-la o usuário entra com a matriz coeficiente da matriz com que deve ser formada a matriz de convolução dada na equação (3.17), o comprimento da matriz de convolução (número de blocos colunas) e o número de colunas (caso a matriz não for quadrada). A função retorna uma matriz de convolução associada à matriz de coeficiente. Essa função tem a seguinte sintaxe:

$$CA = \texttt{convform}(A, l, n),$$

onde A é a matriz coeficiente,l o número de blocos colunas desejado da matriz de convolução e n o número de colunas de $A(s)$. Se a matriz $A(s)$ for quadrada não há a necessidade do 3º argumento.

Exemplo 3.5 *Seja a matriz polinomial*

$$A(s) = \begin{bmatrix} 5s+3 & -3s+1 & 2s+1 \\ -3 & -2 & 5s+1 \\ 5s+4 & -s & 4s+5 \end{bmatrix}, \tag{3.18}$$

cuja matriz coeficiente dada por:

$$A = \left[\begin{array}{ccc|ccc} 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 5 & -3 & -2 & 1 \\ 5 & -1 & 4 & 4 & 0 & 5 \end{array}\right]. \tag{3.19}$$

Para se usar a função `convform` *no Matlab, e supondo que se deseja uma matriz de comprimento igual a 1, procede-se da seguinte forma:*

$>> A = [5 \quad -3 \quad 2 \quad 3 \quad 1 \quad 1; \quad 0 \quad 0 \quad 5 \quad -3 \quad -2 \quad 1; \quad 5 \quad -1 \quad 4 \quad 4 \quad 0 \quad 5];$

$>> CA = \texttt{convform}(A, 1, 3)$

ou

$>> CA = \texttt{convform}(A, 1)$

Ao final, obtém-se:

$$C_A = \left[\begin{array}{ccc} 5 & -3 & 2 \\ 0 & 0 & 5 \\ 5 & -1 & 4 \\ \hline 3 & 1 & 1 \\ -3 & -2 & 1 \\ 4 & 0 & 5 \end{array}\right].$$

Caso o comprimento desejado para a matriz de convolução seja 2, deve-se proceder da seguinte forma:

$>> CB = \texttt{convform}(A, 2, 3)$

ou

$>> CB = \texttt{convform}(A, 2),$

obtendo-se:

$$C_B = \left[\begin{array}{ccc|ccc} 5 & -3 & 2 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ 5 & -1 & 4 & 0 & 0 & 0 \\ \hline 3 & 1 & 1 & 5 & -3 & 2 \\ -3 & -2 & 1 & 0 & 0 & 5 \\ 4 & 0 & 5 & 5 & -1 & 4 \\ \hline 0 & 0 & 0 & 3 & 1 & 1 \\ 0 & 0 & 0 & -3 & -2 & 1 \\ 0 & 0 & 0 & 4 & 0 & 5 \end{array}\right]$$

□

3.1.6 Diaglamb

Este algoritmo cria, a partir de uma matriz cujas linhas correspondem aos coeficientes de um vetor polinomial, uma outra matriz coeficiente representativa de uma matriz diagonal cujos elementos da diagonal são os polinômios cujos coeficientes são as linhas da matriz coeficiente.

Descrição do algoritmo do Diaglamb

Seja o vetor polinomial

$$\underline{n}(s) = \begin{bmatrix} n_{1_\nu}s^\nu + n_{1_{\nu-1}}s^{\nu-1} + \ldots + n_{1_0} \\ n_{2_\nu}s^\nu + n_{2_{\nu-1}}s^{\nu-1} + \ldots + n_{2_0} \\ \vdots \\ n_{p_\nu}s^\nu + n_{p_{\nu-1}}s^{\nu-1} + \ldots + n_{p_0} \end{bmatrix}, \tag{3.20}$$

com a seguinte matriz coeficiente:

$$N_{vec} = \begin{bmatrix} n_{1_\nu} & n_{1_{\nu-1}} & \ldots & n_{1_0} \\ n_{2_\nu} & n_{2_{\nu-1}} & \ldots & n_{2_0} \\ \vdots & \vdots & & \vdots \\ n_{p_\nu} & n_{p_{\nu-1}} & \ldots & n_{p_0} \end{bmatrix}. \tag{3.21}$$

A partir da matriz (3.21), após aplicação do algoritmo Diaglamb deve-se ter uma matriz polonomial de dimensão $p \times p$ com a seguinte matriz de coeficientes:

$$N_{diag} = \left[\begin{array}{cccc|cccc|c|cccc} n_{1_\nu} & 0 & \ldots & 0 & n_{1_{\nu-1}} & 0 & \ldots & 0 & \ldots & n_{1_0} & 0 & \ldots & 0 \\ 0 & n_{2_\nu} & \ldots & 0 & 0 & n_{2_{\nu-1}} & \ldots & 0 & \ldots & 0 & n_{2_0} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & n_{p_\nu} & 0 & 0 & \ldots & n_{p_{\nu-1}} & \ldots & 0 & 0 & \ldots & n_{p_0} \end{array}\right] \tag{3.22}$$

que representa a seguinte matriz polinomial:

$$N_{diag}(s) = \begin{bmatrix} n_{1_\nu}s^\nu + \ldots + n_{1_0} & 0 & \ldots & 0 \\ 0 & n_{2_\nu}s^\nu + \ldots + n_{2_0} & \ldots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \ldots & n_{p_\nu}s^\nu + \ldots + n_{p_0} \end{bmatrix} \tag{3.23}$$

A função `diaglamb`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é o `diaglamb.m` (apêndice A.6). Para usá-la o usuário entra com a matriz coeficiente cujas linhas são os coeficientes dos polinômios a serem inseridos na diagonal da matriz e a função retorna uma matriz coeficiente da correspondente matriz diagonal. Essa função tem a seguinte sintaxe:

$$Ndiag = \texttt{diaglamb}(Nvec),$$

onde $Nvec$ é a matriz coeficiente associada ao vetor polinomial $\underline{n}(s)$ e $Ndiag$ é a matriz coeficiente da matriz polinomial diagonal obtida.

Exemplo 3.6 *Como exemplo, seja o vetor polinomial*

$$\underline{n}(s) = \begin{bmatrix} 5s^2 - 3s + 2 \\ 5 \\ 5s^2 - s + 4 \\ 4s^2 + 5 \end{bmatrix}$$

cuja matriz coeficientes é dada por:

$$N_{vec} = \begin{bmatrix} 5 & -3 & 2 \\ 0 & 0 & 5 \\ 5 & -1 & 4 \\ 4 & 0 & 5 \end{bmatrix}. \tag{3.24}$$

Para se obter a matriz coeficiente da matriz polinomial diagonal utilizando a função `diaglamb` *no Matlab, procede-se da seguinte forma:*

$>> Nvec = [5 \quad -3 \quad 2; 0 \quad 0 \quad 5; 5 \quad -1 \quad 4; 4 \quad 0 \quad 5]$

$>> Ndiag =$ `diaglamb`$(Nvec)$

Obtém-se, portanto:

$$N_{diag} = \left[\begin{array}{cccc|cccc|cccc} 5 & 0 & 0 & 0 & -3 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 5 \end{array}\right], \tag{3.25}$$

que representa a matriz dos coeficientes da seguinte matriz polinomial

$$N_{diag}(s) = \begin{bmatrix} 5s^2 - 3s + 2 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5s^2 - s + 4 & 0 \\ 0 & 0 & 0 & 4s^2 + 5 \end{bmatrix}.$$

□

3.1.7 Diagpol

Este algoritmo retorna a matriz coeficiente do vetor formado pelos polinômios da diagonal de uma matriz polinomial quadrada.

Descrição do algoritmo Diagpol

Seja a matriz

$$N(s) = \left[\begin{array}{c|c|c} n_{11_\nu}s^\nu + n_{11_{\nu-1}}s^{\nu-1} + \ldots + n_{11_0} & \ldots & n_{1m_\nu}s^\nu + n_{1m_{\nu-1}}s^{\nu-1} + \ldots + n_{1m_0} \\ n_{21_\nu}s^\nu + n_{21_{\nu-1}}s^{\nu-1} + \ldots + n_{21_0} & \ldots & n_{2m_\nu}s^\nu + n_{2m_{\nu-1}}s^{\nu-1} + \ldots + n_{2m_0} \\ \vdots & & \vdots \\ n_{p1_\nu}s^\nu + n_{p1_{\nu-1}}s^{\nu-1} + \ldots + n_{p1_0} & \ldots & n_{pm_\nu}s^\nu + n_{pm_{\nu-1}}s^{\nu-1} + \ldots + n_{pm_0} \end{array}\right], \tag{3.26}$$

que possui a seguinte matriz coeficiente

$$N = \left[\begin{array}{ccc|ccc|c|ccc} n_{11_\nu} & \dots & n_{1m_\nu} & n_{11_{\nu-1}} & \dots & n_{1m_{\nu-1}} & \dots & n_{11_0} & \dots & n_{1m_0} \\ n_{21_\nu} & \dots & n_{2m_\nu} & n_{22_{\nu-1}} & \dots & n_{2m_{\nu-1}} & \dots & n_{21_0} & \dots & n_{2m_0} \\ \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ n_{m1_\nu} & \dots & n_{mm_\nu} & n_{m1_{\nu-1}} & \dots & n_{mm_{\nu-1}} & \dots & n_{m1_0} & \dots & n_{mm_0} \end{array}\right], \quad (3.27)$$

com dimensão igual a $N \in \mathbb{R}^{m \times m(\nu+1)}$. Com a utilização do algoritmo Diagpol, obtém-se a seguinte matriz de coeficientes

$$N_{diag} = \left[\begin{array}{ccc} n_{11_\nu} & \dots & n_{1m_\nu} \\ n_{22_{\nu-1}} & \dots & n_{22_{\nu-1}} \\ \vdots & & \vdots \\ n_{mm_0} & \dots & n_{mm_0} \end{array}\right], \quad (3.28)$$

correspondente ao vetor polinomial

$$\underline{n}_{diag}(s) = \left[\begin{array}{c} n_{11_\nu} s^\nu + \ldots + n_{1m_0} \\ n_{22_\nu} s^\nu + \ldots + n_{22_0} \\ \vdots \\ n_{mm_\nu} s^\nu + \ldots + n_{mm_0} \end{array}\right].$$

Para realização deste algoritmo deve-se seguir os seguintes passos:

Algoritmo 3.2 *:*

Passo 1: *Coloque a matriz de coeficientes na forma vetorial, ou seja, execute o algoritmo Matsplit (seção 3.1.1) e com o formato mostrado na equação (3.3), obtendo-se a matriz* N_{vec}*;*

Passo 2: *Selecione as* $\{(k-1)m + k, k = 1, \ldots, m\}$ *linhas de* N_{vec_i}*, onde* m *é o número de linhas da matriz* $N(s)$*.* □

A função `diagpol`

A função Matlab desenvolvida para realizar o algoritmo descrito nesta seção é a `diagpol.m` (apêndice A.7). Para usá-la o usuário entra com a matriz coeficiente associada a uma matriz polinomial e a função retorna uma matriz coeficiente, cujas linhas são os coeficientes dos polinômios da diagonal da matriz polinomial. Essa função tem a seguinte sintaxe:

$$Ndiag = \texttt{diagpol}(N),$$

onde N é a matriz de coeficientes associada a uma matriz polinomial $N(s)$.

Exemplo 3.7 *Para mostrar o funcionamento do algoritmo, seja a matriz polinomial*

$$N(s) = \begin{bmatrix} s^3+s^2+5s+3 & -s^2-3s+1 & 2s^4+s^3+2s+1 \\ -3 & -2 & s^2+5s+1 \\ s^3+5s+4 & -s^2 & 2s^4+s^3+3s^2+4s+5 \end{bmatrix},$$

que possui a seguinte matriz coeficiente:

$$N = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 2 & 1 & 0 & 1 & 0 & -1 & 3 & 5 & 0 & 4 & 4 & 0 & 5 \end{array}\right].$$

Para se usar a função `diagpol` *no Matlab, procede-se da seguinte forma:*

```
>> N = [0  0  2  1  0  1  1  -1  0  5  -3  2   3   1  1;
        0  0  0  0  0  0  0   0  1  0   0  5  -3  -2  1;
        0  0  2  1  0  1  0  -1  3  5   0  4   4   0  5]
>> Ndiag = diagpol(N);
```

O resultado será:

$$N_{diag} = \begin{bmatrix} 0 & 1 & 1 & 5 & 3 \\ 0 & 0 & 0 & 0 & -2 \\ 2 & 1 & 3 & 4 & 5 \end{bmatrix},$$

que corresponde ao vetor polinomial

$$\underline{n}_{diag}(s) = \begin{bmatrix} s^3+s^2+5s+3 \\ -2 \\ 2s^4+s^3+3s^2+4s+5 \end{bmatrix}$$

formado pelos polinômios da diagonal da matriz $N(s)$. □

3.1.8 Polymatval

Neste algoritmo, dada uma matriz $N(s) \in \mathbb{R}^{p\times m}[s]$ e um valor $s_0 \in \mathbb{C}$, calcula-se $N(s_0)$.

Descrição do algoritmo Polymatval

Seja a matriz $N(s) \in \mathbb{R}^{p\times m}[s]$ com os graus máximos dos elementos polinomiais, por facilidade, todos iguais a ν.

$$N(s) = \left[\begin{array}{c|c|c} n_{11_\nu}s^\nu + n_{11_{\nu-1}}s^{\nu-1} + \ldots + n_{11_0} & \ldots & n_{1m_\nu}s^\nu + n_{1m_{\nu-1}}s^{\nu-1} + \ldots + n_{1m_0} \\ n_{21_\nu}s^\nu + n_{21_{\nu-1}}s^{\nu-1} + \ldots + n_{21_0} & \ldots & n_{2m_\nu}s^\nu + n_{2m_{\nu-1}}s^{\nu-1} + \ldots + n_{2m_0} \\ \vdots & & \vdots \\ n_{p1_\nu}s^\nu + n_{p1_{\nu-1}}s^{\nu-1} + \ldots + n_{p1_0} & \ldots & n_{pm_\nu}s^\nu + n_{pm_{\nu-1}}s^{\nu-1} + \ldots + n_{pm_0} \end{array}\right] \quad (3.29)$$

cuja matriz de coeficientes é dada por:

$$N = \left[\begin{array}{ccc|ccc|c|ccc} n_{11_\nu} & \dots & n_{1m_\nu} & n_{11_{\nu-1}} & \dots & n_{1m_{\nu-1}} & \dots & n_{11_0} & \dots & n_{1m_0} \\ n_{21_\nu} & \dots & n_{2m_\nu} & n_{21_{\nu-1}} & \dots & n_{2m_{\nu-1}} & \dots & n_{21_0} & \dots & n_{2m_0} \\ \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ n_{p1_\nu} & \dots & n_{pm_\nu} & n_{p1_{\nu-1}} & \dots & n_{pm_{\nu-1}} & \dots & n_{p1_0} & \dots & n_{pm_0} \end{array}\right], \qquad (3.30)$$

com dimensão igual a $N \in \mathbb{R}^{p \times m(\nu+1)}$. Supondo-se que se deseja calcular $N(s_k)$ onde $s_k \in \mathbb{C}$, deve-se fazer a seguinte multiplicação matricial

$$N(s_k) = N.N_{pot}(s_k)$$

onde $N_{pot}(s_k)$ é a seguinte matriz formada pela substituição da matriz de potências da variável s pelo valor numérico s_k, sendo dado por:

$$N_{pot}(s_k) = \overbrace{\left[\begin{array}{cccc} {s_k}^\nu & 0 & \dots & 0 \\ 0 & {s_k}^\nu & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & {s_k}^\nu \\ \hline {s_k}^{(\nu-1)} & 0 & \dots & 0 \\ 0 & {s_k}^{(\nu-1)} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & {s_k}^{(\nu-1)} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \hline 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{array}\right]}^{m(\nu+1)\times m} \qquad (3.31)$$

A função `polymatval`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `polymatval.m` (apêndice A.8). Para usá-la, o usuário deve fornecer a matriz de coeficientes associada a um vetor polinomial, o valor numérico que substituirá a variável polinomial e o número de colunas de $N(s)$ (caso a matriz não for quadrada). A função retorna uma matriz $N(s_k) \in \mathbb{C}^{p \times m}$. Essa função tem a seguinte sintaxe:

$$Nsk = \texttt{polymatval}(N, sk, m),$$

onde N é a matriz coeficiente associada à uma matriz polinomial $N(s)$, s_k é o valor numérico que será substituído em $N(s)$, m é o número de colunas de $N(s)$ e Nsk é o resultado da substituição $N(s_k)$. Se a matriz for quadrada não é necessário fornecer o valor de m

Exemplo 3.8 *Para mostrar o funcionamento do algoritmo, seja a matriz polinomial*

$$N(s) = \begin{bmatrix} s^3 + s^2 + 5s + 3 & -s^2 - 3s + 1 & 2s^4 + s^3 + 2s + 1 \\ -3 & -2 & s^2 + 5s + 1 \\ s^3 + 5s + 4 & -s^2 & 2s^4 + s^3 + 3s^2 + 4s + 5 \end{bmatrix}, \tag{3.32}$$

cuja matriz de coeficientes é dada por:

$$N = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 2 & 1 & 0 & 1 & 0 & -1 & 3 & 5 & 0 & 4 & 4 & 0 & 5 \end{array}\right] \tag{3.33}$$

Supondo que se queira calcular $N(s_k)$ no Matlab para $s_k = 2$, deve-se proceder da seguinte forma:

```
>> N = [0  0  2  1  0  1  1  -1  0  5  -3  2   3   1  1;
        0  0  0  0  0  0  0   0  1  0   0  5  -3  -2  1;
        0  0  2  1  0  1  0  -1  3  5   0  4   4   0  5]
>> Nsk = polymatval(N, 2, 3);
```

ou

```
>> Nsk = polymatval(N, 2);
```

Ao final, obtém-se:

$$N_{sk} = N.N_{pot}(2) = N. \left[\begin{array}{ccc} 16 & 0 & 0 \\ 0 & 16 & 0 \\ 0 & 0 & 16 \\ \hline 8 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 8 \\ \hline 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \\ \hline 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \\ \hline 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc} 25 & -9 & 45 \\ -3 & -2 & 15 \\ 22 & -4 & 65 \end{array}\right]$$

□

3.1.9 Coldeg

Este algoritmo resulta em um vetor linha com os graus das colunas de uma matriz polinomial.

Descrição do algoritmo Coldeg

Seja a matriz polinomial

$$N(s) = \left[\begin{array}{cccc} \underline{n}_1(s) & \underline{n}_2(s) & \dots & \underline{n}_p(s) \end{array}\right]$$

onde $gr[n_i(s)] = \nu_i, i = 1, 2, \dots, p$. Após a utilização do algoritmo Coldeg deve-se obter:

$$GrauCol = \left[\begin{array}{cccc} \nu_1 & \nu_2 & \dots & \nu_p \end{array}\right]$$

Os graus das colunas de uma matriz polinomial podem ser calculados da seguinte forma.

Algoritmo 3.3 *Suponha que* $N(s) \in \mathbb{R}^{p\times m}(s)$*:*

Passo 1: *Coloque a matriz coeficiente de* $N(s)$ *na forma vetorial* N_{vec}*, executando o algoritmo Matsplit.*

Passo 2: *Faça* $k = 1$ *e* $n_{col} = 1$

Passo 3: *Selecione as linhas* $p(k-1)+1$ *a* pk *de* N_{vec} *e armazene numa matriz* $N_{vec}^{(k)}$

Passo 4: *Execute o algoritmo Cutzeros, para retirar os zeros em excesso a esquerda de* $N_{vec}^{(k)}$

Passo 5: *Conte o número de colunas que permaneceram em* $N_{vec}^{(k)}$ *e armazene e* n_k *e defina* $g_{col,k} = n_k - 1$*, para a* k*-ésima componente do vetor* $\underline{g}_{col}$*.*

Passo 6: *Faça* $ncol = ncol + 1$*. Se* $ncol > m$ *o algoritmo pára; caso contrário, faça* $k = k + 1$ *e volte o* ***passo 3****.* □

A função `coldeg`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `coldeg.m` (apêndice A.9). Para usá-la, o usuário deve fornecer a matriz coeficiente associada a um vetorde uma matriz polinomial e o número de colunas (caso a matriz não seja quadrada). A função retorna um vetor com os graus das colunas de $N(s)$. Essa função tem a seguinte sintaxe:

$$GrauCol = \texttt{coldeg}(N, m),$$

onde N é a matriz de coeficientes associada a uma matriz polinomial $N(s)$ e m o número de colunas de $N(s)$.

Exemplo 3.9 *Como exemplo de aplicação do algoritmo, considere a matriz*

$$N(s) = \begin{bmatrix} s^3+s^2+5s+3 & -s^2-3s+1 & 2s^4+s^3+2s+1 \\ -3 & -2 & s^2+5s+1 \\ s^3+5s+4 & -s^2 & 2s^4+s^3+3s^2+4s+5 \end{bmatrix}, \tag{3.34}$$

cuja matriz de coeficientes é dada por:

$$N = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 2 & 1 & 0 & 1 & 0 & -1 & 3 & 5 & 0 & 4 & 4 & 0 & 5 \end{array}\right]. \tag{3.35}$$

Para utilizar a função `coldeg` *desenvolvida em linguagem Matlab, procede-se da seguinte forma:*

$$>> N = [0\ \ 0\ \ 2\ \ 1\ \ 0\ \ 1\ \ 1\ \ -1\ \ 0\ \ 5\ \ -3\ \ 2\ \ 3\ \ 1\ \ 1;$$
$$0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 5\ \ -3\ \ -2\ \ 1;$$
$$0\ \ 0\ \ 2\ \ 1\ \ 0\ \ 1\ \ 0\ \ -1\ \ 3\ \ 5\ \ 0\ \ 4\ \ 4\ \ 0\ \ 5]$$

$>> GrauCol = \texttt{coldeg}(N, 3);$

ou

$>> GrauCol = \texttt{coldeg}(N);$

Executando o Passo 1 do algoritmo, será obtida a seguinte matriz:

$$N_{vec} = \left[\begin{array}{rrrrr} 0 & 1 & 1 & 5 & 3 \\ 0 & 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 5 & 4 \\ \hline 0 & 0 & -1 & -3 & 1 \\ 0 & 0 & 0 & 0 & -2 \\ 0 & 0 & -1 & 0 & 0 \\ \hline 2 & 1 & 0 & 2 & 1 \\ 0 & 0 & 1 & 5 & 1 \\ 2 & 1 & 3 & 4 & 5 \end{array}\right]. \tag{3.36}$$

De acordo com o Passo 3 obtém-se

$$N_{vec} = \left[\begin{array}{rrrrr} 0 & 1 & 1 & 5 & 3 \\ 0 & 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 5 & 4 \end{array}\right],$$

onde são mostrados os coeficientes da primeira coluna de $N(s)$. Nesta matriz, os elementos da coluna mais à esquerda são zeros, mostrando que o grau desta coluna é inferior ao grau da matriz completa. Portanto, na execução do Passo 4, é obtida a seguinte matriz

$$N_{vec} = \left[\begin{array}{rrrr} 1 & 1 & 5 & 3 \\ 0 & 0 & 0 & -3 \\ 1 & 0 & 5 & 4 \end{array}\right].$$

Com a execução do Passo 5, obtém-se.

$$GrauCol = [3]$$

Repetindo os passos do algoritmo mais duas vezes, obtém-se:

$$GrauCol = \left[\begin{array}{ccc} 3 & 2 & 4 \end{array}\right]$$

□

3.2 Operações básicas com matrizes polinomiais

Nesta seção serão apresentados algoritmos para a realização das operações básicas de soma e multiplicação de matrizes polinomiais.

3.2.1 Soma/Subtração

Este algoritmo resulta, inicialmente, em uma matriz coeficiente que é a soma de outras duas matrizes de dimensão $p \times m(\nu+1)$, onde ν denota o maior grau entre as matrizes cuja soma se deseja realizar. Em seguida deve-se utilizar algoritmo Cutzeros para retirar as submatrizes da matriz coeficiente que podem ser consideradas nulas e, portanto, reduzir o grau da matriz soma.

Descrição do algoritmo Matsum

Para realização deste algoritmo deve-se seguir os seguintes passos:

Algoritmo 3.4 *:*

Passo 1: *Inserir zeros na matriz de coeficientes de menor número de colunas até que ambas as matrizes de coeficientes possuam o mesmo número de colunas;*

Passo 2: *Some as matrizes;*

Passo 3: *Use o algoritmo Cutzeros para reduzir o grau da matriz soma.* □

Observação 3.1 *A subtração das matrizes $A(s)$ e $B(s)$ é feita somando-se $A(s)$ com $-B(s)$.* □

A função `matsum`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `matsum.m` (apêndice A.10). Para usá-la, o usuário entra com as duas matrizes coeficientes associadas às matrizes a serem somadas e a função retorna uma matriz coeficiente que é a soma das matrizes de entrada. Essa função tem a seguinte sintaxe:

$$C = \texttt{matsum}(N, D),$$

onde N e D são matrizes coeficientes associadas às matrizes $N(s)$ e $D(s)$, respectivamente.

Exemplo 3.10 *Como exemplo de aplicação do algoritmo, considere as matrizes:*

$$N(s) = \begin{bmatrix} s^3+s^2+5s+3 & -s^2-3s+1 & 2s^4+s^3+2s+1 \\ -3 & -2 & s^2+5s+1 \\ s^3+5s+4 & -s^2 & 2s^4+s^3+3s^2+4s+5 \end{bmatrix}$$

e

$$D(s) = \begin{bmatrix} 5s^2+2s+3 & 3s+1 & 2s^3 \\ -4 & -2s & 5s+1 \\ s^2 & -s^2 & 2s^3+4s+5 \end{bmatrix},$$

cujas matrizes coeficientes são dadas por:

$$N = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 2 & 1 & 0 & 1 & 0 & -1 & 3 & 5 & 0 & 4 & 4 & 0 & 5 \end{array}\right] \tag{3.37}$$

e

$$D = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 5 & 0 & 0 & 2 & 3 & 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 5 & -4 & 0 & 1 \\ 0 & 0 & 2 & 1 & -1 & 0 & 0 & 0 & 4 & 0 & 0 & 5 \end{array}\right]. \tag{3.38}$$

Para se obter a matriz $C(s) = N(s) + D(s)$ utilizando a função `matsum` *no Matlab, procede-se da seguinte forma:*

```
>> N = [0  0  2  1  0  1  1  -1  0  5  -3  2   3   1  1;
        0  0  0  0  0  0  0   0  1  0   0  5  -3  -2  1;
        0  0  2  1  0  1  0  -1  3  5   0  4   4   0  5]
>> D = [0  0  2  5   0  0  2   3  0   3  1  0;
        0  0  0  0   0  0  0  -2  5  -4  0  1;
        0  0  2  1  -1  0  0   0  4   0  0  5]
>> C = matsum(N, D)
```

Note que para a realização da soma, de acordo com o Passo 1 do algoritmo, deverão ser incluídos zeros à esquerda da matriz D, que ficará da seguinte forma:

$$D = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0 & 2 & 5 & 0 & 0 & 2 & 3 & 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 5 & -4 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & -1 & 0 & 0 & 0 & 4 & 0 & 0 & 5 \end{array}\right] \tag{3.39}$$

Executando o Passo 2 do algoritmo, ou seja, somando-se as matrizes (3.37) e (3.39), resulta:

$$C = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 3 & 6 & -1 & 0 & 7 & 0 & 2 & 6 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -2 & 10 & -7 & -2 & 2 \\ 0 & 0 & 2 & 1 & 0 & 3 & 1 & -2 & 3 & 5 & 0 & 8 & 4 & 0 & 10 \end{array}\right] \tag{3.40}$$

□

3.2.2 Multiplicação

Uma das operações básicas da matemática é a multiplicação. Para sua realização para o caso de matrizes polinomiais, isto não é tão óbvio. Para tanto, faz-se uso do conceito de matriz de convolução ou Sylvester, apresentado na seção 2.2. Entretanto, a estrutura da matriz de Sylvester mostra, em sua formação que haverá multiplicações com vários blocos de zeros, aumentando, com isso, o custo computacional desta operação. Assim, o algoritmo desenvolvido procura otimizar estas operações.

Descrição do algoritmo Convmat

Suponha que se necessite obter:

$$G(s) = B(s)A(s),$$

onde

$$A(s) = A_\alpha s^\alpha + A_1 s^{(\alpha-1)} + \cdots + A_0, A_i \in \mathbb{R}^{n\times m}, i = 0, 1, ..., \alpha \tag{3.41}$$

e

$$B(s) = B_\beta s^\beta + B_{\beta-1} s^{(\beta-1)} + \cdots + B_0, B_i \in \mathbb{R}^{p\times n}, i = 0, 1, ..., \beta. \tag{3.42}$$

Note que a operação de multiplicação usando o conceito das matrizes de Sylvester pode ser feita da seguinte forma:

$$G = \begin{bmatrix} G_{\beta+\alpha} \\ G_{\beta+\alpha-1} \\ \vdots \\ G_0 \end{bmatrix} = \overbrace{\begin{pmatrix} B_\beta & 0 & \cdots & 0 \\ B_{\beta-1} & B_\beta & \cdots & 0 \\ \vdots & \vdots & \ddots & \cdots \\ B_0 & B_1 & \ddots & B_\beta \\ 0 & B_0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & B_1 \\ 0 & 0 & \cdots & B_0 \end{pmatrix}}^{\text{Comprimento } \alpha+1 \text{ } blocos} \begin{pmatrix} A_\alpha \\ A_{\alpha-1} \\ \vdots \\ A_0 \end{pmatrix} =$$

$$= \left[\begin{array}{c} B_\beta A_\alpha \\ B_{\beta-1}A_\alpha \\ \vdots \\ \hline B_0 A_\alpha \\ \hline 0A_\alpha \\ \vdots \\ 0A_\alpha \end{array}\right] + \left[\begin{array}{c} 0A_{\alpha-1} \\ B_\beta A_{\alpha-1} \\ \vdots \\ B_1 A_{\alpha-1} \\ \hline B_0 A_{\alpha-1} \\ \hline \vdots \\ 0A_{\alpha-1} \end{array}\right] + \ldots + \left[\begin{array}{c} 0A_0 \\ 0A_0 \\ \vdots \\ B_\beta A_0 \\ B_{\beta-1}A_0 \\ \vdots \\ B_0 A_0 \end{array}\right] \tag{3.43}$$

Observe que cada vetor da soma possui nos seus blocos matriciais finais multiplicações com blocos de zeros. Assim a equação acima pode ser reescrita como:

$$G = \left[\begin{array}{c} B_\beta \\ B_{\beta-1} \\ \vdots \\ \hline B_0 \\ \hline \\ 0 \\ \\ \end{array}\right] A_\alpha + \left[\begin{array}{c} 0 \\ \hline B_\beta \\ B_{\beta-1} \\ \vdots \\ \hline B_0 \\ \hline \\ 0 \end{array}\right] A_{\alpha-1} + \ldots + \left[\begin{array}{c} \\ 0 \\ \hline \\ B_\beta \\ B_{\beta-1} \\ \vdots \\ B_0 \end{array}\right] A_0$$

Note, portanto, que é necessário, apenas fazer a multiplicação da matriz

$$\overline{B} = \left[\begin{array}{c} B_\beta \\ B_{\beta-1} \\ \vdots \\ B_0 \end{array}\right]$$

pelas matrizes dos coeficientes de $A(s)$ $(A_i,\ i = \alpha, \alpha-1, \ldots, 0)$ e em seguida acrescentar matrizes nulas no topo e em baixo para, então, efetuar as somas. O desenvolvimento acima pode ser resumido no seguinte algoritmo.

Algoritmo 3.5 *Sejam $B(s) \in \mathbb{R}^{p\times m}[s]$ e $A(s) \in \mathbb{R}^{m\times q}[s]$ de graus iguais a β e α, respectivamente.*

Passo 1: *Forme a matriz*

$$\overline{B} = \left[\begin{array}{cccc} B_\beta^T & B_{\beta-1}^T & \ldots & B_0^T \end{array}\right]^T$$

Passo 2: *Para k variando de 1 a $\alpha+1$, fazer*

$$G_k = \left[\begin{array}{c} 0_{(k-1)p\times q} \\ \hline \\ \overline{B}A_{(\alpha-k+1)} \\ \hline \\ 0_{(\alpha-k+1)p\times q} \end{array}\right]$$

Passo 3: *Calcular*

$$G = \sum_{k=1}^{\alpha+1} G_k$$

A função `convmat`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `convmat.m` (apêndice A.11). Para usá-la, o usuário deve entrar com as duas matrizes coeficientes que devem ser multiplicadas e o número de colunas da segunda matriz do argumento (caso ela não seja quadrada).A função retorna a matriz coeficiente associada à matriz polinomial resultante da multiplicação das matrizes de entrada. Essa função tem a seguinte sintaxe:

$$G = \texttt{convmat}(B, A, m),$$

onde B e A são matrizes coeficientes associadas às matrizes $B(s)$ e $A(s)$, respectivamente, e m é o número de colunas de $A(s)$

Exemplo 3.11 *Como exemplo de aplicação do algoritmo, considere a multiplicação das seguintes matrizes:*

$$A(s) = \begin{bmatrix} s^3+s^2+5s+3 & -s^2-3s+1 & 2s^4+s^3+2s+1 \\ -3 & -2 & s^2+5s+1 \\ s^3+5s+4 & -s^2 & 2s^4+s^3+3s^2+4s+5 \end{bmatrix}$$

e

$$B(s) = \begin{bmatrix} 5s^2+2s+3 & 3s+1 & 2s^3 \\ -4 & -2s & 5s+1 \\ s^2 & -s^2 & 2s^3+4s+5 \end{bmatrix},$$

cujas matrizes coeficientes são dadas por:

$$A = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 2 & 1 & 0 & 1 & 0 & -1 & 3 & 5 & 0 & 4 & 4 & 0 & 5 \end{array}\right] \tag{3.44}$$

e

$$B = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 5 & 0 & 0 & 2 & 3 & 0 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 5 & -4 & 0 & 1 \\ 0 & 0 & 2 & 1 & -1 & 0 & 0 & 0 & 4 & 0 & 0 & 5 \end{array}\right] \tag{3.45}$$

Para utilizar a função `convmat` *no Matlab, procede-se da seguinte forma:*

$$>> B = [0\ \ 0\ \ 2\ \ 1\ \ 0\ \ 1\ \ 1\ \ -1\ \ 0\ \ 5\ \ -3\ \ 2\ \ \ 3\ \ \ 1\ \ 1;$$
$$0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ \ 0\ \ 1\ \ 0\ \ \ 0\ \ 5\ \ -3\ \ -2\ \ 1;$$
$$0\ \ 0\ \ 2\ \ 1\ \ 0\ \ 1\ \ 0\ \ -1\ \ 3\ \ 5\ \ \ 0\ \ 4\ \ \ 4\ \ \ 0\ \ 5]$$
$$>> A = [0\ \ 0\ \ 2\ \ 5\ \ \ 0\ \ 0\ \ 2\ \ \ 3\ \ 0\ \ \ 3\ \ 1\ \ 0;$$
$$0\ \ 0\ \ 0\ \ 0\ \ \ 0\ \ 0\ \ 0\ \ -2\ \ 5\ \ -4\ \ 0\ \ 1;$$
$$0\ \ 0\ \ 2\ \ 1\ \ -1\ \ 0\ \ 0\ \ \ 0\ \ 4\ \ \ 0\ \ 0\ \ 5]$$
$$>> G = \textit{convmat}(B, A)$$

ou

$$>> G = \textit{convmat}(B, A, 3)$$

obtendo-se:

$$G = \left[\begin{array}{ccc|ccc|ccc|ccc|ccc} 0 & 0 & 4 & 2 & -2 & 4 & 6 & -1 & 10 & 7 & 3 & 28 & 32 & 4 & 8 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & -1 & 10 & 5 & -5 & 0 \\ 0 & 0 & 4 & 2 & -2 & 4 & 6 & -1 & 14 & 5 & 0 & 32 & 32 & -1 & 30 \end{array}\right.$$

$$\left.\begin{array}{ccc|ccc|ccc} 33 & 21 & -8 & 33 & 12 & 16 & 5 & 3 & 6 \\ -14 & -1 & 25 & -6 & -5 & 19 & -1 & -3 & 3 \\ 39 & 10 & 30 & 23 & 17 & 40 & 12 & 4 & 25 \end{array}\right]$$

□

3.3 Operações Básicas envolvendo a uma única matriz polinomial

Nesta seção serão apresentados alguns algoritmos para as seguintes operações básicas de matrizes polinomiais: transposta, conjugado complexo e o traço.

3.3.1 Transposta

Este algoritmo foi criado para se obter a transposta de uma matriz polinomial.

Descrição do algoritmo Transpol

A transposta de uma matriz polinomial

$$N(s) = \left[\begin{array}{c|c|c} n_{11_\nu}s^\nu + n_{11_{\nu-1}}s^{\nu-1} + \ldots + n_{11_0} & \ldots & n_{1m_\nu}s^\nu + n_{1m_{\nu-1}}s^{\nu-1} + \ldots + n_{1m_0} \\ n_{21_\nu}s^\nu + n_{21_{\nu-1}}s^{\nu-1} + \ldots + n_{21_0} & \ldots & n_{2m_\nu}s^\nu + n_{2m_{\nu-1}}s^{\nu-1} + \ldots + n_{2m_0} \\ \vdots & & \vdots \\ n_{p1_\nu}s^\nu + n_{p1_{\nu-1}}s^{\nu-1} + \ldots + n_{p1_0} & \ldots & n_{pm_\nu}s^\nu + n_{pm_{\nu-1}}s^{\nu-1} + \ldots + n_{pm_0} \end{array}\right] \tag{3.46}$$

é dada por:

$$N^T(s) = \left[\begin{array}{c|c|c} n_{11_\nu}s^\nu + \ldots + n_{11_0} & \ldots & n_{p1_\nu}s^\nu + \ldots + n_{p1_0} \\ n_{12_\nu}s^\nu + \ldots + n_{12_0} & \ldots & n_{p2_\nu}s^\nu + \ldots + n_{p2_0} \\ \vdots & & \vdots \\ n_{1m_\nu}s^\nu + \ldots + n_{1m_0} & \ldots & n_{pm_\nu}s^\nu + \ldots + n_{pm_0} \end{array}\right]. \tag{3.47}$$

Note que, como a matriz coeficiente de $N(s)$ é

$$N = \left[\begin{array}{cccc} N_\nu & N_{\nu-1} & \ldots & N_0 \end{array}\right],$$

então a matriz coeficiente da transposta de $N(s)$ (N_t) pode ser obtida simplesmente transpondo-se cada matriz de coeficientes de N. Portanto:

$$N_t = \left[\begin{array}{cccc} N_\nu^T & N_{\nu-1}^T & \ldots & N_o^T \end{array}\right].$$

A função `transpol`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `transpol.m` (apêndice A.12). Para usá-la, o usuário entra com a matriz coeficiente associada a uma matriz polinomial e o número de colunas (caso a matriz não seja quadrada) e a função retorna uma matriz coeficiente associada à transposta de $N(s)$. Essa função tem a seguinte sintaxe:

$$Nt = \texttt{transpol}(N, m),$$

onde N é a matriz coeficiente associada a uma matriz polinomial $N(s)$,m denota o número de colunas de $N(s)$ e Nt é a matriz de coeficientes da transposta de $N(s)$.

Exemplo 3.12 *Como exemplo de aplicação do algoritmo, considere a matriz*

$$N(s) = \left[\begin{array}{ccc} s^3 + s^2 + 5s + 3 & -s^2 - 3s + 1 & 2s^4 + s^3 + 2s + 1 \\ -3 & -2 & s^2 + 5s + 1 \\ s^3 + 5s + 4 & -s^2 & 2s^4 + s^3 + 3s^2 + 4s + 5 \end{array}\right],$$

cuja matriz coeficiente é dada por:

$$N = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 2 & 1 & 0 & 1 & 0 & -1 & 3 & 5 & 0 & 4 & 4 & 0 & 5 \end{array}\right].$$

Para utilizar a função `transpol` *no Matlab, procede-se da seguinte forma:*

$$\begin{array}{l} >> N = [0\ \ 0\ \ 2\ \ 1\ \ 0\ \ 1\ \ 1\ \ -1\ \ 0\ \ 5\ \ -3\ \ 2\ \ \ 3\ \ \ 1\ \ 1; \\ \qquad\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ \ 0\ \ 1\ \ 0\ \ \ 0\ \ 5\ \ -3\ \ -2\ \ 1; \\ \qquad\ \ 0\ \ 0\ \ 2\ \ 1\ \ 0\ \ 1\ \ 0\ \ -1\ \ 3\ \ 5\ \ \ 0\ \ 4\ \ \ 4\ \ \ 0\ \ 5] \\ >> Nt = \texttt{transpol}(N, 3); \end{array}$$

ou

$>> Nt =$ `transpol`(N);

A função `transpol` *retorna a matriz*

$$Nt = \left[\begin{array}{ccc|ccc|ccc|ccc|ccc} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 5 & 0 & 5 & 3 & -3 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & -3 & 0 & 0 & 1 & -2 & 0 \\ 2 & 0 & 2 & 1 & 0 & 1 & 0 & 1 & 3 & 2 & 5 & 4 & 1 & 1 & 5 \end{array}\right], \tag{3.48}$$

que representa a matriz coeficiente de $N^T(s)$*, onde:*

$$N^T(s) = \begin{bmatrix} s^3 + s^2 + 5s + 3 & -3 & s^3 + 5s + 4 \\ -s^2 - 3s + 1 & -2 & -s^2 \\ 2s^4 + s^3 + 2s + 1 & s^2 + 5s + 1 & 2s^4 + s^3 + 3s^2 + 4s + 5 \end{bmatrix}.$$

□

3.3.2 Conjugado complexo

Para a matriz $N(s) \in \mathbb{R}^{p\times m}[s]$, este algoritmo permite obter a matriz $\overline{N}(s) = N(-s)$. Note, portanto, que quando o expoente da potência de s for par, não haverá a alteração do sinal da matriz de coeficiente correspondente e quando o expoente da potência de s for ímpar, a matriz de coeficientes deverá ser multiplicada por -1.

Descrição do algoritmo do Conjpol

Seja a matriz polinomial

$$N(s) = N_\nu s^\nu + N_{\nu-1} s^{\nu-1} + \ldots + N_0,$$

cuja matriz coeficiente dada por:

$$N = \begin{bmatrix} N_\nu & N_{\nu-1} & \ldots & N_0 \end{bmatrix}$$

Então a matriz coeficiente de $\overline{N}(s) = N(-s)$ deve ser dada por:

$$\overline{N} = \begin{bmatrix} (-1)^\nu N_\nu & \ldots & N_2 & -N_1 & N_0 \end{bmatrix}.$$

Observação 3.2 *Em problemas de controle, em geral deseja-se calcular* $N^*(s) = N^T(-s)$. *Para tanto deve-se utilizar o algoritmo Conjpol e em seguida o algoritmo Transpol.*

A função `conjpol`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `conjpol.m` (apêndice A.13). Para usá-la, o usuário entra com a matriz de coeficientes associada a uma matriz polinomial e o número de colunas (caso a matriz não seja quadrada). A função retorna uma matriz de coeficientes que é o conjugado complexo de $N(s)$. Essa função tem a seguinte sintaxe:

$$Nconj = \texttt{conjpol}(N, m),$$

onde N é a matriz coeficiente associada a uma matriz polinomial $N(s)$, m número de colunas de $N(s)$ e $Nconj$ é a matriz coeficiente de $\overline{N}(s) = N(-s)$.

Exemplo 3.13 *Seja a matriz*

$$N(s) = \begin{bmatrix} s^3+s^2+5s+3 & -s^2-3s+1 & 2s^4+s^3+2s+1 \\ -3 & -2 & s^2+5s+1 \\ s^3+5s+4 & -s^2 & 2s^4+s^3+3s^2+4s+5 \end{bmatrix},$$

cuja matriz coeficiente é dada por

$$N = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 2 & 1 & 0 & 1 & 0 & -1 & 3 & 5 & 0 & 4 & 4 & 0 & 5 \end{array}\right].$$

Para calcular a matriz coeficiente de $\overline{N}(s) = N(-s)$ *utilizando a função* `conjpol` *no Matlab, procede-se da seguinte forma:*

```
>> N = [0  0  2  1  0  1  1  -1  0  5  -3  2   3   1  1;
        0  0  0  0  0  0  0   0  1  0   0  5  -3  -2  1;
        0  0  2  1  0  1  0  -1  3  5   0  4   4   0  5]
>> Nconj = conjpol(N, 3);
```

ou

```
>> Nconj = conjpol(N);
```

A função `conjpol` *retorna a seguinte matriz coeficiente:*

$$Nconj = \left[\begin{array}{ccc|ccc|ccc|ccc|ccc} 0 & 0 & 2 & -1 & 0 & -1 & 1 & -1 & 0 & -5 & +3 & -2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -5 & -3 & -2 & 1 \\ 0 & 0 & 2 & -1 & 0 & -1 & 0 & -1 & 3 & -5 & 0 & -4 & 4 & 0 & 5 \end{array}\right]$$

□

3.3.3 Traço

Este algoritmo permite obter o traço de uma matriz polinomial quadrada.

Descrição do algoritmo Traçopol

Para entender o algoritmo proposto para o cálculo do traço de uma matriz polinomial, seja:

$$N(s) = \left[\begin{array}{c|c|c} n_{11_\nu}s^\nu + n_{11_{\nu-1}}s^{\nu-1} + \ldots + n_{11_0} & \ldots & n_{1m_\nu}s^\nu + n_{1m_{\nu-1}}s^{\nu-1} + \ldots + n_{1m_0} \\ n_{21_\nu}s^\nu + n_{21_{\nu-1}}s^{\nu-1} + \ldots + n_{21_0} & \ldots & n_{2m_\nu}s^\nu + n_{2m_{\nu-1}}s^{\nu-1} + \ldots + n_{2m_0} \\ \vdots & & \vdots \\ n_{p1_\nu}s^\nu + n_{p1_{\nu-1}}s^{\nu-1} + \ldots + n_{p1_0} & \ldots & n_{pm_\nu}s^\nu + n_{pm_{\nu-1}}s^{\nu-1} + \ldots + n_{pm_0} \end{array}\right]. \tag{3.49}$$

Selecionando-se os polinômios da diagonal e empilhando-os para realização do somatório obtém-se:

$$\begin{array}{r} n_{11_\nu}s^\nu + n_{11_{\nu-1}}s^{\nu-1} + \ldots + n_{11_0} \\ n_{22_\nu}s^\nu + n_{22_{\nu-1}}s^{\nu-1} + \ldots + n_{22_0} \\ + \quad \vdots \\ \underline{n_{pp_\nu}s^\nu + n_{pp_{\nu-1}}s^{\nu-1} + \ldots + n_{pp_0}} \\ t_\nu s^\nu + t_{\nu-1}s^{\nu-1} + \ldots + t_0 \end{array}$$

onde

$$t_k = \sum_{i=1}^{p} n_{ii_k}.$$

Assim o cálculo do traço de uma matriz polinomial pode ser feito de acordo com o seguinte algoritmo:

Algoritmo 3.6 *:*

Passo 1: *Utilizar o algoritmo Diagpol para formar uma matriz com os coeficientes dos polinômios da diagonal da matriz.*

Passo 2: *Somar as linhas da matriz criada no* ***Passo 1****.* □

A função `tracopol`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `tracopol.m` (apêndice A.14). Para usá-la, o usuário fornece a matriz de coeficientes de uma matriz polinomial e a função retorna um vetor cujos elementos são os coeficientes do polinômio que representa o traço da matriz. Essa função tem a seguinte sintaxe:

$$tracN = \texttt{tracopol}(N),$$

onde N é a matriz coeficiente associada à matriz polinomial $N(s)$ e $tracN$ é um vetor com os coeficientes do polinômio que é o traço de $N(s)$.

Exemplo 3.14 *Como exemplo de aplicação do algoritmo, considere a matriz*

$$N(s) = \begin{bmatrix} s^3+s^2+5s+3 & -s^2-3s+1 & 2s^4+s^3+2s+1 \\ -3 & -2 & s^2+5s+1 \\ s^3+5s+4 & -s^2 & 2s^4+s^3+3s^2+4s+5 \end{bmatrix},$$

cuja matriz coeficiente é dada por:

$$N = \left[\begin{array}{ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 2 & 1 & 0 & 1 & 0 & -1 & 3 & 5 & 0 & 4 & 4 & 0 & 5 \end{array}\right].$$

Para se usar a função `tracopol`*, no Matlab procede-se da seguinte forma:*

```
>> N = [0  0  2  1  0  1  1  -1  0  5  -3  2   3   1  1;
        0  0  0  0  0  0  0   0  1  0   0  5  -3  -2  1;
        0  0  2  1  0  1  0  -1  3  5   0  4   4   0  5]
>> tracN = tracopol(N);
```

Executando o Passo 1 do algoritmo será obtida a seguinte matriz:

$$N_{diag} = \begin{bmatrix} 0 & 1 & 1 & 5 & 3 \\ 0 & 0 & 0 & 0 & -2 \\ 2 & 1 & 3 & 4 & 5 \end{bmatrix}.$$

Para finalizar, executando o Passo 2, resulta:

$$Ntrac = \begin{bmatrix} 2 & 2 & 4 & 9 & 6 \end{bmatrix}$$

□

3.4 Determinação do número de valores singulares nulos de uma matriz

Na seção 2.10 foi feita uma exposição sobre a robustez do DVS. Contudo, para sua utilização na determinação do posto de matrizes deve ser calculado o número valores singulares nulos ou mais formalmente abaixo de um valor de tolerância ξ. Isto pode ser feito a partir da determinação da posição do primeiro elemento do vetor de valores singulares menor que um valor de tolerância ξ. O algoritmo que faz esse cálculo é o NumsingNull.

Descrição do algoritmo NumsingNull

A idéia deste algoritmo é vasculhar num vetor de valores singulares, onde ocorre o salto de variação nos elemento deste vetor, considerando que quando é feito a DVS é verificado que os valores singulares σ_i são não negativos e ordenados de forma decrescente, ou seja,

$$\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n$$

Para realização deste algoritmo devem-se seguir os seguintes passos.

Algoritmo 3.7 *:*

Passo 1: *Atribua a NumSing o número de valores singulares do vetor, sendo que*

$$\underline{\sigma} = [\sigma_1, \ldots, \sigma_{NumSing}]$$

considerando que

$$\sigma_1 \geq \sigma_2 \geq \ldots \sigma_{NumSing},$$

e defina tol igual à tolerância que será considerada para definir se um valor singular poderá ser nulo, $NumSing0 = 0$ *e* $j = NumSing$

Passo 2: *Se* $\sigma_1/\sigma_{NumSing} > 10^7$*, o vetor* $\underline{\sigma_i}$ *deverá ser normalizado. Então, deverá ser feita a seguinte manipulação*

$$\underline{\sigma}_{normalizado} = \underline{\sigma}/\sigma_1,$$

caso contrário

$$\underline{\sigma}_{normalizado} = \underline{\sigma}$$

Passo 3: *Se* ${\sigma^j}_{normalizado} < tol$*, faça*

$$NumSing0 = NumSing0 + 1$$

caso contrário o algoritmo estará terminado neste ponto

Passo 4: *Faça* $j = j - 1$*. Se* $j = 0$*, o algoritmo deverá ser finalizado. Caso contrário, volte para o* ***Passo 3:****.* □

Observação 3.3 *No teste realizado, no passo 2, deve-se preocupar com a possibilidade último valor singular ser exatamente* 0*, o que levará, provavelmente, à ocorrência de um erro de inconsistência. Portanto, quando* $sigma_NumSing = 0$*,ele deve ser alterado para um valor mínimo para a máquina onde está sendo executado. No caso do Matlab, rodando num Pentium IV, foi utilizado o valor* 10^{-16}*.* □

A função `numsingnull`

A função Matlab desenvolvida para realizar o algoritmo descrito nesta seção é o `numsingnull.m` (apêndice A.15). Para usá-la, o usuário entra como parâmetro um vetor de valores singulares, obtidos a partir da utilização da função `svd` do Matlab, e a função retorna o número de valores singulares nulos deste vetor de acordo com o algoritmo 3.7. Essa função tem a seguinte sintaxe:

$$NumSing0 = \texttt{numsingnull}(sing, tol),$$

onde *sing* é um vetor de valores singulares de uma matriz qualquer, *tol* é igual a tolerância que será considerada para definir se um valor singular poderá ser nulo. A função retorna $NumSing0$ que é o número de valores singulares nulos.

Exemplo 3.15 *Para ilustrar o funcionamento do algoritmo, como exemplo, seja a matriz (2.36) apresentada na seção 2.10.2*

$$A'_p = \begin{bmatrix} 10 & 10 & 10 \\ 1 & 1 & 1 \\ 1 & 0.9999 & 1 \end{bmatrix}, \tag{3.50}$$

com o sua respectivo vetor de valores singulares dado por:

$$\underline{\sigma}_{A'_p} = \begin{bmatrix} 17.4928 & 0.0001 & 0 \end{bmatrix} \tag{3.51}$$

Para se usar a função `numsingnull` *no Matlab procede-se da seguinte forma:*

$>> Aplinha = \begin{bmatrix} 10 & 10 & 10; & 1 & 1 & 1; & 1 & 0.9999 & 1 \end{bmatrix};$

$>> sigmaA =$ `svd`$(Aplinha);$

$>> sigmaA = [17.4929; 0.0001; 0];$

$>>$ `numsingnull`$(sigmaA, 1e-5)$

ou

$>>$ `numsingnull`$(sigmaA)$

Note que na última linha de comando, a tolerância utilizada será 10^{-9}*, logo a resposta poderá ser diferente das opções anteriores.*

Executando Passo 1 do algoritmo, obtém-se:

$$\sigma_A = \begin{bmatrix} 17.4929 \\ 0.0001 \\ 0 \end{bmatrix}, tol = 10^{-5}, NumSing0 = 0 \text{ e } j = 3$$

Antes da execução do Passo 2 é observado que o último valor da lista é 0*, portanto deve-se modificar o vetor* σ_A *para*

$$\sigma_A = \begin{bmatrix} 17.4929 \\ 0.0001 \\ 10^{-16} \end{bmatrix}$$

Na execução do Passo 2 tem-se:

$$\frac{\sigma_A}{\sigma_{NumSing}} = \frac{17.4929}{10^{-16}} = 1.7493.10^{17}$$

e, portanto o vetor deverá ser normalizada. Logo:

$$\sigma_{normalizado} = \frac{\sigma_A}{\sigma_1} = [1 \quad 5.716e-6 \quad 5.7166e-018]$$

Executando o Passo 3 tem-se que $\sigma_{normalizado}{}^{(3)} < tol$ *e portanto,* $NumSing0 = 1$. *Agora, de acordo como o Passo 4,* $j = 2$. *Voltando ao Passo 3* $\sigma_{normalizado}{}^{(2)} < tol$ *e desta forma* $NumSing0 = 2$. *Em seguida, executando o Passo 4,* $j = 1$, *finalmente, tem-se que* $\sigma_{normalizado}{}^{(1)} > tol$, *terminando o algoritmo. Assim o número de de valores singulares nulos será armazenado em* $NumSing0$ *e, portanto* $NumSing0 = 2$. □

Capítulo 4

Algoritmos principais para operações com matrizes polinomiais

Neste capítulo são apresentados algoritmos para matrizes polinomiais que apresentam resultados que são importantes, tanto para a área matemática quanto para área de controle, como por exemplo: a inversa, o posto normal de uma matriz, base polinomial mínima, identidade de Bezout etc. Na seção 4.1 são apresentados os algoritmos classificados como fundamentais, que junto com os do capítulo 3, formam a base para os futuros algoritmos. São eles: redução por coluna, conversão de uma realização em espaço de estados em uma DFM, base polinomial mínima e posto normal de uma matriz polinomial. Na seção 4.2 são apresentados os algoritmos principais, que fazem uso desta base criada. São eles: inversa de uma matriz polinomial, determinante de uma matriz polinomial, descrição por função de matrizes à direita e à esquerda, máximo divisor comum de uma matriz polinomial e identidade de Bezout.

4.1 Algoritmos Fundamentais

Os algoritmos apresentados nessa seção servirão de base para os propostos na seção 4.2 e poderão, também, servir de suporte para futuros desenvolvimentos nas aplicações de matrizes polinomiais.

4.1.1 Redução por coluna

A redução por coluna (ou equivalentemente, redução por linha) de uma matriz polinomial é, em geral, um passo preliminar, antes da utilização de algoritmos mais complexos para matrizes polinomiais. Como exemplo, podem ser citados vários artigos que tratam da inversão de matrizes polinomiais (Zhang 1987, Inouye 1979), onde supõe-se que a matriz a ser invertida deverá ser reduzida por coluna. A redução por coluna de matrizes polinomiais pode ser feita de diversas maneiras (Wolovich 1974, Kailath 1980, Neven e Praagman 1993). Os métodos disponíveis na literatura são, geralmente, baseados em operações elementares sobre as colunas, que tendem a ser numericamente instáveis. Neste trabalho é apresentado um algoritmo que foi proposto por Basilio (2002), baseado na decomposição por valores singulares, sendo uma modificação de um outro algoritmo proposto anteriormente (Basilio e Kouvaritakis 1997) que foi utilizado para encontrar as matrizes polinomiais de uma DFM coprima à direita de uma dada matriz de transferência.

Descrição do algoritmo para redução por coluna

Seja $M(s) \in \mathbb{R}^{p\times p}[s]$ não-reduzida por coluna. A idéia do algoritmo de redução por coluna é, a cada iteração, pós-multiplicar $M(s)$ por matrizes unimodulares, com o objetivo de reordenar (em ordem decrescente) os graus das colunas e, em seguida, pós multiplicar a matriz resultante por outras matrizes unimodulares com vistas a reduzir os graus das colunas.

O algoritmo a seguir fornece, não só a matriz reduzida por coluna $\overline{M}(s)$, que é equivalente a $M(s)$, como também a matriz unimodular $U(s)$ que realiza a redução de $M(s)$.

Algoritmo 4.1 *:*

Passo 1: *Defina as matrizes*

$$\overline{M}(s) = M(s) \ \textit{e}\ U(s) = I_p.$$

Passo 2: *Encontre uma matriz elementar E tal que*

$$\hat{M}(s) = \overline{M}(s)E$$

e a $\hat{M}(s)$ tenha os graus das colunas ordenados de forma decrescente, ou seja:

$$\hat{\nu}_1 \geq \hat{\nu}_2 \geq \ldots \geq \hat{\nu}_p.$$

A matriz E é obtida permutando-se apropriadamente as colunas da matriz identidade de ordem p. Calcule

$$\hat{U}(s) = U(s)E.$$

Passo 3:*Escreva o $\hat{M}(s)$ como:*

$$\hat{M}(s) = \hat{M}_{hc}\hat{S}(s) + \hat{M}_{lc}\hat{\Psi}(s), \tag{4.1}$$

onde $\hat{S}(s)$ e $\hat{\Psi}(s)$ são matrizes polinomiais definidas de acordo com a (2.23).
Passo 4:*Realize a DVS de $\hat{M}_{hc}$ e escreva*

$$\hat{M}_{hc} = \hat{X}\hat{\Sigma}\hat{Y}^*.$$

Seja q o número de valores singulares nulos de $\hat{M}_{hc}$. Se $q = 0$, então $\hat{M}_{hc}$ é não singular e, portanto:

$$\overline{M}(s) = \hat{M}(s), \hat{U}(s) = U(s)$$

e o algoritmo chega ao fim. Caso contrário, retire as colunas

$$\hat{y}_{p-q+j}, j = 1, \ldots, q \text{ de } \hat{Y}$$

associados aos q valores singulares nulos e forme o conjunto

$$\hat{\gamma} = \{\hat{y}_{p-q+j}, j = 1, \ldots, q\}$$

Passo 5: *Seja $\hat{\kappa}_j$ a posição do primeiro elemento não-nulo de $\hat{y}_{p-q+1}$ e note que, pela forma como a DVS é realizada,*

$$\hat{\kappa}_{j+1} \geq \hat{\kappa}_j.$$

Forme o conjunto

$$\overline{\gamma} = \{\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_q\},$$

tomando apenas os elementos de $\hat{\gamma}$ com posições distintas do primeiro elemento não-nulo. Defina a posição do primeiro elemento não-nulo de $\overline{y}_j$ como $\overline{\kappa}_j$.

Passo 6: *Forme uma matriz unimodular $\overline{U}(s)$ substituindo-se as colunas $\overline{\kappa}_j, j = 1, \ldots, \overline{p}$, da matriz identidade de ordem q pelos vetores polinomiais $\overline{u}_j$ cujos elementos $\overline{u}_{k,j}(s), k = 1, \ldots, p$ são dados por:*

$$\overline{u}_{k,j}(s) = \begin{cases} 0, & k < \overline{\kappa}_j \\ \overline{y}_{k,j} s^{\hat{\nu}_{\overline{\kappa}_j} - \hat{\nu}_k}, & k \geq \overline{\kappa}_j \end{cases}$$

Passo 7: *Calcule*

$$\overline{M}(s) = \hat{M}(s)\overline{U}(s), U(s) = \hat{U}(s)\overline{U}(s)$$

e volte ao ***Passo 2.*** □

Note que esse algoritmo só será finalizado quando a matriz $\overline{M}(s)$ for reduzida por coluna.

A função `reducol`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `reducol.m` (apêndice A.16). Para usá-la, o usuário deve fornecer a matriz coeficiente associada a uma matriz polinomial $M(s)$ e o número de colunas (caso a matriz não seja quadrada). A função retorna a matriz coeficiente de uma matriz $\overline{M}(s)$ reduzida por coluna, equivalente à matriz $M(s)$, e a matriz unimodular $U(s)$ que realizou a redução de $M(s)$. Essa função tem a seguinte sintaxe:

$$[Mbarra, U] = \texttt{reducol}(M, m),$$

onde M é a matriz coeficiente associada a uma matriz polinomial $M(s)$, m número de colunas de $M(s)$, $Mbarra$ é a matriz coeficiente da matriz reduzida por coluna equivalente a $M(s)$ e U é a matriz coeficiente da matriz unimodular $U(s)$ que realiza a redução de $M(s)$.

Exemplo 4.1 *Para mostrar o funcionamento do algoritmo, seja a matriz polinomial*

$$M(s) = \begin{bmatrix} s^3+s+7 & 5s^2-3s+1 & 2s^4+s+1 \\ 1 & -2 & 2s+1 \\ s^3+s+2 & 5s^2 & 2s^4+2s^3+3 \end{bmatrix}, \tag{4.2}$$

com matriz coeficiente dada por

$$M = \left[\begin{array}{ccc|ccc|ccc|ccc|ccc} 0 & 0 & 2 & 1 & 0 & 0 & 0 & 5 & 0 & 1 & -3 & 1 & 7 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & -2 & 1 \\ 0 & 0 & 2 & 1 & 0 & 2 & 0 & 5 & 0 & 1 & 0 & 0 & 2 & 0 & 3 \end{array}\right]. \tag{4.3}$$

Para utilizar a função `reducol` *no Matlab, procede-se da seguinte forma:*

```
>> M = [0 0 2 1 0 0 0 5 0 1 -3 1 7  1 1;
        0 0 0 0 0 0 0 0 0 0  0 2 1 -2 1;
        0 0 2 1 0 2 0 5 0 1  0 0 2  0 3];
>> [Mbarra, U] = reducol(M, 3);
```

ou

```
>> [Mbarra, U] = reducol(M);
```

Analisando a matriz (4.3) pode ser verificado que

$$M_{hc} = \begin{bmatrix} 1 & 5 & 2 \\ 0 & 0 & 0 \\ 1 & 5 & 2 \end{bmatrix}$$

é singular e, portanto, $M(s)$ é não-reduzida por coluna. Logo deverá ser utilizado o algoritmo 4.1 para obter a matriz $\overline{M}(s)$, reduzida por coluna, e também a matriz U(s), tais que

$$\overline{M}(s) = M(s)U(s).$$

Seguindo o Passo 1 será definido

$$\overline{M}(s) = M(s) \text{ e } U(s) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

De acordo com o Passo 2 serão obtidas as matrizes E e $\hat{M}(s)$ objetivando colocar as colunas de $\overline{M}(s)$ de forma decrescente. Note que os graus das colunas de $M(s)$ são $3, 2, 4$ portanto,

$$E = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Seguindo, agora, o Passo 3, obtém-se

$$\hat{M} = \begin{bmatrix} 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 5 & 1 & 1 & -3 & 1 & 7 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 1 & 1 & -2 \\ 2 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 5 & 0 & 1 & 0 & 3 & 2 & 0 \end{bmatrix}$$

e

$$\hat{M}_{hc} = \begin{bmatrix} 2 & 1 & 5 \\ 0 & 0 & 0 \\ 2 & 1 & 5 \end{bmatrix}.$$

Note que os graus das colunas de $\hat{M}(s)$ são iguais a 4, 3 e 2.

Pode-se ver claramente que $\hat{M}_{hc}$ tem posto igual a 1. Desta forma, de acordo com o Passo 4, realizando-se a DVS de M_{hc} obtém-se:

$$\hat{X} = \begin{bmatrix} -0.7071 & 0 & -0.7071 \\ 0 & 1.0000 & 0 \\ -0.7071 & 0 & 0.7071 \end{bmatrix}, \hat{\Sigma} = \begin{bmatrix} 7.7460 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

e

$$\hat{Y} = \begin{bmatrix} -0.3651 & 0.9309 & 0 \\ -0.1826 & -0.0716 & -0.9806 \\ -0.9129 & -0.3581 & 0.1961 \end{bmatrix},$$

e retirando-se as colunas de $\hat{Y}$ correspondentes aos dois valores singulares nulos, forma-se o seguinte

$$\hat{\gamma} = \left\{ \begin{bmatrix} 0.9309 \\ -0.0716 \\ -0.3581 \end{bmatrix}, \begin{bmatrix} 0 \\ -0.9806 \\ 0.1961 \end{bmatrix} \right\}.$$

Pode-se, então, de acordo com o Passo 5 do algoritmo 4.1 obter a seguinte matriz:

$$\overline{\gamma} = [\overline{y}_1 \quad \overline{y}_2] = \begin{bmatrix} 0.9309 & 0 \\ -0.0716 & -0.9806 \\ -0.3581 & 0.1961 \end{bmatrix},$$

de onde se pode ver que $\overline{\kappa}_1 = 1$ e $\overline{\kappa}_2 = 2$

Assim é possível construir, de acordo com o Passo 6, a matriz a $\overline{U}(s)$ diretamente,que será, dada por:

$$\overline{U}(s) = \begin{bmatrix} 0.9309 & 0 & 0 \\ -0.0716s & -0.9806 & 0 \\ -0.3581s^2 & 0.1961s & 1 \end{bmatrix}.$$

Pelo Passo 7 do algoritmo deve-se, então, calcular $\overline{M}(s)$ e $U(s)$, que serão dados por

$$\overline{M}(s) = \begin{bmatrix} 1.0742 & 0 & 0 \\ 0 & 0 & 0 \\ 1.8619 & 0 & 0 \end{bmatrix} s^3 + \begin{bmatrix} -0.4297 & -0.5883 & 5.0000 \\ 0.7161 & 0 & 0 \\ -0.0716 & 0 & 5.0000 \end{bmatrix} s^2 +$$

$$\begin{bmatrix} 0.4297 & -0.7845 & -3.0000 \\ 1.7903 & -0.3922 & 0 \\ -0.1432 & -0.9806 & 0 \end{bmatrix} s + \begin{bmatrix} 0.9309 & -6.8641 & 1.0000 \\ 0.9309 & -0.9806 & -2.0000 \\ 2.7928 & -1.9612 & 0 \end{bmatrix}$$

e

$$U(s) = \overline{U}(s)E = \begin{bmatrix} 0 & 0 & 0 \\ -0.3581 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} s^2 + \begin{bmatrix} -0.0716 & 0 & 0 \\ 0 & 0.1961 & 0 \\ 0 & 0 & 0 \end{bmatrix} s + \begin{bmatrix} 0 & -0.9806 & 0 \\ 0 & 0 & 1.0000 \\ 0.9309 & 0 & 0 \end{bmatrix}.$$

Observando a matriz $\overline{M}(s)$, verifica-se, ainda, que é uma matriz não-reduzida por coluna. Portanto, deve-se retornar ao Passo 2 do algoritmo 4.1 e encontrar a nova matriz E. É fácil observar que:

$$E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

já que os graus das colunas de $\overline{M}(s)$ estão em ordem decrescente. Logo,

$$\hat{M}(s) = \overline{M}(s) \text{ e } \hat{U}(s) = U(s).$$

Assim como na iteração anterior, no Passo 3 é necessário escrever explicitamente a matriz $\hat{M}_{hc}$ e matriz $\hat{S}(s)$ dos graus das colunas de $\hat{M}(s)$, quais sejam: serão dados por:

$$\hat{M}_{hc} = \begin{bmatrix} 1.0742 & -0.5883 & 5.0000 \\ 0 & 0 & 0 \\ 1.8619 & 0 & 5.0000 \end{bmatrix}$$

e

$$\hat{S}(s) = diag\{s^3, s^2, s^2\}.$$

Continuando o algoritmo, deve-se, no Passo 4, calcular a DVS de $\hat{M}_{hc}$, que será dada por:

$$\hat{X} = \begin{bmatrix} -0.6941 & -0.7199 & 0 \\ 0 & 0 & 1.0000 \\ -0.7199 & 0.6941 & 0 \end{bmatrix}, \hat{\Sigma} = \begin{bmatrix} 7.3825 & 0 & 0 \\ 0 & 0.6822 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

e

$$\hat{Y} = \begin{bmatrix} -0.2825 & 0.7609 & -0.5841 \\ 0.0553 & 0.6208 & 0.7820 \\ -0.9577 & -0.1886 & 0.2175 \end{bmatrix},$$

de onde se pode verificar que existe um valor singular nulo. Conseqüentemente $\hat{\gamma}$ *será formado por um único vetor, isto é:*

$$\hat{\gamma} = \begin{bmatrix} -0.5841 \\ 0.7820 \\ 0.2175 \end{bmatrix}$$

o que implica que

$$\overline{U} = \begin{bmatrix} -0.5841 & 0 & 0 \\ 0.7820 & 1 & 0 \\ 0.2175 & 0 & 1 \end{bmatrix}$$

Conseqüentemente, as novas matrizes $M(s)$ *e* $U(s)$ *serão dadas por:*

$$\overline{M}(s) = \begin{bmatrix} -1.0150 & -0.5883 & 5.0000 \\ -0.7250 & 0 & 0 \\ -0.7250 & 0 & 5.0000 \end{bmatrix} s^2 + \begin{bmatrix} -5.4012 & -0.7845 & -3.0000 \\ -2.2475 & -0.3922 & 0 \\ -1.4500 & -0.9806 & 0 \end{bmatrix} s + \begin{bmatrix} -0.5437 & -6.8641 & 1.0000 \\ -0.5437 & -0.9806 & -2.0000 \\ -1.6312 & -1.9612 & 0 \end{bmatrix} \tag{4.4}$$

e

$$U(s) = \hat{U}(s)\overline{U}(s)) = \begin{bmatrix} 0 & 0 & 0 \\ 0.3625 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} s^2 + \begin{bmatrix} -0.7250 & 0 & 0 \\ 0.2175 & 0.1961 & 0 \\ 0 & 0 & 0 \end{bmatrix} s + \begin{bmatrix} 0 & -0.9806 & 0 \\ 0 & 0 & 1.0000 \\ -0.5437 & 0 & 0 \end{bmatrix}. \tag{4.5}$$

É fácil verificar que $\overline{M}(s)$ *é reduzida por coluna, o que permite terminar o algoritmo.*

□

4.1.2 Obtenção da função de transferência de um sistema a partir de sua realização em espaço de estados.

O objetivo desta seção é apresentar um algoritmo para obter a matriz $N(s) \in \mathbb{R}^{p\times m}[s]$ e um polinômio $d(s)$ da função de transferência

$$H(s) = \frac{N(s)}{d(s)}$$

conhecendo-se uma realização em espaço de estados(A, B, C, D) para esse sistema. Este algoritmo foi inspirado nas fórmulas dadas, resumidamente, em Kailath (1980,

páginas 656-657) que são conhecidas como fórmulas de*Leverrier-Souriau-Faddeeva-Frame* . Basicamente o objetivo do algoritmo é calcular

$$H(s) = C(sI - A)^{-1}B + D \tag{4.6}$$

com $H(s) \in \mathbb{R}^{p\times m}(s)$.

Descrição do algoritmo

Note que, a equação (4.6)requer o cálculo de $(sI-A)^{-1}$, que pode ser obtida usando a relação:

$$(sI - A)^{-1} = \frac{Adj(sI - A)}{det(sI - A)}.$$

Desta forma, o algoritmo a ser adotado fará duas operações principais: o cálculo do determinante e o cálculo da adjunta da matriz $(sI - A)$. De acordo com o algoritmo de *Leverrier*, tem-se:

$$a(s) \triangleq det(sI - A) = s^n + a_1 s^{n-1} + a_2 s^{n-2} + \ldots + a_{n-1}s + a_n$$

e

$$S(s) = Adj(sI - A) = S_1 s^{n-1} + S_2 s^{n-2} + \ldots + S_{n-1}s^1 + S_n,$$

onde

$$\begin{cases} a_1 = -tr(A) & S_1 = I \\ a_2 = -\frac{1}{2}tr(S_2A) & S_2 = S_1A + a_1I \\ a_3 = -\frac{1}{3}tr(S_3A) & S_3 = S_2A + a_2I \\ \vdots & \vdots \\ a_{n-1} = -\frac{1}{n-1}tr(S_{n-1}A) & S_n = S_{n-1}A + a_{n-1}I \\ a_n = -\frac{1}{n}tr(S_nA) & \end{cases} . \tag{4.7}$$

Assim sendo, os coeficientes $a_i, i = 1, \ldots, n$ e as matrizes de coeficientes $S_i, i = 1, \ldots, n$, poderão ser calculados recursivamente de acordo com a equação 4.7.

Observação 4.1 *Pode-se observar que a matriz coeficiente da adjunta de* $(sI - A)$ *será dada por*

$$S = [S_1 \quad S_2 \quad \ldots \quad S_n]$$

e a matriz de coeficientes do determinante será dada por:

$$[1 \quad a_1 \quad a_2 \quad \ldots \quad a_n]$$

□

Após o cálculo da inversa, $N(s)$ poderá ser calculada facilmente, sendo dada por:

$$N(s) = CS(s)B + d(s)D.$$

Algoritmo 4.2 *Seja* (A, B, C, D) *uma realização em espaço de estados onde* $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$ *e* $D \in \mathbb{R}^{p\times m}$.

Passo 1: *Faça* $S_i = 0$, $a_1 = 1$, $d = 1$, $S = [\,]$;

Passo 2: *Para* i *variando de* 1 *a* n, *faça* $S_i = S_iA + A_iI$, $a_i = -1/i[tr(S_iA)]$, $d = [d \;\; a_i]$, $S = [S \;\; S_i]$, *fim;*

Passo 3: *Utilizando o algoritmo Convmat, calcule:*

$$N_1(s) = CAdj(sI - A)B \text{ e } N_2 = d(s)D;$$

Passo 4: *Utilizando o algoritmo Matsum, calcule*

$$N(s) = N_1(s) + N_2(s).$$

□

A função `Ss2tflev`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `ss2tflev.m` (apêndice A.17). Para usá-la, o usuário deve fornecer as matrizes da representação em espaço de estados (A, B, C, D) e a função retorna a matriz coeficiente N associada à matriz $N(s)$ e o vetor de coeficientes d associado ao polinômio $d(s)$, que representam a função racional: $H(s) = N(s)/d(s)$. Essa função tem a seguinte sintaxe:

$$[N, d] = \texttt{ss2tflev}(A, B, C, D),$$

onde (A, B, C, D) são as matrizes da representação em espaço de estados, N é a matriz coeficiente associada à matriz $N(s)$ e d é o vetor de coeficientes associado ao polinômio $d(s)$.

Exemplo 4.2 *Sejam as seguintes matrizes de uma realização em espaço de estados de um sistema.*

$$A = \begin{bmatrix} -3.1000 & -0.7500 & -0.5410 & -1.3525 & 0.0000 & -2.7586 \\ 1.0000 & 0 & 0 & 0 & 0 & 0 \\ -5.1878 & 2.2181 & 0.6000 & -7.6667 & -5.0990 & 3.0594 \\ 0 & 0 & 1.0000 & 0 & 0 & 0 \\ -0.1595 & 0.2175 & 0.1177 & 0.1961 & 0 & -0.4000 \\ 0 & 0 & 0 & 0 & 1.0000 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.0000 & -1.3793 & -0.0000 \\ 0 & 0 & 0 \\ -1.6997 & 0.6799 & 1.6997 \\ 0 & 0 & 0 \\ 0 & -0.2000 & 0.2000 \\ 0 & 0 & 0 \end{bmatrix} C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

e a matriz D será um bloco de zeros 3×3.

Para utilizar a função `ss2tflev` *no Matlab, procede-se da seguinte forma:*

```
>> A = [-3.1000  -0.7500  -0.5410  -1.3525   0.0000  -2.7586;
         1.0000        0        0        0        0        0;
        -5.1878   2.2181   0.6000  -7.6667  -5.0990   3.0594;
              0        0   1.0000        0        0        0;
        -0.1595   0.2175   0.1177   0.1961        0  -0.4000;
              0        0        0        0   1.0000        0];
>> B = [0.0000  -1.3793  -0.0000;
              0        0        0;
        -1.6997   0.6799   1.6997;
              0        0        0;
              0  -0.2000   0.2000;
              0        0        0];
>> C = 0  1  0  0  0  0;  0  0  0  1  0  0;  0  0  0  0  0  1]
>> D = [0  0  0;  0  0  0;  0  0  0];
>> [N, d] = ss2tflev(A, B, C, D);
```

Executando o Passo 1 do algoritmo tem-se com atribuições iniciais: $S_i = 0$, $a_1 = 1$, $d = 1$, $S = [\,]$,

na primeira iteração do Passo 2 obtém-se

$$i = 1; S_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}; a_i = 2.5;\ d = [1 \quad 2.5]; S = S_i$$

e novamente executando o Passo 2:

$$i = 2;\ \ S_i = \begin{bmatrix} -0.6000 & -0.7500 & -0.5410 & -1.3525 & 0.0000 & -2.7586 \\ 1.0000 & 2.5000 & 0 & 0 & 0 & 0 \\ -5.1878 & 2.2181 & 3.1000 & -7.6667 & -5.0990 & 3.0594 \\ 0 & 0 & 1.0000 & 2.5000 & 0 & 0 \\ -0.1595 & 0.2175 & 0.1177 & 0.1961 & 2.5000 & -0.4000 \\ 0 & 0 & 0 & 0 & 1.0000 & 2.5000 \end{bmatrix};$$

$$a_i = 4.75; d = [1 \ \ 2.5 \ \ 4.75];$$

$$S = [S \ \ S_i] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -0.6000 & -0.7500 & \dots \\ 0 & 1 & 0 & 0 & 0 & 0 & 1.0000 & 2.5000 & \dots \\ 0 & 0 & 1 & 0 & 0 & 0 & -5.1878 & 2.2181 & \dots \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1 & 0 & -0.1595 & 0.2175 & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots \end{bmatrix}$$

$$\begin{bmatrix} \dots & -0.5410 & -1.3525 & 0.0000 & -2.7586 \\ \dots & 0 & 0 & 0 & 0 \\ \dots & 3.1000 & -7.6667 & -5.0990 & 3.0594 \\ \dots & 1.0000 & 2.5000 & 0 & 0 \\ \dots & 0.1177 & 0.1961 & 2.5000 & -0.4000 \\ \dots & 0 & 0 & 1.0000 & 2.5000 \end{bmatrix}.$$

Executando Passo 2 sucessivamente até que i seja igual a $n = 6$ *e logo em seguida na execução dos passo 3 e 4 será obtido o seguinte resultado final*

$$H(s) = \frac{n(s)}{d(s)}$$

onde

$$n(s) = \begin{bmatrix} 0.0000 & -1.3793 & -0.0000 \\ -1.6997 & 0.6799 & 1.6997 \\ 0 & -0.2000 & 0.2000 \end{bmatrix} s^4 + \begin{bmatrix} 0.9195 & 0.4598 & -0.9195 \\ -5.2690 & 10.2830 & 4.2492 \\ -0.2000 & -0.2000 & 0.7000 \end{bmatrix} s^3 +$$

$$\begin{bmatrix} 2.2989 & -12.8736 & -2.2989 \\ -1.9546 & -0.8498 & -0.5949 \\ -1.1000 & 0.1000 & 1.8500 \end{bmatrix} s^2 + \begin{bmatrix} 0.9195 & -2.2989 & 0.4598 \\ -1.3597 & 1.6147 & 5.3540 \\ -1.3500 & -0.0500 & 4.8500 \end{bmatrix} s +$$

$$\begin{bmatrix} 1.8391 & 0.9195 & -6.8966 \\ -1.5297 & -0.7649 & 0.7649 \\ 0.2500 & -4.7500 & 1.5000 \end{bmatrix}$$

e

$$d(s) = s^6 + 2.5s^5 + 4.75s^4 + 21s^3 + 11.5s^2 - 2s + 9.75$$

□

4.1.3 Posto normal de uma matriz polinomial

O conhecimento do posto normal de uma matriz, geralmente, é um passo preliminar para o cálculo da base polinomial mínima do espaço nulo de uma matriz polinomial (Basilio e Moreira 2004, Zúñiga e Henrion 2005), conforme observado na seção 4.1.4. Nesta seção será apresentado um algoritmo baseado em DVS para o cálculo do posto normal de matrizes polinomiais que foi desenvolvido por Moreira e Basilio (2004).

Descrição do algoritmo

Suponha que $m > p$, por simplicidade, e que $A(s)$ possui a seguinte forma de Smith:

$$\Sigma_A(s) = \begin{bmatrix} \epsilon_1(s) & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \epsilon_1(s) & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \epsilon_m(s) & 0 & 0 & 0 \end{bmatrix} \tag{4.8}$$

onde

$$\epsilon_k(s) = 0 \text{ para } k = m - \nu + 1, \dots, m.$$

Então $\alpha = p - \nu$ é o posto normal de $A(s)$, e $m - p + \nu$ é a dimensão do espaço nulo à direita para todo s.

Uma maneira simples para o cálculo do posto normal pode ser desenvolvida a partir da equação (4.8) notando que a matriz $A(s)$ somente perde posto, além da normal, para um número finito de valores de s, que são os zeros dos polinômios invariantes $\epsilon_i(s), i = 1, \dots, \alpha$. Além disso, como $\epsilon_i(s)$ divide $\epsilon_{i+1}(s)$, então o número de valores de $s_k \in \mathbb{C}$ que tornam $\rho[A(s_k)] < \alpha$ é determinado pelo grau de $\epsilon_\alpha(s)$. Portanto, para saber qual é o posto normal de $A(s)$ basta verificar o posto de $A(s_k)$ para k valores distintos de $s = s_k$, onde k é maior ou igual que o grau de $\epsilon_\alpha(s)$ e

obter α fazendo:

$$\alpha = \max_{s_k}\{\rho[A(s_k)]\},$$

onde $\rho(.)$ denota o posto de uma matriz complexa.

Uma estimativa do número de freqüências que precisam ser utilizadas para a verificação do posto normal de $A(s)$ pode ser obtida a partir da definição de polinômios invariantes. Seja $\Delta_i(s)$ o máximo divisor comum de todos os menores $i \times i$ de A(s). Então os polinômios invariantes $\epsilon_i(s)$ são obtidos fazendo-se

$$\epsilon_i(s) = \frac{\Delta_i(s)}{\Delta_{i-1}(s)}.$$

Portanto, é fácil verificar que o somatório dos graus dos polinômios invariantes é igual ao grau de $\Delta_\alpha(s)$, isto é,

$$\sum_{i=1}^{\alpha} gr(\epsilon_i) = gr(\Delta_\alpha)$$

e, portanto,

$$gr(\epsilon_\alpha) \leq gr(\Delta_\alpha).$$

Um limite superior para o grau de $\Delta_\alpha(s)$ pode ser obtido supondo $\alpha = p$ e lembrando que o máximo divisor comum de todos os menores $p \times p$ de $A(s)$ não excede o menor somatório dos graus de p colunas de $A(s)$. Desta forma, os seguinte passos podem ser utilizados para calcular o posto normal de uma matriz polinomial.

Algoritmo 4.3 *:*

Passo 1: *Verifique a quantidade de linhas e colunas da matriz, caso o número de linhas for maior que o número de colunas faça a transposição da matriz de entrada;*

Passo 2: *Calcule os graus da colunas da matriz polinomial. Em seguida forme um vetor GrauOrd em que os elementos são os graus da matriz polinomial ordenados de forma crescente.*

Passo 3: *Selecione os m primeiros elementos de GrauOrd e faça l igual a somatório deles.*

Passo 4: *Atribua valores reais ou complexos a $s_k, k = 1, \ldots, l+1$, de forma que todos os s_k sejam distintos. Para cada s_k substitua este valor em $A(s_k)$ utilizando o*

algoritmo Polymatval descrito na seção 3.1.8 e faça a DVS. Em seguida, obtenha o posto $\rho[A(s_k)]$ a partir dos valores singulares nulos de $A(s_k)$ utilizando o algoritmo NumSingNull.

Passo 5: *O posto normal α será dado por:*

$$\alpha = \max_{s_k}\{\rho[A(s_k)]\}.$$

□

Observação 4.2 *O algoritmo poderá ser terminado antes que todos os valores de s_k tenham sido verificados se, para algum s_k, $\rho[A(s_k)] = \min(p, m)$.* □

A função `polymatposto`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `polymatposto.m` (apêndice A.19) . Para usá-la, o usuário deve fornecer a matriz de coeficientes associada a uma matriz polinomial $A(s)$, a tolerância no qual o algoritmo considerará a perda de posto de uma matriz (caso não seja fornecida, a tolerância adotada será 10^{-9}) e o número de colunas (caso a matriz não seja quadrada). A função retorna o posto da matriz $A(s)$. Essa função tem a seguinte sintaxe:

$$rA = \texttt{polymatposto}(A, tol, ncol),$$

onde A é a matriz coeficiente associada a uma matriz polinomial $A(s)$, *tol* tolerância no qual o algoritmo considerará a perda de posto de uma matriz, *ncol* número de colunas de $A(s)$ e *rank* é posto normal da matriz polinomial $A(s)$.

Exemplo 4.3 *Como exemplo de aplicação será utilizado o seguinte exemplo, seja a matriz polinomial:*

$$A(s) = \begin{bmatrix} s^3 + s^2 + 5s + 3 & -s^2 - 3s + 1 & 2s^4 + s^3 + 2s + 1 \\ -3 & -2 & s^2 + 5s + 1 \\ -6 & -4 & 2 * (s^2 + 5s + 1) \end{bmatrix},$$

cuja matriz de coeficientes é dada por:

$$A = \begin{bmatrix} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 10 & -6 & -4 & 2 \end{bmatrix}.$$

Conforme observado anteriormente a segunda e a terceira linhas são linearmente dependentes; logo deverá ser encontrado posto normal igual a dois. Então, para se calcular o posto normal de uma matriz polinomial no Matlab, usando o algoritmo Polymatposto, procede-se da seguinte forma:

$$>> A = [0\ \ 0\ \ 2\ \ 1\ \ 0\ \ 1\ \ 1\ \ -1\ \ 0\ \ 5\ \ -3\ \ 2\ \ 3\ \ 1\ \ 1;$$
$$0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 5\ \ -3\ \ -2\ \ 1;$$
$$0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 2\ \ 0\ \ 0\ \ 10\ \ -6\ \ -4\ \ 2];$$
$$>> rA = \texttt{polymatposto}(A, 1e-8, 3);$$

ou

$$>> rA = \texttt{polymatposto}(A, 1e-8);$$

ou

$$>> rA = \texttt{polymatposto}(A);$$

Na execução do Passo 1 é observado que o número de linhas e colunas é igual e, desta forma, não existe a necessidade de se fazer a transposição da matriz. No Passo 2 serão calculados os graus das colunas através do algoritmo Coldeg visto na seção 3.1.9, obtendo-se:

$$Graus\ das\ colunas\ de\ A(s) \Longrightarrow [3\ \ 2\ \ 4],$$

logo

$$GrauOrd = [2\ \ 3\ \ 4].$$

De acordo com o Passo 3, como o número de linhas m é igual a três, o somatórios dos graus será igual a nove. Assim para a execução do Passo 4, tem-se:

$$k = 1, \ldots, 9 + 1, \Rightarrow k = 1, \ldots, 10,$$

isto é, devem ser escolhidos 10 valores numéricos, aleatoriamente, para serem substituído na variável s da matriz através do algoritmo polymatval e verificado o posto da matriz através do número de valores singulares nulos utilizando o algoritmo Numsingnull. É verificado, depois das substituições, que o posto de todas as matrizes gerada é igual a dois e, portanto, o posto da matriz deste exemplo é dois, como já sabido inicialmente. □

4.1.4 Base polinomial mínima

Nesta seção será apresentado um algoritmo para calcular uma base polinomial mínima para o espaço nulo de matrizes polinomiais, conforme descrito teoricamente na seção 2.8. Este algoritmo tem a vantagem de trabalhar com DVS e matrizes de Sylvester associada a matriz polinomial correspondente.

Calcular o espaço nulo de uma matriz polinomial tem várias aplicações em controle, um delas foi explorada em Neven e Praagman (1993) para resolver o problema da redução por coluna em uma matriz polinomial, que é freqüentemente necessária num projeto de sistemas de controle. Em Zúñiga e Henrion (2005) foi apresentado uma maneira sistemática de como se obter a função de transferência de um sistema linear computando a o espaço nulo de uma matriz polinomial, onde é dado um exemplo de um sistema de segunda ordem massa-mola; este mesmo exemplo é mais bem descrito em Zúñiga e Henrion (2004). Também em Basilio e Kouvaritakis (1997) e Basilio e Moreira (2004) são apresentados algoritmos para o cálculo do espaço nulo para se obter uma DFM coprima à direita para uma $G(s) \in \mathbb{R}^{p\times m}(s)$. Nesse trabalho o algoritmo adotado para o cálculo de bases polinomiais mínimas para o espaço nulo de uma matriz polinomial será aquele proposto em Basilio e Moreira (2004).

Descrição do algoritmo

O teorema 2.1 sugere a seguinte maneira para computar base polinomial mínima do espaço nulo à direita.

Algoritmo 4.4 *Suponha que $A(s) \in \mathbb{R}^{p\times m}[s]$ e que o espaço nulo de $A(s)$ tenha dimensão α:*

Passo 1: *Ache um vetor polinomial $\underline{f}_1(s)$ não-nulo, de menor grau possível ν_1, tal que*

$$A(s)\underline{f}_1(s) = 0.$$

Passo 2: *Ache um vetor $\underline{f}_2(s)$ que satisfaça*

$$A(s)\underline{f}_2(s) = 0,$$

com o menor grau possível $\nu_2 \geq \nu_1$ entre todos os vetores linearmente independentes de $\underline{f}_1(s)$.

Passo 3: *Continue calculando os vetores até que se tenha obtido os α vetores polinomiais com graus*

$$\nu_1 \leq \nu_2 \leq \ldots \leq \nu_\alpha.$$

□

Quando a dimensão do espaço nulo a direita é conhecido, ou porque o problema a ser considerado possui esta informação ou porque de alguma forma ele foi calculado, por exemplo, através do cálculo do seu posto normal (Moreira 2001), então o problema de se computar o espaço nulo de uma matriz polinomial A(s) qualquer passa a ser um problema de se calcular o espaço nulo de uma matriz de coeficientes reais. Para tanto escreva a matriz $A(s)$ e o vetor $f_i(s)$ como:

$$A(s) = A_0 s^\phi + A_1 s^{(\phi-1)} + \cdots + A_\phi, A_i \in \mathbb{R}^{p\times m}, i = 0, \ldots, \phi \tag{4.9}$$

e

$$\underline{f}_i(s) = \underline{f}_{i0} s^{\nu_i} + \underline{f}_{i1} s^{(\nu_i-1)} + \cdots + \underline{f}_{i\nu_i}, \underline{f}_{ij} \in \mathbb{R}^n, j = 0, \ldots, \nu_i. \tag{4.10}$$

Utilizando o conceito de matrizes de convolução, dado na seção 2.2, pode-se definir:

$$C_{\nu_i}(A) = \underbrace{\begin{bmatrix} A_0 & 0 & \cdots & 0 \\ A_1 & A_0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_\phi & A_{\phi-1} & \ddots & A_0 \\ 0 & A_\phi & \ddots & \vdots \\ \vdots & \vdots & \ddots & A_{\phi-1} \\ 0 & 0 & \cdots & A_\phi \end{bmatrix}}_{\nu_i+1 \text{ blocos}} \tag{4.11}$$

e

$$\underline{f}^{(i)} = \begin{bmatrix} \underline{f}_{i0} \\ \underline{f}_{i1} \\ \vdots \\ \underline{f}_{i\nu_i} \end{bmatrix} \tag{4.12}$$

onde $C_{\nu_i}(A) \in \mathbb{R}^{p(\phi+\nu_i+1)\times m(\nu_i+1)}$ e $\underline{f}^{(i)} \in \mathbb{R}^{n(\nu_i+1)}$. Logo existe a seguinte equivalência:

$$A(s)\underline{f}_i(s) = \underline{0} \Longleftrightarrow C_{\nu_i}(A)\underline{f}^{(i)} = \underline{0}.$$

Baseado na nesta análise, pode-se enunciar o seguinte teorema.

Teorema 4.1 *Uma condição necessária para um vetor polinomial* $\underline{f}_i(s)$ *de grau* ν_i *ser um elemento de uma base polinomial mínima para o espaço nulo à direita de* $A(s)$ *é que* $C_{\nu_i}(A)$ *perca posto e o vetor* $\underline{f}^{(i)}$ *formado empilhando-se os vetores dos coeficientes de* $\underline{f}_i(s)$*, dado na equação (4.12), pertença ao espaço nulo à direita de* $C_{\nu_i}(A)$ *da equação (4.11).* □

Os teoremas 2.1 e 4.1 juntos podem ser usados para criar um algoritmo robusto para a computação de uma base polinomial mínima para o espaço nulo à direita de uma matriz polinomial, se ela existir. De acordo com os teoremas, a procura por um vetor polinomial da base polinomial mínima pode ser feita a partir da DVS de $C_{\nu_i}(A)$, ou seja,

$$C_{\nu_i}(A) = U_{\nu_i}\Sigma_{\nu_i}V_{\nu_i}^t.$$

O conjunto dos vetores que podem ser candidatos a base polinomial mínima do espaço nulo à direita de $A(s)$ será formados pelas colunas de V_{ν_i} associadas aos valores singulares nulos da matriz $C_{\nu_i}(A)$ dada na equação (4.11) e pelas colunas correspondentes ao excesso de colunas em relação as linhas, isto é, se $C_{\nu_i}(A)$ tem mais colunas do que linhas.

Assim uma matriz

$$F(s) = \left[\begin{array}{cccc} \underline{f}_1(s) & \underline{f}_2(s) & \ldots & \underline{f}_\alpha(s) \end{array} \right],$$

nas quais as colunas formam uma base polinomial mínima para o espaço nulo à direita de $A(s)$, pode ser obtida executando-se os seguintes passos.

Algoritmo 4.5 *:*

Passo 1: *Calcule o posto normal* ρ *da matriz A(s), utilizando o algoritmo da seção 4.1.3, e calcule o espaço nulo a direita de A(s), fazendo* $\alpha = m - p + \rho$

Passo 2: *Faça* $i = 1$ *e* $grau[\underline{f}_i(s)] = \nu_i = 0$

Passo 3: *Forme a matriz* $C_{\nu_i}(A)$ *de acordo com a equação (4.11), utilizando o algoritmo dado na seção 3.1.5*

Passo 4 *Obtenha a DVS de*

$$C_{\nu_i}(A) = U_{\nu_i}\Sigma_{\nu_i}V_{\nu_i}^t. \tag{4.13}$$

Passo 5*Atribua a n_{ν_i} a dimensão do espaço nulo de $C_{\nu_i}(A)$, no qual é dado pela número de valores singulares nulos dada na equação (4.13) mais o número de colunas em excesso.*

Passo 6 *Se $n_{\nu_i} = 0$, faça $\nu_i = \nu_i + 1$ e volte para o* ***Passo 5****.*

Se $n_{\nu_i} > 0$, então haverá até n_{ν_i} vetores de grau ν_i que podem ser inseridos na base polinomial mínima. Estes vetores serão formados pelas n_{ν_i} colunas da matriz V_{ν_i}. Quando $n_{\nu_i} > 0$ e $i = 0$, um vetor $\underline{f}^{(i)}$ será usado para formar um vetor polinomial da base se pelo menos um dos seus primeiros m elementos não for nulo. Quando $n_{\nu_i} > 0$ e $i \neq 0$, $\underline{f}^{(i)}$ será inserido na matriz $F(s)$ se a matriz F_{hc} associada, formada pelos m primeiros coeficientes dos elementos de $\underline{f}_i^{(k)}, k = 1, \ldots, i$ tiver posto completo.

Repita este passo até que:

- *todos os vetores formados de V_{ν_i} tenham sido verificados ou*
- *a base tenha sido completada.*

Faça $i = i + 1$, a cada vez que um vetor polinomial é somado à base polinomial mínima.

Passo 7: *Se $i < \alpha$ então faça $i = i + 1$ e $\nu_i = \nu_i + 1$ e volte para o* ***Passo 3:****.* □

A função `polybases`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `polybases.m` (apêndice A.18) . Para usá-la, o usuário deve fornecer a matriz de coeficientes associada a uma matriz polinomial $A(s)$, o posto normal desta matriz e o número de colunas (caso a matriz não seja quadrada). A função retorna uma matriz coeficientes F associada à matriz polinomial $F(s)$ que é uma base polinomial mínima para o espaço nulo à direita de $A(s)$. Essa função tem a seguinte sintaxe:

$$[F] = \texttt{polybases}(A, rA, ncol),$$

onde F é a matriz coeficiente associada a uma matriz polinomial $F(s)$, $ncol$ é número de colunas de $A(s)$ e rA é o posto normal de $A(s)$.

Exemplo 4.4 *Neste exemplo será mostrada a execução, passo a passo, do algoritmo. Seja a matriz polinomial:*

$$A(s) = \begin{bmatrix} s^3+s^2+5s+3 & -s^2-3s+1 & 2s^4+s^3+2s+1 \\ -3 & -2 & s^2+5s+1 \\ -6 & -4 & 2(s^2+5s+1) \end{bmatrix},$$

que possui a seguinte matriz coeficiente:

$$A = \begin{bmatrix} 0 & 0 & 2 & 1 & 0 & 1 & 1 & -1 & 0 & 5 & -3 & 2 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 5 & -3 & -2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 10 & -6 & -4 & 2 \end{bmatrix}.$$

Pode ser observado claramente que a segunda e a terceira linhas são linearmente dependentes. O que permite concluir que o posto normal de $A(s)$ será igual a 2 (na seção 4.1.3 será apresentado um algoritmo para o cálculo do posto normal de uma matriz polinomial). Assim, para se calcular uma base polinomial mínima para o espaço nulo à direita de $A(s)$ no Matlab, usando o algoritmo Polybases, procede-se da seguinte forma:

$$\begin{aligned} >> A = [&0\ 0\ 2\ 1\ 0\ 1\ 1\ -1\ 0\ 5\ -3\ 2\ 3\ 1\ 1; \\ &0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 5\ -3\ -2\ 1; \\ &0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 0\ 0\ 10\ -6\ -4\ 2]; \end{aligned}$$

$>> F = \texttt{polybases}(A, 2, 3);$

ou

$>> F = \texttt{polybases}(A, 2);$

Na execução do Passo 1 do algoritmo, como o o posto da matriz $A(s)$ é dois então a dimensão do espaço nulo a direita de $A(s)$ será um. Para este exemplo tem-se que para as matrizes $C_{\nu_i}(A), \nu_i = 0, 1, \ldots, 4$, terão $n_{\nu_i} = 0$, isto é, $C_{\nu_i}(A)$ tem posto cheio e, portanto, não haverá vetores à direita tais que $C_{\nu_i}(A)\underline{f}^{(i)} = \underline{0}$. Para $\nu = 5, n_{\nu_i} = 1$ uma vez que os valores singulares de $C_{\nu_i}(A)$ são

$$\Sigma_{\varphi_i} = [17.3215, 16.0847, 14.5737, 12.9081, 11.5063, 10.9440, 10.3592, 9.0833, \ldots$$

$$7.7532, 5.9452, 5.1358, 4.6514, 3.8374, 1.4487, 1.3466, 0.3780, 0.0267, 0.0000].$$

Logo executando o Passo 6:, tem-se que o vetor $\underline{f}^{(1)}$ será formado pela coluna dezoito de V_{ν_i} , sendo dado por:

$$f^{(1)} = [-0.0000, 0.0196, -0.0000, -0.0587, 0.2350, 0.0000, 0.1175, 0.2741, \ldots$$

$$0.0392, 0.2937, 0.5679, 0.0979, -0.1175, 0.5091, 0.3720, -0.0587, 0.1175, 0.0587].$$

Então a matriz coeficiente F da matriz polinomial $F(s)$ será

$$F = \begin{bmatrix} -0.0000 & -0.0587 & 0.1175 & 0.2937 & -0.1175 & -0.0587 \\ 0.0196 & 0.2350 & 0.2741 & 0.5679 & 0.5091 & 0.1175 \\ -0.0000 & 0.0000 & 0.0392 & 0.0979 & 0.3720 & 0.0587 \end{bmatrix},$$

e, portanto,

$$F(s) = \begin{bmatrix} -0.0587s^4 + 0.1175s^3 + 0.2937s^2 - 0.1175s - 0.0587 \\ 0.0196s^5 + 0.2350s^4 + 0.2741s^3 + 0.5679s^2 + 0.5091s + 0.1175 \\ 0.0392s^3 + 0.0979s^2 + 0.3720s + 0.0587 \end{bmatrix}.$$

□

Exemplo 4.5 *Considere , agora, uma interconexão massa-mola visto em Zúñiga e Henrion (2005) que foi teoricamente descrito em Zúñiga e Henrion (2004). Foi aplicado o algoritmo para achar a matriz da função de transferência da força agindo sobre a primeira massa para a posição da última massa. Resumidamente, este problema pode ser descrito por um sistema SISO modelado por uma equação diferencial de segunda ordem, dada por:*

$$M\frac{d^2}{dt^2}x + Kx = Bu \tag{4.14}$$

e cuja saída é

$$y = Cx$$

A equação (4.14) pode ser escrita da seguinte forma

$$D(s)x = Bu, \text{ onde } D(s) = Ms^2 + K$$

Assim, a função de transferência do sistema é dada por

$$G(s) = CD^{-1}(s)B = C\frac{N(s)}{d(s)} = \frac{n(s)}{d(s)},$$

e segue que

$$D^{-1}(s)B = \frac{N(s)}{d(s)}, \quad D(s)N(s) = Bd(s).$$

E é possível obter a função de transferência do sistema achando o espaço nulo da relação

$$[D(s) \quad -B][\times \quad \times \quad \dots \quad n(s) \quad d(s)]^T = 0 \tag{4.15}$$

Suponha agora que

$$D(s) = \begin{bmatrix} 1+s^2 & -1 & 0 \\ -1 & 2+s^2 & -1 \\ 0 & -1 & 2+s^2 \end{bmatrix} \text{ e } B = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

logo, o objetivo é achar uma base polinomial mínima para o espaço nulo à direita de

$$A(s) = [D(s) \quad -B] = \begin{bmatrix} 1+s^2 & -1 & 0 & -1 \\ -1 & 2+s^2 & -1 & 0 \\ 0 & -1 & 2+s^2 & 0 \end{bmatrix}.$$

Claramente, pode ser observado que a matriz $A(s)$ tem posto igual ao da matriz $D(s)$ e, portanto, igual a três. Conseqüentemente, a matriz $F(s)$ será composta de apenas um vetor. Então, aplicando o algoritmo, é obtido o seguinte resultado para F:

$$F = \begin{bmatrix} 0 & -0.0000 & 0.1026 & 0.0000 & 0.4104 & 0.0000 & 0.3078 \\ 0.0000 & -0.0000 & 0.0000 & 0.0000 & 0.1026 & 0.0000 & 0.2052 \\ 0.0000 & -0.0000 & -0.0000 & -0.0000 & 0.0000 & 0.0000 & 0.1026 \\ 0.1026 & 0.0000 & 0.5130 & 0.0000 & 0.6156 & 0.0000 & 0.1026 \end{bmatrix}$$

com $F(s)$ igual a

$$F(s) = \begin{bmatrix} 0.1026s^4 + 0.4104s^2 + 0.3078 \\ 0.1026s^2 + 0.2052 \\ 0.1026 \\ 0.1026s^6 + 0.5130s^4 + 0.6156s^2 + 0.1026 \end{bmatrix}. \tag{4.16}$$

Conseqüentemente, a função de transferência do sistema massa-mola será

$$G(s) = \frac{n(s)}{d(s)} = \frac{0.1026}{0.1026s^6 + 0.5130s^4 + 0.6156s^2 + 0.1026}$$

□

Exemplo 4.6 *Considere um sistema linear multivariável representado por uma DFM à esquerda, não necessariamente coprima,*

$$H(s) = \tilde{M}^{-1}(s)\tilde{N}(s).$$

É possível mostrar (Basilio e Moreira 2004), que uma DFM coprima à direita pode ser obtida via cálculo do espaço nulo, descrito por

$$[-\tilde{N}(s) \quad \tilde{M}(s)] \begin{bmatrix} M(s) \\ N(s) \end{bmatrix} = 0.$$

De forma análoga, é possível também, a partir de uma DFM à direita calcular uma DFM coprima à esquerda através da seguinte relação:

$$[-N^T(s) \quad M^T(s)] \begin{bmatrix} \tilde{M}^T(s) \\ \tilde{N}^T(s) \end{bmatrix} = 0.$$

□

4.2 Algoritmos Principais

Nesta seção serão descritos os algoritmos que inspiraram este trabalho.

4.2.1 Inversa de matriz polinomial

Entre os tópicos relativos a matrizes polinomiais, o cálculo da inversa tem tido grande interesse recentemente (Inouye 1979, Buslowicz 1980, Zhang 1987). Inouye (1979) propõe um algoritmo com a restrição da matriz a ser invertida ser própria por coluna [1]. Esta restrição foi removida por Buslowicz (1980), porém sem garantir que a matriz do numerador da inversa seja irredutível e que o polinômio do denominador da matriz inversa seja mínimo. Este problema foi solucionado por Zhang (1987), mas apenas para matrizes polinomiais reduzidas por coluna. Todos este algoritmos têm, em comum, o fato de se basearem um fórmulas recursivas, que são mais eficientes para matrizes de baixa ordem.

Nesta seção será utilizado um algoritmo que foi inicialmente proposto Basilio (2002), que também requer que a matriz a ser invertida seja reduzida por coluna. Ele utiliza como base realizações em espaço de estados.

Contudo é possível, caso a matriz não seja reduzida por coluna, utilizar o algoritmo que foi apresentado na seção 4.1.1 para redução por coluna.

Descrição do algoritmo

Para uma matriz $M(s) \in \mathbb{R}^{m\times m}[s]$, o problema de se calcular a inversa desta matriz pode ser descrito da seguinte maneira: encontrar uma matriz $H(s) \in \mathbb{R}^{m\times m}(s)$ tal

[1] Uma matriz é dita ser própria por coluna se a matriz coeficiente da potência de s de maior grau tiver posto cheio

que

$$H(s)M(s) = I_m,$$

onde I_m denota a matriz identidade de ordem m. Supondo que $M^{-1}(s)$ existe, então $H(s)$ pode ser escrita como:

$$H(s) = I_m M^{-1}(s) \tag{4.17}$$

Note que $H(s)$ é uma matriz de transferência definida por uma DFM em que as matrizes do numerador e denominador são, respectivamente, a matriz identidade de ordem m e a matriz a ser invertida $M(s)$. O seguinte resultado pode ser enunciado.

Proposição 4.1 *$H(s) = I_m M^{-1}(s)$ é uma DFM irredutível de $H(s)$*

Prova : Ver Basilio (2002). □

Com isso em mente, uma realização mínima $H(s)$ pode ser obtida de acordo com seguinte resultado.

Proposição 4.2 *Qualquer realização de ordem igual ao grau do determinante da matriz denominador de uma DFM será mínima se e somente se a DFM for irredutível.*

Prova : Ver Kailath (1980, página 439). □

Portanto, o problema de encontrar a inversa da matriz polinomial $M(s)$ se resume a encontrar uma realização mínima para a DFM dada na equação (4.17). Quando $M(s)$ for reduzida por coluna, esta realização pode ser obtida diretamente (Kailath 1980, página 403), desde que as seguintes condições sejam satisfeitas:

1. $M(s)$ seja reduzida por coluna;
2. $H(s)$ seja estritamente própria.

Neste ponto será suposto que a condição 1 é satisfeita (caso esta condição não seja é possível aplicar o algoritmo 4.1 visto na seção 4.1.1, que faz a redução por coluna de uma matriz polinomial). Para a condição 2 ser satisfeita note que uma ou

mais colunas de $M(s)$ tiver graus das colunas iguais a zero, então $H(s) = I_m M^{-1}(s)$ não será estritamente própria. Quando isto acontece, é necessário primeiro pós-multiplicar $M(s)$ por uma matriz diagonal $\Delta(s)$ cujos elementos da diagonal são s ou 1, se o correspondente grau das colunas de $M(s)$ são 0 ou maior do que 0, respectivamente. Por exemplo, seja $M(s) \in \mathbb{R}^{2\times 2}[s]$ e suponha que seu graus das colunas sejam 1 e 0. Neste caso $\Delta(s) = diag\{1, s\}$.

A pós-multiplicação de $M(s)$ por uma matriz diagonal $\Delta(s)$, definida anteriormente, implica que a matriz polinomial cuja inversa será calculada, será dada por:

$$\tilde{M}(s) = M(s)\Delta(s).$$

Do ponto de vista da computacional isto não apresenta nenhum problema, uma vez que:

1. $\tilde{M}(s)$ e $M(s)$ têm a mesma matriz M_{hc};
2. $det[\tilde{M}(s)] = det[M(s)]s^{n_0}$, onde n_0 é o número de colunas de $M(s)$ com graus iguais a zero;
3. $M^{-1}(s) = \Delta(s)\tilde{M}^{-1}(s)$.

Com isto em mente, uma realização de ordem mínima $\tilde{M}^{-1}(s)$ pode ser obtida de acordo com o seguinte algoritmo.

Algoritmo 4.6 *:*

Passo 1:*: Encontre matrizes constantes M_{hc} e M_{lc} tais que*

$$M(s) = M_{hc}S(s) + M_{lc}\Psi(s). \tag{4.18}$$

Passo 2: *Forme as matrizes A_{c0} e B_{c0} como foi mostrado na seção 2.7 nas equações (2.28) e (2.30) e calcule as matrizes A_c e B_c:*

$$\begin{cases} A_c = A_{c0} - B_{c0}M_{hc}^{-1}M_{lc} \\ B_c = B_{c0}M_{hc}^{-1} \end{cases} \tag{4.19}$$

Passo 3: *Escreva I_m, a matriz numerador de $H(s)$, como*

$$I_m = N_{lc}\Psi(s) \tag{4.20}$$

e obtenha

$$C_c = N_{lc}. \tag{4.21}$$

Passo 4: *Ache um realização de* $[A_c, B_c, C_c]$. □

Observação 4.3 *O algoritmo 4.6 proporciona uma maneira simples e direta de ser encontrar uma realização de ordem mínima da inversa de uma matriz polinomial reduzida por coluna de* $M(s)$*. Além disso, note que, como*

$$M^{-1}(s) = C_c(sI - A_c)^{-1}B_c$$

então a representação na forma de função de transferência de $M^{-1}(s)$ *pode ser obtida através do algoritmo 4.2 apresentado na seção 4.1.2.* □

Caso a matriz a ser invertida não seja reduzida por coluna o algoritmo 4.1 visto na seção 4.1.1, pode ser utilizado, permitindo a obtenção da matriz $\overline{M}(s)$ (reduzida por coluna) e da matriz unimodular $U(s)$, que satisfazem à seguinte relação:

$$\overline{M}(s) = M(s)U(s), \tag{4.22}$$

onde $M(s)$ é uma matriz polinomial qualquer.

Seja

$$\overline{H}(s) = \overline{M}^{-1}(s)$$

e note que, como $\overline{M}(s)$ é reduzida por coluna, então o algoritmo 4.6 pode ser aplicado para calcular $\overline{H}(s)$. Além disso, pela equação (4.24), pode-se verificar que:

$$M^{-1}(s) = U(s)\overline{M}^{-1}(s), \tag{4.23}$$

e, portanto:

$$H(s) = U(s)\overline{H}(s).$$

O cálculo da inversa de matrizes polinomiais que não são reduzidas por coluna pode ser resumido no seguinte algoritmo:

Algoritmo 4.7 *:*

Passo 1: *Use o algoritmo 4.1 para calcular* $\overline{M}(s)$ *e* $U(s)$.

Passo 2: *Use o algoritmo 4.6 para calcular*

$$\overline{H}(s) = \overline{M}^{-1}(s).$$

Passo 3: *Calcule*

$$H(s) = U(s)\overline{H}(s).$$

□

A função `inverse`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `inverse.m` (apêndice A.20) . Para usá-la, o usuário deve fornecer a matriz coeficientes associada a uma matriz polinomial $M(s)$. A função retorna a matriz de coeficiente N associado à matriz polinomial $N(s)$ e o vetor d associado ao polinômio $d(s)$ de $M^{-1}(s)$ da matriz racional $H(s) = I_m M^{-1}(s) = N(s)/d(s)$. Essa função tem a seguinte sintaxe:

$$[N, d] = \texttt{inverse}(M),$$

onde M é a matriz coeficiente associada a uma matriz polinomial $M(s)$, N é a matriz coeficiente associada a $N(s)$ e d é o vetor associado ao polinômio $d(s)$.

Exemplo 4.7 *Seja a inversão da matriz polinomial* $M(s)$*,reduzida por coluna, dada por:*

$$M(s) = \begin{bmatrix} -1.0150 & -0.5883 & 5.0000 \\ -0.7250 & 0 & 0 \\ -0.7250 & 0 & 5.0000 \end{bmatrix} s^2 + \begin{bmatrix} -5.4012 & -0.7845 & -3.0000 \\ -2.2475 & -0.3922 & 0 \\ -1.4500 & -0.9806 & 0 \end{bmatrix} s + \begin{bmatrix} -0.5437 & -6.8641 & 1.0000 \\ -0.5437 & -0.9806 & -2.0000 \\ -1.6312 & -1.9612 & 0 \end{bmatrix} \quad (4.24)$$

Uma realização de ordem mínima para

$$H(s) = M^{-1}(s),$$

pode ser obtida utilizando a função `inverse` *no Matlab. Para tanto, procede-se da seguinte forma:*

$$>> M = [-1.0150 \quad -0.5883 \quad 5 \quad -5.4012 \quad -0.7845 \quad -3 \quad -0.5437 \quad -6.8641 \quad 1$$
$$-0.7250 \quad 0 \quad 0 \quad -2.2475 \quad -0.3922 \quad 0 \quad -0.5437 \quad -0.9806 \quad -2$$
$$-0.7250 \quad 0 \quad 5 \quad -1.4500 \quad -0.9806 \quad 0 \quad -1.6312 \quad -1.9612 \quad 0]$$

$>> [N, d] = inverse(M);$

Executando o Passo 1 do algoritmo, tem-se:

$$M(s) = M_{hc} \begin{bmatrix} s^2 & 0 & 0 \\ 0 & s^2 & 0 \\ 0 & 0 & s^2 \end{bmatrix} + M_{lc} \begin{bmatrix} s & 0 & 0 \\ 1 & 0 & 0 \\ 0 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \\ 0 & 0 & 1 \end{bmatrix},$$

onde

$$M_{hc} = \begin{bmatrix} -1.0150 & -0.5883 & 5.0000 \\ -0.7250 & 0 & 0 \\ -0.7250 & 0 & 5.0000 \end{bmatrix}$$

e

$$M_{lc} = \begin{bmatrix} -5.4012 & -0.5437 & -0.7845 & -6.8641 & -3.0000 & 1.0000 \\ -2.2475 & -0.5437 & -0.3922 & -0.9806 & 0 & -2.0000 \\ -1.4500 & -1.6312 & -0.9806 & -1.9612 & 0 & 0 \end{bmatrix}$$

Na execução do Passo 2 tem-se, de acordo com a equação (2.29) (seção 2.7) a seguinte matriz $A_{c0}^{(i)}$*, sabendo que os graus* $\nu_i, i = 1, 2, 3$ *das colunas de* $M(s)$ *são todos iguais a dois*

$$A_{c0}^{(1)} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}, A_{c0}^{(2)} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \text{ e } A_{c0}^{(3)} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}.$$

Assim, de acordo com a equação (2.28), pode escrever:

$$A_{c0} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Para a formação da matriz $B_{c0}^{(i)}$ *vista na equação (2.30), tem-se:*

$$B_{c0}^{(1)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, B_{c0}^{(2)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ e } B_{c0}^{(3)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

e, então,

$$B_{c0} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Assim, é possível calcular as matrizes

$$A_c = A_{c0} - B_{c0}M_{hc}^{-1}M_{lc} =$$

$$\begin{bmatrix} -3.1000 & -0.7500 & -0.5410 & -1.3525 & 0.0000 & -2.7586 \\ 1.0000 & 0 & 0 & 0 & 0 & 0 \\ -5.1878 & 2.2181 & 0.6000 & -7.6667 & -5.0990 & 3.0594 \\ 0 & 0 & 1.0000 & 0 & 0 & 0 \\ -0.1595 & 0.2175 & 0.1177 & 0.1961 & 0 & -0.4000 \\ 0 & 0 & 0 & 0 & 1.0000 & 0 \end{bmatrix}$$

e

$$B_c = B_{c0}M_{hc}^{-1} = \begin{bmatrix} 0.0000 & -1.3793 & -0.0000 \\ 0 & 0 & 0 \\ -1.6997 & 0.6799 & 1.6997 \\ 0 & 0 & 0 \\ 0 & -0.2000 & 0.2000 \\ 0 & 0 & 0 \end{bmatrix}.$$

De acordo com o Passo 3 tem-se:

$$I_m = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \Psi(s) = \begin{bmatrix} s & 0 & 0 \\ 1 & 0 & 0 \\ 0 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \\ 0 & 0 & 1 \end{bmatrix},$$

e, desta forma:

$$C_c = N_{lc} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Para a realização do passo 4, deve ser utilizado o algoritmo 4.2 apresentado na seção 4.1.2. Note que as matrizes (A_c, B_c, C_c, D_c) são as mesmas utilizadas no exemplo 4.2 e, portanto o resultado final será:

$$M^{-1}(s) = \frac{N(s)}{d(s)}$$

onde

$$N(s) = \begin{bmatrix} 0.0000 & -1.3793 & -0.0000 \\ -1.6997 & 0.6799 & 1.6997 \\ 0 & -0.2000 & 0.2000 \end{bmatrix} s^4 + \begin{bmatrix} 0.9195 & 0.4598 & -0.9195 \\ -5.2690 & 10.2830 & 4.2492 \\ -0.2000 & -0.2000 & 0.7000 \end{bmatrix} s^3 +$$

$$\begin{bmatrix} 2.2989 & -12.8736 & -2.2989 \\ -1.9546 & -0.8498 & -0.5949 \\ -1.1000 & 0.1000 & 1.8500 \end{bmatrix} s^2 + \begin{bmatrix} 0.9195 & -2.2989 & 0.4598 \\ -1.3597 & 1.6147 & 5.3540 \\ -1.3500 & -0.0500 & 4.8500 \end{bmatrix} s+$$

$$\begin{bmatrix} 1.8391 & 0.9195 & -6.8966 \\ -1.5297 & -0.7649 & 0.7649 \\ 0.2500 & -4.7500 & 1.5000 \end{bmatrix} \tag{4.25}$$

e

$$d(s) = s^6 + 2.5s^5 + 4.75s^4 + 21s^3 + 11.5s^2 - 2s + 9.75. \tag{4.26}$$

□

Agora será exemplificado o cálculo da inversa polinomial de uma matriz não reduzida por coluna

Exemplo 4.8 *Para mostrar o funcionamento do algoritmo 4.7, seja a matriz polinomial*

$$M(s) = \begin{bmatrix} s^3 + s + 7 & 5s^2 - 3s + 1 & 2s^4 + s + 1 \\ 1 & -2 & 2s + 1 \\ s^3 + s + 2 & 5s^2 & 2s^4 + 2s^3 + 3 \end{bmatrix}.$$

Esta matriz é a mesma utilizada no exemplo 4.1, seção 4.1.1 que trata da redução por colunas de uma matriz polinomial. Logo executando o Passo 1 do algoritmo 4.7 serão obtidas as matrizes $\overline{M}(s)$ *e* $U(s)$ *vista em (4.4) e (4.1), respectivamente. Também pode ser observado que, para a execução do Passo 2 do algoritmo, a matriz de entrada do exemplo 4.7 desta seção é a matriz gerada com a redução por coluna dada em (4.4) onde o resultado da inversa é dada na equação (4.25) e (4.26). Portanto,*

$$M^{-1}(s) = \frac{1}{d(s)} N(s)$$

onde

$$N(s) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.5 & 0 \\ 0 & 0 & 0 \end{bmatrix} s^6 + \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & -0 \\ 0 & 0 & 0 \end{bmatrix} s^5 + \begin{bmatrix} 1 & -1 & -1 \\ 0 & -2.75 & 0 \\ 0 & 0.75 & 0 \end{bmatrix} s^4 +$$

$$\begin{bmatrix} 3.5 & -0.75 & -2.5 \\ 0.25 & -4 & 0.25 \\ -0.5 & -0.25 & 0.5 \end{bmatrix} s^3 + \begin{bmatrix} 1.25 & 2.5 & 0.25 \\ -0.5 & 0.25 & 0.5 \\ -1.25 & 7 & 1.25 \end{bmatrix} s^2 +$$

$$\begin{bmatrix} 0 & -2.25 & -0.25 \\ -1.25 & 0 & 3.5 \\ -0.5 & 1.25 & -0.25 \end{bmatrix} s + \begin{bmatrix} 1.5 & 0.75 & -0.75 \\ 0.25 & -4.75 & 1.5 \\ -1 & -0.5 & 3.75 \end{bmatrix}$$

e

$$d(s) = s^6 + 2.5s^5 + 4.75s^4 + 21s^3 + 11.5s^2 - 2s + 9.75$$

□

4.2.2 Determinante

Neste seção será apresentado um algoritmo que calcula o determinante de uma matriz polinomial. Este algoritmo tem como base os conceitos que foram comentados para o cálculo da inversa polinomial.

Descrição do algoritmo

Seja $M(s) \in \mathbb{R}^{m\times m}[s]$. Então, uma forma de calcular o seu determinante, conforme Kailath (1980, pág. 409) é

$$|M(s)| = |M_{hc}||sI - A_c|. \tag{4.27}$$

Assim, para se calcular o determinante de $M(s)$, deve-se encontrar uma realização de estados para esta matriz polinomial. Note, também, que a matriz deverá ser reduzida por coluna, isto é, $|M_{hc}| \neq 0$, caso ela não seja, deverá ser utilizado o algoritmo apresentado na seção 4.1.1.

Algoritmo 4.8 *Seja $M(s) \in \mathbb{R}^{m\times m}[s]$ reduzida por coluna.*
Passo 1:*:Encontre matrizes constantes M_{hc} e M_{lc} tais que*

$$M(s) = M_{hc}S(s) + M_{lc}\Psi(s); \tag{4.28}$$

Passo 2: *Forme as matrizes A_{c0} e B_{c0} como foi mostrado na seção 2.7 nas equações (2.28) e (2.30). E calcule as matrizes A_c:*

$$A_c = A_{c0} - B_{c0}M_{hc}^{-1}M_{lc}; \tag{4.29}$$

Passo 3:*Calcule*

$$|M(s)| = |M_{hc}||sI - A_c|.$$

□

Caso a matriz que se deseje calcular o determinante não seja reduzida por coluna, o algoritmo 4.1.1 deverá ser utilizado, conseqüentemente a relação dada na equação 4.27, será modificada para

$$|M(s)| = \frac{|\overline{M}_{hc}||sI - A_c|}{|U(s)|} \tag{4.30}$$

onde $U(s)$ é a matriz unimodular que foi utilizada para realizar a redução por coluna de $M(s)$ e $\overline{M}_{hc}$ foi a matriz gerada com esta redução.

Observação 4.4 *Quando a matriz $M(s)$ for unimodular a execução do algoritmo da redução por coluna fará com que a matriz gerada tenha grau 0. Conseqüentemente a matriz A_c será formada com blocos de zeros e a equação 4.33 ficará reduzida a*

$$|M(s)| = \frac{|\overline{M}_{hc}|}{|U(s)|}, \tag{4.31}$$

A função `polydet`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `polydet.m` (apêndice A.21) . Para usá-la, o usuário deve fornecer a matriz de coeficientes associada a uma matriz polinomial $M(s)$. A função retorna um vetor linha dos coeficientes do polinômio $d(s)$ que representa o determinante de $M(s)$. Essa função tem a seguinte sintaxe:

$$[d] = \texttt{polydet}(M),$$

onde M é a matriz de coeficientes associada a uma matriz polinomial $M(s)$, e d é o vetor linha associado ao polinômio $d(s)$.

Exemplo 4.9 *Como exemplo de aplicação do algoritmo, considere a matriz reduzida por coluna*

$$M(s) = \begin{bmatrix} s^2+1 & -6s \\ 1 & 5s^2 \end{bmatrix},$$

cuja matriz coeficiente é dada por:

$$M = \left[\begin{array}{cc|cc|cc} 1 & 0 & 0 & -6 & 1 & 0 \\ 0 & 5 & 0 & 0 & 1 & 0 \end{array}\right].$$

Então, para se calcular o determinante da matriz polinomial no Matlab, usando o algoritmo Polydet, procede-se da seguinte forma:

$>> M = [1 \quad 0 \quad 0 \quad -6 \quad 1 \quad 0; \quad 0 \quad 5 \quad 0 \quad 0 \quad 1 \quad 0];$

$>> [d] = polydet(M);$

Executando o Passo 1 do algoritmo, obtém-se:

$$M_{hc} = \left[\begin{array}{cc} 1 & 0 \\ 0 & 5 \end{array}\right]$$

e

$$M_{lc} = \left[\begin{array}{cccc} 0 & 1 & -6 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right].$$

Portanto, de acordo com o Passo 2 pode-se escrever:

$$A_c = A_{c0} - B_{c0} M_{hc}^{-1} M_{lc} = \left[\begin{array}{cccc} 0 & -1 & 6 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -0.2 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right].$$

Na execução do Passo 3 deve-se calcular o determinante $|sI - Ac|$, que pode ser feito através da função `poly` *do Matlab, procedendo da seguinte forma:*

$>> Ac = [0 \quad -1 \quad 6 \quad 0; \quad 1 \quad 0 \quad 0 \quad 0; \quad 0 \quad -0.2 \quad 0 \quad 0; \quad 0 \quad 0 \quad 1 \quad 0];$

$>> [da] = \texttt{poly}(Ac);.$

Resultando:

$$da = \left[\begin{array}{cccc} 1 & 0 & 1 & 1.2 \quad 0 \end{array}\right],$$

e finalmente, para calcular $d = |M_{hc}||sI - A_c|$, procede-se da seguinte forma:

$>> d = det([1 \;\; 0; \;\; 0 \;\; 5]) * da,$

resultando no seguinte polinômio que é o determinante de $M(s)$:

$$d(s) = 5s^4 + 5s^2 + 6s.$$

Exemplo 4.10 *Para descrevermos os passos do algoritmo para matrizes não reduzida por coluna, será utilizada a mesma matriz dos exemplos 4.8 e 4.1, qual seja:*

$$M(s) = \begin{bmatrix} s^3+s+7 & 5s^2-3s+1 & 2s^4+s+1 \\ 1 & -2 & 2s+1 \\ s^3+s+2 & 5s^2 & 2s^4+2s^3+3 \end{bmatrix}.$$

Note que no exemplo 4.8 a matriz do denominador é dada por

$$|sI - Ac| = d(s) = s^6 + 2.5s^5 + 4.75s^4 + 21s^3 + 11.5s^2 - 2s + 9.75$$

e, no exemplo 4.1, é possível verificar que

$$\overline{M}_{hc} = \begin{bmatrix} -1.0150 & -0.5883 & 5.0000 \\ -0.7250 & 0 & 0 \\ -0.7250 & 0 & 5.0000 \end{bmatrix}$$

e

$$U(s) = \begin{bmatrix} 0 & 0 & 0 \\ 0.3625 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} s^2 + \begin{bmatrix} -0.7250 & 0 & 0 \\ 0.2175 & 0.1961 & 0 \\ 0 & 0 & 0 \end{bmatrix} s + \begin{bmatrix} 0 & -0.9806 & 0 \\ 0 & 0 & 1.0000 \\ -0.5437 & 0 & 0 \end{bmatrix} \quad (4.32)$$

Logo,

$$|M(s)| = \frac{|\overline{M}_{hc}||sI - A_c|}{|U(s)|} = -4s^6 - 10s^5 - 19s^4 - 84s^3 - 46s^2 + 8s - 39 \quad (4.33)$$

4.2.3 Descrição por Frações de Matrizes(DFM) à direita e à esquerda

Um DFM pode ser descrita, de acordo com o exposto no capítulo 2, da seguinte forma:

$$H(s) = N(s)D^{-1}(s) = \tilde{D}^{-1}(s)\tilde{N}(s). \quad (4.34)$$

O objetivo desta seção é descrever um algoritmo que, de qualquer DFM não coprima de entrada gere na sua saída uma DFM coprima à direita. Também, é proposto um algoritmo que de um DFM coprima à direita resulte em um DFM coprima à esquerda.

Descrição do algoritmo

Seja $H(s) \in \mathbb{R}^{p\times m}(s)$ e suponha que $H(s)$ seja descrita pela seguinte equação:

$$H(s) = \frac{N_H(s)}{d(s)} \tag{4.35}$$

onde $N_H(s) \in \mathbb{R}^{p\times m}[s]$ e $d(s)$ denota o polinômio definido como o mínimo múltiplo comum de todos os denominadores de $H(s)$. Suponha, ainda, que $H(s)$ seja própria, isto é,:

$$\lim_{s\to\infty} H(s) = H_\infty \in \mathbb{R}^{p\times m}.$$

Seja uma MFD não coprima à esquerda de $H(s)$ dada por:

$$H(s) = \tilde{A}^{-1}(s)\tilde{B}(s),$$

que de acordo com a equação 4.35, pode ser definida como:

$$\tilde{A}(s) = d(s)I_p \text{ e } \tilde{B}(s) = N_H(s)$$

Seja

$$H(s) = N(s)M^{-1}(s)$$

uma DFM coprima à direita para $H(s)$. Então as matrizes polinomiais $A(s), B(s)$,$N(s)$ e $M(s)$ devem satisfazer:

$$\begin{bmatrix} \tilde{B}(s) & -\tilde{A}(s) \end{bmatrix} \begin{bmatrix} M(s) \\ N(s) \end{bmatrix} = 0. \tag{4.36}$$

Observando a equação (4.36), é possível encontrar a DFM coprima à direita computando-se a base polinomial mínima da matriz

$$T_1(s) = \begin{bmatrix} \tilde{B}(s) & -\tilde{A}(s) \end{bmatrix}. \tag{4.37}$$

Note, ainda que, a equação 4.36 provê, também, uma maneira sistemática de se obter uma DFM coprima à esquerda. Para tanto observe que:

$$N(s)M^{-1}(s) = \tilde{M}^{-1}(s)\tilde{N}(s) \Leftrightarrow N(s)M^{-1}(s) - \tilde{M}^{-1}(s)\tilde{N}(s) = 0,$$

logo

$$\begin{bmatrix} -\tilde{N}(s) & \tilde{M}(s) \end{bmatrix} \begin{bmatrix} M(s) \\ N(s) \end{bmatrix} = 0 \Longrightarrow \begin{bmatrix} -N^T(s) & M^T(s) \end{bmatrix} \begin{bmatrix} \tilde{M}^T(s) \\ \tilde{N}^T(s) \end{bmatrix} = 0. \tag{4.38}$$

Assim, de posse de uma DFM à direita é, também, possível encontrar uma DFM coprima à esquerda computando-se uma base polinomial mínima para o espaço nulo da matriz polinomial

$$T_2 = \left[\begin{array}{cc} -N^T(s) & M^T(s) \end{array} \right]. \tag{4.39}$$

Assim, será apresentado um algoritmo para encontrar uma DFM coprima à direita a partir de uma DFM a esquerda não coprima.

Algoritmo 4.9 *:*

Passo 1: *Utilize o algoritmo 3.1 para gerar a matriz coeficiente da matriz (4.37);*

Passo 2: *Utilize o algoritmo 4.5 para calcular uma base polinomial mínima para o espaço nulo à direita de* $T_1(s)$. □

Conforme exposto acima, é possível criar um algoritmo, semelhante ao anterior, que de uma DFM coprima à direita achar uma DFM coprima à esquerda utilizando a relação dada em (4.38).

Algoritmo 4.10 *:*

Passo 1: *Ache a transposta das matrizes coprimas a direita* $M(s)$ *e* $N(s)$ *utilizando o algoritmo dado na seção 3.3.1 e forme a matriz de coeficientes da matriz* T_2 *dada em (4.39).*

Passo 2: *Utilize o algoritmo 4.5 para calcular uma base polinomial mínima para o espaço nulo à direita de* $T_2(s)$.

Passo 3: *Utilizando o algoritmo para transposição de matrizes obtenha* $\tilde{M}$ *e* $\tilde{N}$. □

A função `rcmfd`

Uma das funções Matlab desenvolvidas para implementar os algoritmos descritos nesta seção é a `rcmfd.m` (apêndice A.22) . Para usá-la, o usuário deve fornecer às matrizes coeficientes associadas as matrizes polinomiais $\tilde{A}(s)$ e $\tilde{B}(s)$ geradas pela equação $H(s) = \tilde{A}^{-1}(s)\tilde{B}(s)$ e com o número de colunas de $\tilde{B}(s)$ (caso a matriz não seja quadrada). A função retorna as matrizes coeficientes coprimas M e N associadas às matrizes polinomiais $M(s)$ e $N(s)$ da DFM coprima à direita de $H(s)$, isto

é, $H(s) = N(s)M^{-1}(s)$. Essa função tem a seguinte sintaxe:

$$[M, N] = \texttt{rcmfd}(B, A, Bncol),$$

onde M e N são as matrizes coeficientes associadas, respectivamente às matrizes polinomiais $M(s)$ e $N(s)$, A e B são as matrizes coeficientes associadas, respectivamente às matrizes polinomiais $A(s)$ e $B(s)$ e $Bncol$ é o número de colunas de $\tilde{B}(s)$.

A função `lcmfd`

Uma das funções Matlab desenvolvidas para implementar os algoritmos descritos nesta seção é a `lcmfd.m` (apêndice A.23) . Para usá-la, o usuário deve fornecer as matrizes coeficientes associadas às matrizes polinomiais $M(s)$ e $N(s)$ que satisfazem a equação $H(s) = N(s)M^{-1}(s)$. A função retorna as matrizes coeficientes $\tilde{M}$ e $\tilde{N}$ coprimas à esquerda associadas, respectivamente às matrizes polinomiais $\tilde{M}(s)$ e $\tilde{N}(s)$ que satisfazem a DFM $H(s) = \tilde{M}^{-1}(s)\tilde{N}(s)$. Essa função tem a seguinte sintaxe:

$$[Mtil, Ntil] = \texttt{lcmfd}(M, N),$$

onde $Mtil$ e $Ntil$ são as matrizes coeficientes associadas, respectivamente às matrizes polinomiais $\tilde{M}(s)$ e $\tilde{N}(s)$, M e N são as matrizes coeficientes associadas, respectivamente às matrizes polinomiais coprimas à direita $M(s)$ e $N(s)$.

Exemplo 4.11 *Seja a matriz de transferência apresentada em Kailath (1980, página 368)*

$$H(s) = \begin{bmatrix} \frac{s}{(s+1)^2(s+2)^2} & \frac{s}{(s+2)^2} \\ \frac{-s}{(s+2)^2} & \frac{-s}{(s+2)^2} \end{bmatrix} = A^{-1}(s)B(s), \tag{4.40}$$

em que uma DFM à esquerda é:

$$H(s) = \begin{bmatrix} (s+1)^2(s+2)^2 & 0 \\ 0 & (s+2)^2(s+1)^2 \end{bmatrix}^{-1} \begin{bmatrix} s & s(s+1)^2 \\ -s(s+1)^2 & -s(s+1)^2 \end{bmatrix}.$$

É imediato verificar que as matrizes coeficientes são dadas por:

$$B = \left[\begin{array}{cc|cc|cc|cc} 0 & 1 & 0 & 2 & 1 & 1 & 0 & 0 \\ -1 & -1 & -2 & -2 & -1 & -1 & 0 & 0 \end{array}\right]$$

e

$$A = \left[\begin{array}{cc|cc|cc|cc|cc} 1 & 0 & 6 & 0 & 13 & 0 & 12 & 0 & 4 & 0 \\ 0 & 1 & 0 & 6 & 0 & 13 & 0 & 12 & 0 & 4 \end{array}\right].$$

Um exemplo de utilização da função `rcmfd` *no Matlab é o seguinte:*

$>> A = [1 \quad 0 \quad 6 \quad 0 \quad 13 \quad 0 \quad 12 \quad 0 \quad 4 \quad 0;\ 0 \quad 1 \quad 0 \quad 6 \quad 0 \quad 13 \quad 0 \quad 12 \quad 0 \quad 4];$

$>> B = [0 \quad 1 \quad 0 \quad 2 \quad 1 \quad 1 \quad 0 \quad 0;\ -1 \quad -1 \quad -2 \quad -2 \quad -1 \quad -1 \quad 0 \quad 0];$

$>> [M, N] =$ `rcmfd`$(B, A, 3);$

ou

$>> [M, N] =$ `rcmfd`$(B, A);$.

Executando o Passo 1 do algoritmo obtém-se:

$$T_1 = \left[\begin{array}{cccc|cccc|} 0 & 0 & -1 & 0 & 0 & 1 & -6 & 0 \\ 0 & 0 & 0 & -1 & -1 & -1 & 0 & -6 \end{array}\right.$$

$$\left.\begin{array}{cccc|cccc|cccc} 0 & 2 & -13 & 0 & 1 & 1 & -12 & 0 & 0 & 0 & -4 & 0 \\ -2 & -2 & 0 & -13 & -1 & -1 & 0 & -12 & 0 & 0 & 0 & -4 \end{array}\right].$$

De acordo com o Passo 2, deve-se calcular uma base polinomial para o espaço nulo à direita de $T_1(s)$*. Assim:*

$$F = \left[\begin{array}{cccccccc} 0 & 0.0528 & 0.0000 & 0.2110 & 0.0000 & 0.2638 & 0.0000 & 0.1055 \\ 0 & 0.0896 & 0.1690 & 0.1901 & 0.6761 & -0.3680 & 0.6761 & -0.7793 \\ 0 & -0.0000 & -0.0000 & 0.0896 & 0.1690 & -0.1684 & -0.0000 & 0.0000 \\ 0 & 0.0000 & -0.0000 & -0.1424 & -0.1690 & 0.1684 & -0.0000 & -0.0000 \end{array}\right].$$

Finalmente, no Passo 3 , separando as matrizes M *e* N *obtém-se as seguintes matrizes:*

$$M = \left[\begin{array}{cc|cc|cc|cc} 0 & 0.0528 & 0.0000 & 0.2110 & 0.0000 & 0.2638 & 0.0000 & 0.1055 \\ 0 & 0.0896 & 0.1690 & 0.1901 & 0.6761 & -0.3680 & 0.6761 & -0.7793 \end{array}\right]$$

e

$$N = \left[\begin{array}{cc|cc|cc} 0.0000 & 0.0896 & 0.1690 & -0.1684 & -0.0000 & 0.0000 \\ 0.0000 & -0.1424 & -0.1690 & 0.1684 & -0.0000 & -0.0000 \end{array}\right].$$

□

Observação 4.5 *O determinante da matriz* $M(s)$ *é dado por:*

$$det(M) = (s+2)^3(s+1)^2.$$

Conforme exposto em Kailath (1980, página 369) o grau mínimo para qualquer DFM de $H(s)$ deveria ter grau 5. Isto, de fato, ocorre com a DFM coprima gerada pelo algoritmo. O exemplo 4.11 revela o potencial do algoritmo desta seção, mostrando que dada uma DFM qualquer de um sistema, é possível obter uma DFM equivalente de ordem mínima. □

Em Garcia e Basilio (2000) é feito um algoritmo que faz a redução de sistemas através de truncamento balanceado. Nele é apresentado um exemplo em que a matriz de estados A possui antes da aplicação do algoritmo ordem 8 e, após aplicação do mesmo a ordem do sistema passa a ser 4. Assim, qualquer DFM para o sistema desse exemplo deverá ter ordem 4, conforme será verificado a seguir.

Exemplo 4.12 *Seja a matriz $H(s)$ do exemplo da aplicação do algoritmo de Garcia e Basilio (2000)*

$$H(s) = \frac{\begin{bmatrix} s+1 & (s+1)(2s+1) & s(s+1) \\ s+2 & (s+2)(s^2+5*s+3) & s(s+2) \\ 1 & 2s+1 & s \end{bmatrix}}{d(s)}, \tag{4.41}$$

com

$$d(s) = (s+1)^2(s+2). \tag{4.42}$$

Em Garcia e Basilio (2000), a matriz gerada após a aplicação do algoritmo é:

$$A = \begin{bmatrix} -1.19912 & 1.17669 & -0.20410 & 0.08115 \\ -0.22153 & -0.61453 & -0.23408 & 0.09276 \\ -1.20966 & 0.99303 & -2.28659 & 0.50952 \\ -0.23454 & -0.34857 & -0.25357 & -0.89976 \end{bmatrix},$$

cujo polinômio característico é dado por:

$$d(s) = s^4 + 5s^3 + 9s^2 + 7s + 2,$$

com autovalores dados por -2 e -1 (multiplicidade 3).

Agora aplicando o algoritmo 4.9 serão obtidas as seguintes matrizes

$$N = \begin{bmatrix} 0 & 0 & 0.0000 & 0.0000 & -0.0000 & 0.0574 & 0.0000 & 0.0000 & 0.0574 \\ 0 & 0 & 0.0062 & 0.1826 & -0.1651 & 0.1764 & 0.3653 & -0.3301 & 0.3281 \\ 0 & 0 & -0.0000 & -0.0000 & -0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0574 \end{bmatrix} \tag{4.43}$$

e

$$M = \begin{bmatrix} 0 & 0 & 0.2751 & 0.2001 & 0.7722 & -0.5991 & -0.1826 & 0.1651 & 0.0082 \\ 0 & 0 & 0.0062 & 0.1826 & -0.1651 & 0.1128 & 0.1826 & -0.1651 & 0.1066 \\ 0 & 0 & 0.0450 & -0.3653 & 0.3301 & -0.2773 & -0.7481 & -0.2770 & 0.5601 \end{bmatrix}. \tag{4.44}$$

Note que o determinante da matriz $M(s)$ é dado por:

$$d(s) = s^4 + 5s^3 + 9s^2 + 7s + 2,$$

mostrando que a resposta é a mesma que a dada em Garcia e Basilio (2000). Portanto a matrizes geradas utilizando-se o algoritmo desta seção permite, também, que se obtenha uma representação em espaço de estados de ordem mínima. □

Exemplo 4.13 *Para mostrar a aplicação do algoritmo 4.10, que calcula a matriz coprima à esquerda, será utilizada a matriz de transferência (4.41)em que a DFM coprima à direita é dada pelas matrizes de coeficientes (4.43) e (4.44). O resultado da aplicação do algoritmo é*

$$\tilde{M} = \begin{bmatrix} 0 & 0 & 0 & 0.1864 & -0.1864 & \dots \\ 0 & 0 & 0 & -0.2429 & 0.2429 & \dots \\ 0 & 0.1263 & 0.2240 & -0.0375 & 0.0662 & \dots \end{bmatrix}$$

$$\begin{bmatrix} \dots & 0.5823 & -0.2094 & -0.1864 & 0.5823 \\ \dots & 0.1896 & -0.6754 & 0.2429 & 0.1896 \\ \dots & -0.1333 & 0.7301 & -0.0601 & -0.3573 \end{bmatrix}$$

e

$$\tilde{N} = \begin{bmatrix} 0 & 0 & 0 & 0 & -0.1864 & -0.0000 & -0.0000 & -0.3729 & 0 \\ 0 & 0 & 0 & 0 & 0.2429 & -0.0000 & -0.0000 & 0.4858 & 0 \\ 0 & 0.1263 & 0 & 0 & 0.4451 & 0.1263 & 0.1263 & 0.0061 & 0 \end{bmatrix},$$

e o polinômio característico obtido calculando-se o determinante de $\tilde{M}(s)$ será dado por:

$$d(s) = s^4 + 5s^3 + 9s^2 + 7s + 2,$$

como esperado. □

4.2.4 Máximo Divisor Comum de matrizes polinomiais

O problema de calcular o MDC para matrizes polinomiais tem atraído atenção de alguns pesquisadores (Kaylath et al. 1978) e tem muitas aplicações na teoria de

controle ou em sistemas lineares (Kailath 1980). Por exemplo, a partir do cálculo do MDC de matrizes polinomiais, pode-se obter uma DFM irredutível (e, portanto, uma realização em espaço de estados mínima) de um matriz de função de transferência. Outro exemplos são o estudo de zeros de desacoplamentos e modos não-controláveis e não-observáveis de um dado sistema; e obtenção da estrutura de pólos e zeros de um dado sistema multivariável. MDC são claramente não únicos, já que pode a multiplicação deles por matrizes unimodulares não alteram as suas características de sistema.

Descrição do algoritmo do Máximo Divisor Comum de matrizes polinomiais (MDC)

O algoritmo que será apresentado nesta seção é baseado nos algoritmos propostos na seção anterior. Sejam as DFM de uma matriz $H(s) \in \mathbb{R}^{p\times m}(s)$ dadas por:

$$H(s) = B(s)A^{-1}(s) = N(s)M^{-1}(s),$$

onde $B(s)$ e $A(s)$ não são necessariamente coprimas à direita e $N(s)$ e $M(s)$ são coprimas à direita. Portanto,

$$B(s) = N(s)R(s) \text{ e } A(s) = M(s)R(s),$$

onde $R(s)$ é o MDC de $B(s)$ e $A(s)$. Então, para calcular o MDC de $B(s)$ e $A(s)$ deve-se usar uma das seguintes relações

$$R(s) = M^{-1}(s)A(s).$$

Logo, calculando uma DFM coprima à direita é possível achar a matriz $R(s)$. Portanto, foi produzido o seguinte algoritmo para calcular a matriz $R(s)$

Algoritmo 4.11 *Sejam $B(s) \in \mathbb{R}^{p\times m}[s]$ e $A(s) \in \mathbb{R}^{m\times m}[s]$. Então para o cálculo do MDC à direita de $B(s)$ e $A(s)$, procede-se da seguinte forma: :*

Passo 1: *Calcule as matrizes $N(s)$ e $M(s)$ matrizes coprimas à direita geradas, supondo que $B(s)$ e $A(s)$ formam uma DFM à direita $H(s) = B(s)A^{-1}(s)$, utilizando os algoritmos 4.10 e 4.9;*

Passo 2: *Calcule $M^{-1}(s)$ através do algoritmo 4.6;*

Passo 3: *Faça a seguinte multiplicação*

$$R(s) = M^{-1}(s)A(s).$$

cancelando o denominador de $M^{-1}(s)$ com os numeradores da $Adj[M(s)]A(s)$. □

A função `mdc`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `mdc.m` (apêndice A.24) . Para usá-la, o usuário deve fornecer as matrizes coeficientes associadas às matrizes polinomiais $A(s)$ e $B(s)$. A função retorna a matriz coeficiente do MDC entre $A(s)$ e $B(s)$. Essa função tem a seguinte sintaxe:

$$[R] = \texttt{mdc}(B, A),$$

onde A e B são, respectivamente, as matrizes coeficientes de $A(s)$ e $B(s)$ e R é a matriz coeficiente da matriz polinomial $R(s)$ que é o MDC à direita de $A(s)$ e $B(s)$.

Exemplo 4.14 *Seja a matriz de transferência apresentada em Kailath (1980, pág. 368)*

$$H(s) = \begin{bmatrix} \frac{s}{(s+1)^2(s+2)^2} & \frac{s}{(s+2)^2} \\ \frac{-s}{(s+2)^2} & \frac{-s}{(s+2)^2} \end{bmatrix}, \tag{4.45}$$

no qual uma DFM à direita é

$$H(s) = \begin{bmatrix} s & s(s+1)^2 \\ -s(s+1)^2 & -s(s+1)^2 \end{bmatrix} \begin{bmatrix} (s+1)^2(s+2)^2 & 0 \\ 0 & (s+2)^2(s+1)^2 \end{bmatrix}^{-1},$$

cujas matrizes coeficientes são dadas por

$$B = \left[\begin{array}{cc|cc|cc|cc} 0 & 1 & 0 & 2 & 1 & 1 & 0 & 0 \\ -1 & -1 & -2 & -2 & -1 & -1 & 0 & 0 \end{array}\right]$$

e

$$A = \left[\begin{array}{cc|cc|cc|cc} 1 & 0 & 6 & 0 & 13 & 0 & 12 & 0 & 4 & 0 \\ 0 & 1 & 0 & 6 & 0 & 13 & 0 & 12 & 0 & 4 \end{array}\right].$$

Um exemplo de utilização da função `mdc` no Matlab é o seguinte:

$>> A = [1 \quad 0 \quad 6 \quad 0 \quad 13 \quad 0 \quad 12 \quad 0 \quad 4 \quad 0; \quad 0 \quad 1 \quad 0 \quad 6 \quad 0 \quad 13 \quad 0 \quad 12 \quad 0 \quad 4];$

$>> B = [0 \quad 1 \quad 0 \quad 2 \quad 1 \quad 1 \quad 0 \quad 0; \quad -1 \quad -1 \quad -2 \quad -2 \quad -1 \quad -1 \quad 0 \quad 0];$

$>> [R] = \texttt{mdc}(B, A);$

Executando o Passo 1 do algoritmo, serão obtidas as seguintes matrizes:

$$M = \left[\begin{array}{cc|cc|cc|cc} 0 & -0.1455 & 0 & -0.5819 & 0 & -0.7274 & 0 & -0.2909 \\ 0 & 0.0238 & 0.1690 & 0.0317 & 0.6761 & -0.0132 & 0.6761 & 0.0370 \end{array}\right].$$

e

$$N = \left[\begin{array}{cc|cc|cc} 0 & 0.0238 & 0.1690 & -0.0635 & 0 & 0 \\ 0 & 0.1217 & -0.1690 & 0.0635 & 0 & 0 \end{array}\right].$$

No Passo 2 obtém-se:

$$NuminvM = \left[\begin{array}{cc|cc|cc|} 0.9681 & 5.9161 & 1.2908 & 23.6643 & -0.5378 & 29.5804 \\ 0 & 0 & -6.8742 & 0 & -27.4968 & 0 \end{array}\right.$$
$$\left.\begin{array}{cc|} 1.5059 & 11.8322 \\ -27.4968 & 0 \end{array}\right].$$

e

$$deninvM = \left[\begin{array}{cccccc} 1 & 8 & 25 & 38 & 28 & 8 \end{array}\right].$$

e finalmente no Passo 3

$$R = \left[\begin{array}{cc|cc|cc} 0 & 0 & 0 & -6.8742 & 0 & -13.7484 \\ 0.9681 & 5.9161 & -0.6454 & 11.8322 & 0.7530 & 5.9161 \end{array}\right].$$

onde, o cancelamento é feito dividindo-se cada elemento de $NuminvM$ por $deninvM$.

Observação 4.6 *Note que no exemplo 4.14 a DFM não coprima à direita $H(s) = A^{-1}(s)B(s)$ tem $gr|A(s)| = 8$ enquanto que na DFM coprima à direita $H(s) = N(s)M^{-1}(s)$ possui $gr|M(s)| = 5$, mostrando, conforme Kailath (1980, pág. 369), que uma realização mínima para $H(s)$ deveria ser de ordem 5.* □

4.2.5 Identidade de Bezout

O cálculo da identidade de Bezout Generalizada é importante tanto do ponto de vista da matemática (Gohberg et al. 1982, Gantmacher 1959), como também na teoria de sistemas (Wolovich 1974, Kailath 1980) Este problema consiste em dada uma matriz $H(s) \in \mathbb{R}^{p\times m}(s)$ calcular as matrizes polinomiais $M(s), N(s)$, $\tilde{M}(s), \tilde{N}(s)$, $X(s), Y(s), \tilde{X}(s)$ e $\tilde{Y}(s)$ de dimensões apropriadas tais que $H(s) = N(s)M^{-1}(s) = \tilde{M}^{-1}(s)\tilde{N}(s)$ e

$$\left[\begin{array}{cc} \tilde{X}(s) & -\tilde{Y}(s) \\ -\tilde{N}(s) & \tilde{M}(s) \end{array}\right] \left[\begin{array}{cc} M(s) & Y(s) \\ N(s) & X(s) \end{array}\right] = \left[\begin{array}{cc} I_m & 0 \\ 0 & I_p \end{array}\right] \tag{4.46}$$

Nesta seção será apresentado um algoritmo que foi proposto em Basilio e Moreira (2004), que é baseado no cálculo de uma base polinomial mínima para o espaço nulo à direita de matrizes polinomiais. Por esta razão, o algoritmo apresentado na seção 4.1.4 será de fundamental importância para a determinação das matrizes da identidade de Bezout generalizada. Note na equação 4.46 que o algoritmo deve ter como saída as matrizes $M(s), N(s), \tilde{M}(s), \tilde{N}(s), X(s), Y(s), \tilde{X}(s)$ e $\tilde{Y}(s)$ para uma dada entrada $H(s) = B(s)A^{-1}(s)$. Uma vez que na seção 4.2.3 já foram apresentados os cálculos de $M(s), N(s), \tilde{M}(s)$ e $\tilde{N}(s)$, nesta seção serão considerados apenas os algoritmos que calcularão as demais matrizes da identidade de Bezout generalizada, isto é, $X(s), Y(s), \tilde{X}(s)$ e $\tilde{Y}(s)$.

Descrição do algoritmo Polybezout

Para uma dada DFM qualquer não coprima $H(s) = A^{-1}(s)B(s), H(s) \in \mathbb{R}^{p\times m}(s)$, o algoritmo para cálculo das matrizes polinomiais da identidade de Bezout Generalizada pode ser dividido em 4 etapas, isto é:

1. Calcular uma DMF coprima à direita $H(s) = N(s)M^{-1}(s)$;
2. Calcular uma DFM coprima à esquerda $H(s) = \tilde{M}^{-1}(s)\tilde{N}(s)$;
3. Calcular as matrizes $\tilde{X}(s)$ e $\tilde{Y}(s)$;
4. Calcular as matrizes $X(s)$ e $Y(s)$.

Os dois primeiros ítens descritos acima foram apresentados na seção 4.2.3. Nesta seção serão apresentados algoritmos para o obtenção dos dois últimos ítens.

Da equação (4.46), pode-se retirar a seguinte relação:

$$M^T(s)\tilde{X}(s) - N^T(s)\tilde{Y}^T(s) - I_m = 0 \tag{4.47}$$

ou equivalentemente

$$\begin{bmatrix} M^T(s) & -N^T(s) & -I_m \end{bmatrix} \begin{bmatrix} \tilde{X}^T(s) \\ \tilde{Y}^T(s) \\ I_m \end{bmatrix} = 0. \tag{4.48}$$

Como mostrado em Lai (1989) a equação terá solução se e somente se for possível achar $[\tilde{X}(s) \quad \tilde{Y}(s) \quad C]^T$, com C não singular, que resolve a equação (4.48). Portanto, o problema de achar $\tilde{X}(s)$ e $\tilde{Y}(s)$, solução da equação (4.47), é equivalente ao problema de se calcular uma $(p+2m)\times m$ matriz polinomial $\widehat{F}(s) \in \mathbb{R}^{(p+2m)\times m}$ nos quais os vetores colunas pertencem ao espaço nulo de

$$T_1(s) = [M^T(s) \quad -N^T(s) \quad -I_m] \tag{4.49}$$

com a restrição que todas as sub-matrizes formadas com as ultimas m linhas das matrizes coeficientes de $\widehat{F}(s)$ devem ser identicamente zero, exceto aquelas de potência independente de s. Contudo, as últimas m linhas dessa matriz de coeficientes devem formar uma matriz não singular. Por exemplo, se $\widehat{F}(s)$ tem grau ϕ, então

$$\widehat{F}(s) = \widehat{F}_0 s^{\phi} + \widehat{F}_1 s^{\phi-1} + \ldots + \widehat{F}_{\phi-1} s + \widehat{F}_{\phi},$$

onde

$$\widehat{F}_i = \left\{ \begin{bmatrix} \widehat{F}_i^{(superior)} \\ 0_{m\times m} \end{bmatrix}, i = 0, \ldots, \phi-1, \text{ e } \begin{bmatrix} \widehat{F}_i^{(superior)} \\ C \end{bmatrix}, i = \phi, \right. \tag{4.50}$$

onde C de ser uma matriz $m \times m$ não-singular. Não é difícil verificar que tal matriz sempre existirá uma vez que as matrizes $M(s)$ e $N(s)$ são, por construção, coprimas à direita.

É importante notar que o cálculo de $\tilde{X}(s)$ e $\tilde{Y}(s)$ não requer que uma base polinomial mínima para o espaço nulo de $T_1(s)$ seja encontrada. Além disso, a matriz de convolução formada com $T_1(s)$ deverá ter uma forma especial, uma vez que a matriz $\widehat{F}_i$ deve ter uma forma mostrada na equação (4.50). Por esta razão esta matriz de convolução será denotada por $\widehat{C}_{\phi_i}(T_1)$ e será referida como a matriz de convolução modificada de $T_1(s)$. Para obter $\widehat{C}_{\phi_i}(T_1)$ defina

$$\widehat{T}_1(s) = [M^T(s) \quad -N^T(s)]. \tag{4.51}$$

Logo, a matriz de convolução modificada de $T_1(s)$ será a seguinte formação:

$$\widehat{C}_{\phi_i}(T_1) = \begin{bmatrix} C_{\phi_i}(\widehat{T}_1) & O_{(\phi_i+\alpha)m\times m} \\ & -I_m \end{bmatrix}, \tag{4.52}$$

onde α é o grau de $\widehat{T}_1(s)$ e $C_{\phi_i}(\widehat{T}_1)$ é a matriz de convolução de $\widehat{T}_1(s)$ com sua formatação mostrada na seção 2.2. Portanto, os m vetores polinomiais que satisfazem a equação (4.48) podem também ser encontrada através do cálculo do espaço nulo de $C_{\phi_i}(\widehat{T}_1)$, definida em (4.52). Então para a obtenção de $F(s)$, deve-se seguir os seguintes passos.

Algoritmo 4.12 *:*

Passo 1: *Faça* $i = 1$ *e* $gr[\underline{\hat{f}}_i(s)] = \phi_i = 0$.

Passo 2: *Forme a matriz* $\widehat{C}_{\phi_i}(T_1)$ *de acordo com a equação (4.52) e calcule o sua DVS, isto é,* $\widehat{C}_{\phi_i}(T_1) = U_{\phi_i}\Sigma_{\phi_i}V_{\phi_i}^T$.

Passo 3: *Faça* n_{ϕ_i} *a dimensão do espaço nulo de* $\widehat{C}_{\phi_i}(T_1)$ *mais o número de colunas em excesso.*

Se $n_{\phi_i} = 0$, *faça* $\phi_i = \phi_i + 1$ *e volte para o* ***Passo 2****.*

Se $n_{\phi_i} > 0$, *haverá pelo menos um vetor polinomial de grau* ϕ_i *que satisfaz a equação (4.48). Estes vetores serão formados a partir das* n_{ϕ_i} *últimas colunas de* V_{ϕ_i}.

Quando $i = 1$, $\underline{\hat{f}}^{(1)}$ *será escolhido entre as últimas* n_{ϕ_i} *colunas de* V_{ϕ_i} *em que pelo menos um dos* m *últimos elementos é diferente de zero. Quando* $n_{\phi_i} > 1$, *outros vetores polinomiais que satisfazem a equação (4.48) poderão ser escolhidos desde que a matriz formada com os* m *últimos elementos dos vetores* $\underline{\hat{f}}^{(i)}$, *que já tinham sido escolhidos, e o últimos* m *elementos do vetor considerado tenha posto completo por coluna. Repita este passo até que todos os possíveis vetores formados de* V_{ϕ_i} *tenham sido verificados ou que base tenha sido completada. Faça* $i = i + 1$, *a cada vez que um novo vetor polinomial for somado à base.*

Passo 4: *Se* $i \leq m$ *então faça* $\phi_i = \phi_i + 1$ *e volte para o* ***Passo 2****. Caso contrário suponha que* $\max \phi_i = \phi$ *e denote*

$$\widehat{F}(s) = \widehat{F}_0 s^{\phi} + \widehat{F}_1 s^{\phi-1} + \ldots + \widehat{F}_{\phi-1} s + \widehat{F}_{\phi},$$

e forme, de acordo com a equação (4.50), a matriz C *com as últimas* m *linhas de* $\widehat{F}_{\phi}$ *e calcule* $F(s) = \widehat{F}(s)C^{-1}$.

Uma vez obtidas as matrizes $\tilde{X}(s)$ e $\tilde{Y}$ deve-se agora calcular as matrizes $X(s)$ e $Y(s)$, a partir das matrizes $\tilde{M}(s)$, $\tilde{N}(s)$, $\tilde{X}(s)$ e $\tilde{Y}(s)$ já calculadas. Assim, de acordo

com a equação (4.46), $X(s)$ e $Y(s)$ devem satisfazer:

$$\begin{bmatrix} \tilde{X}(s) & -\tilde{Y}(s) \\ -\tilde{N}(s) & \tilde{M}(s) \end{bmatrix} \begin{bmatrix} Y(s) \\ X(s) \end{bmatrix} = \begin{bmatrix} 0 \\ I_p \end{bmatrix} \tag{4.53}$$

ou, equivalentemente,

$$\begin{bmatrix} \tilde{X}(s) & -\tilde{Y}(s) & 0 \\ -\tilde{N}(s) & \tilde{M}(s) & -I_p \end{bmatrix} \begin{bmatrix} Y(s) \\ X(s) \\ I_p \end{bmatrix} = 0. \tag{4.54}$$

Portanto, definindo

$$T_2(s) = \begin{bmatrix} \tilde{X}(s) & -\tilde{Y}(s) & 0 \\ -\tilde{N}(s) & \tilde{M}(s) & -I_p \end{bmatrix}, \tag{4.55}$$

então o problema de calcular $X(s)$ e $Y(s)$ é equivalente ao de achar p vetores polinomiais pertencentes ao espaço nulo de $T_2(s)$ nos quais a matriz formada com as últimas p linhas seja não-singular. Este problema é similar ao do cálculo das matrizes $\tilde{X}(s)$ e $\tilde{Y}(s)$ e, portanto, os mesmos procedimentos para achar os vetores polinomiais do espaço nulo à direita de $T_1(s)$, pode ser agora adotado. Para obter as matrizes $X(s)$ e $Y(s)$, que resolvem a equação (4.54), deve-se utilizar:

$$\widehat{T}_2(s) = \begin{bmatrix} \tilde{X}(s) & -\tilde{Y}(s) \\ -\tilde{N}(s) & \tilde{M}(s) \end{bmatrix}, \tag{4.56}$$

então a matriz de convolução modificada de $T_2(s)(\widehat{C}_{\phi_i}(T_2))$ será dada por:

$$\widehat{C}_{\phi_i}(T_2) = \begin{bmatrix} & O_{(\phi_i+\alpha)(p+m)\times p} \\ C_{\phi_i}(\widehat{T}_2) & O_{m\times p} \\ & -I_p \end{bmatrix}, \tag{4.57}$$

onde α é o grau de $\widehat{T}_2(s)$ e $C_{\phi_i}(\widehat{T}_2)$ é a matriz de convolução de $\widehat{T}_2(s)$ com sua formatação mostrada na seção 2.2. Portanto, é possível encontrar as matrizes polinomiais $X(s)$ e $Y(s)$ que satisfazem a equação (4.54) utilizando o algoritmo 4.12 para achar os p vetores polinomiais pertencentes ao espaço nulo de $T_2(s)$

A função `polybezout`

A função Matlab desenvolvida para implementar o algoritmo descrito nesta seção é a `polybezout.m` (apêndice A.27). Para usá-la, o usuário deve fornecer as matrizes coeficientes associadas às matrizes polinomiais $B(s)$ e $A(s)$ geradas pela equação

$H(s) = A^{-1}(s)B(s)$. A função retorna as matrizes coeficientes $M(s), N(s)$, $\tilde{N}(s)$, $\tilde{M}(s)$,$X(s), Y(s)$, $\tilde{X}(s)$ e $\tilde{Y}(s)$, que satisfazem a equação

$$\begin{bmatrix} \tilde{X}(s) & -\tilde{Y}(s) \\ -\tilde{N}(s) & \tilde{M}(s) \end{bmatrix} \begin{bmatrix} M(s) & Y(s) \\ N(s) & X(s) \end{bmatrix} = \begin{bmatrix} I_m & 0 \\ 0 & I_p \end{bmatrix}.$$

Essa função tem a seguinte sintaxe:

$$[M, N, Mtil, Ntil, X, Y, Xtil, Ytil] = \texttt{polybezout}(B, A, Bncol),$$

onde $M, N, Mtil$, $Ntil, X$, $Y, Xtil$ e $Ytil$ são as matrizes coeficientes de $M(s), N(s)$, $\tilde{N}(s), \tilde{M}(s)$, $X(s), Y(s)$, $\tilde{X}(s)$ e $\tilde{Y}(s)$, respectivamente, B e A são as matrizes coeficientes de $A(s)$ e $B(s)$, e $Bncol$ é o número de colunas da matriz $B(s)$.

Exemplo 4.15 *Seja a matriz de transferência apresentada em Kailath (1980, pág. 368):*

$$H(s) = \begin{bmatrix} \frac{s}{(s+1)^2(s+2)^2} & \frac{s}{(s+2)^2} \\ \frac{-s}{(s+2)^2} & \frac{-s}{(s+2)^2} \end{bmatrix} = A^{-1}(s)B(s) \tag{4.58}$$

Uma DFM à esquerda é

$$H(s) = \begin{bmatrix} (s+1)^2(s+2)^2 & 0 \\ 0 & (s+2)^2(s+1)^2 \end{bmatrix}^{-1} \begin{bmatrix} s & s(s+1)^2 \\ -s(s+1)^2 & -s(s+1)^2 \end{bmatrix},$$

cujas matrizes coeficientes são dadas por

$$B = \left[\begin{array}{cc|cc|cc|cc} 0 & 1 & 0 & 2 & 1 & 1 & 0 & 0 \\ -1 & -1 & -2 & -2 & -1 & -1 & 0 & 0 \end{array}\right]$$

e

$$A = \left[\begin{array}{cc|cc|cc|cc|cc} 1 & 0 & 6 & 0 & 13 & 0 & 12 & 0 & 4 & 0 \\ 0 & 1 & 0 & 6 & 0 & 13 & 0 & 12 & 0 & 4 \end{array}\right].$$

Um exemplo de utilização da função **`polybezout`** *no Matlab é o seguinte:*

$>> A = [1 \quad 0 \quad 6 \quad 0 \quad 13 \quad 0 \quad 12 \quad 0 \quad 4 \quad 0;\ 0 \quad 1 \quad 0 \quad 6 \quad 0 \quad 13 \quad 0 \quad 12 \quad 0 \quad 4];$

$>> B = [0 \quad 1 \quad 0 \quad 2 \quad 1 \quad 1 \quad 0 \quad 0;\ -1 \quad -1 \quad -2 \quad -2 \quad -1 \quad -1 \quad 0 \quad 0];$

$>> [M, N, Mtil, Ntil, X, Y, Ytil, Xtil] =$ **`polybezout`**$(B, A, 2);$

O cálculo das matrizes coprimas à direita $M(s)$ e $N(s)$, e à esquerda $\tilde{M}(s)$ e $\tilde{N}(s)$ foram mostradas na seção 4.2.3. Portanto, os seus resultados serão dados diretamente:

$$M = \left[\begin{array}{cc|cc|cc|cc} 0 & 0.0528 & 0.0000 & 0.2110 & 0.0000 & 0.2638 & 0.0000 & 0.1055 \\ 0 & 0.0896 & 0.1690 & 0.1901 & 0.6761 & -0.3680 & 0.6761 & -0.7793 \end{array}\right],$$

$$N = \left[\begin{array}{cc|cc|cc} 0.0000 & 0.0896 & 0.1690 & -0.1684 & -0.0000 & 0.0000 \\ 0.0000 & -0.1424 & -0.1690 & 0.1684 & -0.0000 & -0.0000 \end{array}\right],$$

$$\tilde{M} = \left[\begin{array}{cc|cc|cc|cc} 0 & 0 & 0 & 0.1690 & 0 & 0.6761 & 0 & 0.6761 \\ 0.1455 & 0.0238 & 0.5819 & 0.0317 & 0.7274 & -0.0132 & 0.2909 & 0.0370 \end{array}\right],$$

e

$$\tilde{N} = \left[\begin{array}{cc|cc|cc} 0 & 0 & -0.1690 & -0.1690 & 0 & 0 \\ -0.0238 & 0.1217 & 0.0635 & 0.0635 & 0 & 0 \end{array}\right].$$

Considere, agora, o cálculo das matrizes $\tilde{X}(s)$ *e* $\tilde{Y}(s)$*. Para tanto, deve-se encontrar encontrar dois vetores polinomiais do espaço nulo à direita da matriz* $T_1(s)$*, de tal sorte que a matriz formada com as últimas duas linhas de* $\widehat{F}_\phi$*(definida na equação (4.50)) seja não-singular. Esta procura é feita utilizando-se o algoritmo 4.12, como segue:*

Na execução do Passo 1 do algoritmo, $i = 1$ *e* $\phi_1 = 0$ *e a matriz de convolução*

$$\widehat{C}_0(T_3) = \left[\begin{array}{cccccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0528 & 0.0896 & 0 & 0 & 0 & 0 \\ 0.0000 & 0.1690 & 0 & 0 & 0 & 0 \\ 0.2110 & 0.1901 & -0.0896 & 0.1424 & 0 & 0 \\ 0.0000 & 0.6761 & -0.1690 & 0.1690 & 0 & 0 \\ 0.2638 & -0.3680 & 0.1684 & -0.1684 & 0 & 0 \\ 0.0000 & 0.6761 & 0 & 0 & -1 & 0 \\ 0.1055 & -0.7793 & 0 & 0 & 0 & -1 \end{array}\right].$$

Calculando o DVS desta matriz, obtém-se os seguintes valores singulares: 1.5787*,* $1.0027, 0.6113$*,* $0.3347, 0.1286$ *e* 0.0101*, mostrando que é necessário aumentar a ordem da matriz de convolução, isto é,* $\phi_1 = 1$*. Repetindo o procedimento acima obtém-se uma matriz (*10×10*) onde os seus valores singulares são:* $1.7186, 1.2422$*,* 1.0025*,* $0.5611, 0.3830, 0.1631$*,* $0.0950, 0.0317, 0.0085, 0.0000$*. Note que existe um valor singular nulo e, conseqüentemente, a última coluna de* V_1 *poderá formar a primeira coluna de* $\widehat{F}(s)$*, desde que pelo menos um dos últimos dois elementos seja não-nulo. De fato,*

$$V_1^{(10)} = \left[\begin{array}{cccccccc} 0 & 0 & -0.2100 & -0.1127 & 0.1127 & -0.0973 & -0.8400 & -0.4510 \end{array} \ldots\right.$$

$$\left.\begin{array}{cc} \ldots & -0.0658 \quad 0.0877 \end{array}\right]$$

e, portanto,

$$\hat{f}_1(s) = \begin{bmatrix} 0.1127 \\ -0.0973 \\ -0.2100s - 0.8400 \\ -0.1127s - 0.4510 \\ -0.0658 \\ 0.0877 \end{bmatrix}.$$

Dando continuidade à procura do segundo vetor, tem-se que $i = 2$ *e* $\phi_2 = 2$*, gerando uma matriz de convolução* $\widehat{C}_2(T_1)$ *de dimensão* (12×14)*, com os seguintes valores singulares:* $1.7680, 1.5109, 1.0299,\ 1.0025, 0.4838, 0.4478,\ 0.2719, 0.1031, 0.0735,\ 0.0289,\ 0.0090, 0$*. Como* $\widehat{C}_2(T_1)$ *tem dimensão* (12×14)*, haverá três candidatos a vetores para* $\hat{f}_2(s)$*. Tomando a última coluna de* V_2*, obtém-se:*

$$V_2^{(14)} = \begin{bmatrix} 0 & 0 & 0.2280 & -0.0131 & 0.0131 & 0.2412 & 0.6223 & -0.2504 & 0.1978 & \dots \end{bmatrix}$$

$$\dots \quad -0.0920 \quad -0.0190 \quad -0.6158 \quad -0.0622 \quad 0.0925 \end{bmatrix}.$$

e

$$\hat{f}_2(s) = \begin{bmatrix} 0.0131s + 0.1978 \\ 0.2412s - 0.0920 \\ 0.2280s^2 + 0.6223s - 0.0190 \\ -0.0131s^2 - 0.2504s - 0.6158 \\ -0.0622 \\ 0.0925 \end{bmatrix}.$$

Note que as duas linhas dos vetores $\hat{f}_1(s)$ *e* $\hat{f}_2(s)$ *formam a matriz*

$$C = \begin{bmatrix} -0.0658 & -0.0622 \\ 0.0877 & 0.0925 \end{bmatrix}$$

que é não-singular, portanto, a base está completa. Assim, de acordo com o Passo 4 do algoritmo,

$$F(s) = \widehat{F}(s)C^{-1} =$$

$$\begin{bmatrix} 0.1127 & 0.0131s + 0.1978 \\ -0.0973 & 0.2412s - 0.0920 \\ -0.2100s - 0.8400 & 0.2280s^2 + 0.6223s - 0.0190 \\ -0.1127s - 0.4510 & -0.0131s^2 - 0.2504s - 0.6158 \end{bmatrix} \begin{bmatrix} -0.0658 & -0.0622 \\ 0.0877 & 0.0925 \end{bmatrix}^{-1}.$$

Finalmente, particionando $F(s)$ *apropriadamente, obtém-se:*

$$\tilde{X} = \begin{bmatrix} 1.8208s + 10.9225 & 33.4132s + 1.4790 \\ 1.3654s + 9.4765 & 25.0566s \end{bmatrix}$$

e

$$\tilde{Y} = \begin{bmatrix} 31.5924s^2 + 116.9260s + 120.1879 & -1.8208s^2 - 18.2057s - 19.3809 \\ 23.6912s^2 + 85.2882s + 80.5500 & -1.3654s^2 - 14.9382s - 19.6764 \end{bmatrix}$$

Para o cálculo de $X(s)$ e $Y(s)$, o mesmo procedimento acima deve ser seguido. A única diferença é que agora as matrizes $\widehat{C}_{\phi_i}(T_2), i = 1, 2$ devem ser formadas de acordo com a equação (4.57). Executando-se o algoritmo 4.12 para T_2 obtém-se:

$$X = \begin{bmatrix} -2.7801s^2 - 0.4873s - 0.1882 & 14.5988s + 3.4371 \\ 4.4164s^2 - 0.2657s + 1.4790 & -23.1915s \end{bmatrix}$$

e

$$Y = \begin{bmatrix} -1.6363s^3 - 5.7923s^2 - 6.4605s - 4.6710 & 8.5927s^2 + 30.9339s + 29.2153 \\ -2.7801s^3 - 11.6075s^2 - 11.6213s - 0.1825 & 14.5988s^2 + 61.8323s + 63.5509 \end{bmatrix}.$$

Capítulo 5

Conclusão e trabalhos futuros

Neste trabalho operações com matrizes polinomiais foram consideradas e algoritmos para a realização dessas operações foram apresentados. Esses algoritmos são especialmente adequados para utilização em sistemas de controle multivariáveis, embora suas aplicações possam ser estendidas para outras áreas da matemática. Dentro do escopo dos algoritmos robustos, foi apresentado métodos computacionais elaborados utilizando, principalmente, matrizes de Sylvester e DVS, que atendem aos requisitos deste trabalho.

Dentro dos algoritmos propostos destacam-se os principais por possuir resultados significativos nas áreas da matemática e de sistemas lineares. Entretanto, não pode ser desconsiderado que a grande contribuição está nos algoritmos fundamentais destacando-se os algoritmos da redução por coluna(reducol, inversa e determinante) e da base polinomial mínima(rcmfd, lcmfd, MDC e polybezout) que são o núcleo de funcionamento dos algoritmos principais. Em complexidade destacam-se os algoritmos MDC e Polybezout que implementados fizeram uso da integração entre os algoritmos produzidos.

No que se refere a robustez dos algoritmos isto se deve, principalmente, à utilização da DVS que, conforme mostrado na seção 2.10. Note que uma pequena perturbação numérica em uma matriz qualquer, não implicará em uma mudança no posto desde que o menor valor singular esteja dentro de uma tolerância. É importante também mencionar a contribuição que a sua fatoração traz na procura do vetor pertencente ao espaço nulo de uma matriz.

Os algoritmos foram implementados em Matlab e testados através de alguns exemplos que são apresentados neste trabalho, contudo durante a sua realização foram utilizados outros que não foram mostrados cujos resultados foram eficientes no tempo de execução como também nas respostas encontradas.

No que se refere a trabalhos futuros que podem ser desenvolvidos a partir desta tese pode-se destacar:

1. A utilização da fatoração LQ no lugar de DVS como foi sugerido em Zúñiga e Henrion (2005), com o objetivo de diminuir o tempo de execução dos algoritmos;

2. Implementação de uma algoritmo que ache a forma de Smith-McMillan de um sistema multivariável.

E, finalmente, não pode deixar de ser mencionado o principal objetivo deste trabalho que é a gratuidade da "toolbox", onde será disponibilizado para área acadêmica os programas produzidos no Matlab, podendo no futuro ser um grande toolbox com a contribuição de trabalhos deste tipo.

Bibliografia

Anderson, B. D. O. e Jury, E. I. (n.d.). Generalized Bezoutian and Sylvester matrices in multivariable linear control, *IEEE - Transactions on Automatic Control* **01**: 551–556.

Barnett, S. (1983). *Polynomial and Linear Control Synthesis*, Marcel Dekker, London.

Basilio, J. (2002). Inversion of polynomial matrices via state space, *Linear Algebra and Its Applications* **357**: 259–271.

Basilio, J. e Kouvaritakis, B. (1997). An algorithm for coprime matrix fraction description using Sylvester matrices, *Linear Algebra and Its Applications* **266**: 107–125.

Basilio, J. e Moreira, M. V. (2004). A robust solution of the generalized polynomial Bezout identity, *Linear Algebra and Its Applications* **357**: 287–300.

Buslowicz, M. (1980). Inversion of polynomial matrices, *International Journal of Control* **33**: 977–905.

Chen, C. T. (1999). *Linear System Theory and Design*, Oxford University Press, New York Oxford.

Gantmacher, F. R. (1959). *The Theory of matrices, Vol I e II*, Chelsea Publishing Company, New York.

Garcia, J. S. e Basilio, J. C. (2000). Redução direta de modelos de sistemas multivariáveis por truncamento balanceado, *Anais do* 63^o *Congresso Brasileiro de Automática*, Florianópolis, Brasil, pp. 873–878.

Gohberg, I. C., Lancaster, P. e Rodman, L. (1982). *Matrix Polynomials*, Academic Press, New York.

Inouye, I. (1979). An algorithm for inverting polynomial matrices, *International Journal of Control* **30**: 989–990.

Kailath, T. (1980). *Linear System*, Prentice Hall, Englewood Cliffs.

Kaylath, T., Bitmead, R. R., Kung, S. Y. e Anderson, B. D. O. (1978). Greatest common divisors via generalized Sylvester and Bezout matrices, *IEEE - Transactions on Automatic Control* **23**: 1043–1045.

Kucera, V. (1979). *Discrete Linear Control: the polynomial Equation Approach*, John Wiley an Sons, Chichester.

Lai, Y. S. (1989). An algorithm for solving the matrix polynomial equation $b(s)d(s)+a(s)n(s) = h(s)$, *IEEE Transactions on Circuits and Systems* **36**: 1083–1089.

Moreira, M. V. (2001). *Controladores robustos h_∞ não-frágeis*, Tese de Mestrado, UFRJ/COPPE - Programa de Engenharia Elétrica.

Moreira, M. V. e Basilio, J. C. (2004). Controladores comutativos estabilizantes para plantas instáveis, *Anais do 15º Congresso Brasileiro de Automática*.

Neven, W. H. L. e Praagman, C. (1993). Column reduction of polynomial matrices, *Linear Algebra and Its Applications* **188**: 569–589.

Polyx (2004). *www.Polyx.com*, Polyx,Ltd, Tchecoslovaquia.

Strang, G. (1988). *Linear Algebra and Its Applications*, Harcourt Brace Jovanovich, San Diego.

Wolovich, W. A. (1974). *Linear Multivariable System*, Springer-Verlag, New York.

Zhang, S. Y. (1987). Inversion of polynomial matrices, *International Journal of Control* **46**: 33–37.

Zúñiga, J. C. e Henrion, D. (2004). On the application of different numerical methods to obtain null-spaces of polynomial matrices.Part 1 and 2., *IEEE Conference on Decision and Control*, Paradise Island, Bahamas.

Zúñiga, J. C. e Henrion, D. (2005). Block Toeplits algorithms for polynomial matrix null-space computation. submetido para publicação.

Apêndice A

Funções MATLAB

A.1 Matsplit

```
function avec = matsplit(a,m)
[p ma]=size(a);
if nargin==1;
    m=p;
end
for j=1:m
  for i=1:p
    avec((j-1)*p+i,:)=a(i,j:m:ma);
  end
end
```

A.2 Vecmat

```
function a = vecmat(avec,p,m);
if nargin==2;
    m=p;
end
[mavec,lavec]=size(avec);la=m*lavec;a=zeros(p,la);
for i=1:p;
    for j=1:m;
        a(i,j:m:la)=avec((j-1)*p+i,:);
```

```
    end;
end
```

A.3 Cutzeros

```
function a=cutzeros(a,q);
[p,ma]=size(a);
if nargin==1;
    q=p;
end
flag=1;
while flag
    aa=a(:,1:q);flagi=1;i=1;
    while flagi
        if norm(aa(:,i))>1e-9;
          flagi=0;flag=0;
        else
            if i<q;i=i+1;else flagi=0;end
        end
    end
    if flag
        a=a(:,q+1:ma);ma=size(a,2);
        if ma==q;flag=0;end
    end
end
```

A.4 Matform

```
function c=matform(m,n,mcol,ncol);
[lm,cm]=size(m);[ln,cn]=size(n);
if nargin==2;
```

```
    mcol=lm;ncol=ln;
end
mmax=cm/mcol;nmax=cn/ncol;[imin,ind]=min([mmax nmax]);c=[];
for i=1:imin;
    c=[m(:,cm-i*mcol+1:cm-(i-1)*mcol)n(:,cn-i*ncol+1:cn-(i-1)*ncol) c];
end
if ind==1;
    for i=imin+1:nmax;
        c=[zeros(lm,mcol) n(:,cn-i*ncol+1:cn-(i-1)*ncol) c];
    end
else
    for i=imin+1:mmax;
        c=[m(:,cm-i*mcol+1:cm-(i-1)*mcol) zeros(lm,ncol) c];
    end
end
```

A.5 Convform

```
function [C] = convform(N,wl,p);
[m,n]=size(N);
if
    nargin==2;p=m;
end;
ord=n/p;S=[];
for i=1:ord;
    S=[S;N(:,(i-1)*p+1:i*p)];
end;
[ms ns]=size(S);C=[];
for i=1:wl;
  CC=zeros(ms+m*(wl-1),p);CC((i-1)*m+1:(i-1)*m+ms,:)=S;C=[C CC];
```

```
end
```

A.6 Diaglamb

```
function lam=diaglamb(Lb);
[n,m]=size(Lb);
for i=1:n;
   lam(i,i:n:n*m)=Lb(i,:);
end
```

A.7 Diagpol

```
function Diag = Diagpol(g);
[m,n]=size(g); gvec=matsplit(g,m);
Diag=[];
for i=1:m+1:m*m,
    Diag=[ Diag;gvec(i,:)];
end
```

A.8 Polymatval

```
function as = polymatval(a,s,n);
m=size(a,1);
if nargin==2;
    n=m;
end
la=size(a,2)/n;as=[]; potsk=[];
for jj=la-1:-1:0;
    potsk=[potsk;s^jj*eye(n)];
end
as=a*potsk;
```

A.9 Coldeg

```
function GrauCol = coldeg(g,col); m=size(g,1);
if
    nargin==1;col=m;
end
GrauCol=[];gvec=matsplit(g,col); ind=1;
for j=1:col;
    gveccol=gvec(ind:ind+m-1,:);
    gvecj=cutzeros(gveccol,1);
    GrauCol=[GrauCol size(gvecj,2)-1];
    ind=ind+m;
end
```

A.10 Matsum

```
function [c]=matsum(a,b);
[p,ma]=size(a);mb=size(b,2);
c=[zeros(p,mb-ma) a]+[zeros(p,ma-mb) b];
```

A.11 Convmat

```
function C = convmat(A,B,p);
[ma na]=size(A); [mb nb]=size(B);
orda=na/mb;
if nargin==2
    p=mb;
end
ordb=nb/p; ordc=orda+ordb-1; AA=[]; C=[];
CVEC=zeros(ordc*ma,p);
for i=1:orda;
```

```
    AA=[AA;A(:,(i-1)*mb+1:i*mb)];
end
maa=ma*orda;
for i=1:ordb;
  CC=zeros(ordc*ma,mb);
  CC((i-1)*ma+1:(i-1)*ma+maa,:)=AA;
  CVEC=CVEC+CC*B(:,(i-1)*p+1:i*p);
end
for i=1:ordc;
    C=[C CVEC((i-1)*ma+1:i*ma,:)];
end
```

A.12 Transpol

```
function nt = transpol(n,q);
[ln,cn]=size(n);nt=[];
if nargin==1;
    q=ln;
end
for i=1:q:cn;
    nt=[nt n(:,i:i+q-1).'];
end
```

A.13 Conjpol

```
function Conjpol = coldeg(g,col);
[m,n]=size(g);
if
    nargin==1;col=m;
end;
Graumax=n/col-1;Conjpol=[];
```

```
for i=0:Graumax,
    Conjpol=
    [ ((-1)^i)*g(:,(Graumax-i)*col+1:(Graumax-i)*col+col) Conjpol ];
end
```

A.14 Tracopol

```
function Traco = Tracopol(g);
diagvec=diagpol(g);
[m,n]=size(diagvec); Traco=zeros(1,n);
for i=1:(m),
    Traco=Traco+diagvec(i,:);
end
```

A.15 NumSingNull

```
function num_sing0=numsingnull(sing,tol)
if nargin==1;
    tol=1e-9;num_sing=size(sing,2);
end num_sing=size(sing,2); flag1=1;jj=num_sing;num_sing0=0;
if sing(num_sing)==0;
   sing(num_sing)=1e-16;
end
if sing(1)/sing(num_sing)>1e7;
    singnorm=sing/sing(1);
else
    singnorm=sing;
end
while flag1
    if singnorm(jj) < tol;
        if jj>0;jj=jj-1;else flag1=0;end
```

```
        num_sing0=num_sing0+1;
    else
        flag1=0;
    end
end
```

A.16 Reducol

```
function [a,U_total]=reducol(a,m); p=size(a,1);
if nargin==1;
    m=p;
end
graucol = coldeg(a,m); avec=matsplit(a);[n]=size(avec,2);
Dhc=[];
for i=1:m;
    Dhc=[Dhc avec((i-1)*p+1:i*p,n-graucol(i))];
end;
E_new=eye(m);U_total=eye(m);
while abs(det(Dhc))<1^-9,
    [U,E]=redumat(Dhc,graucol,m);
    U_new=convmat(E,U);
    U_total=convmat(U_total,U_new);
    a=cutzeros(convmat(a,U_new));
    graucol = coldeg(a,m);
    avec=matsplit(a);[n]=size(avec,2);
    Dhc=[];
    for i=1:m;
        Dhc=[Dhc avec((i-1)*p+1:i*p,n-graucol(i))];
    end;
end;
```

A.16.1 Redumat

```
function [U,E] = redumat(Dhc,coldeg,glin);
coldeg_old=coldeg;
[coldeg,ind]=sort(coldeg);ind=fliplr(ind);coldeg=fliplr(coldeg);
U=eye(glin); E=eye(glin);
if norm(coldeg_old-coldeg,1)~=0;
    E=E(:,ind);Dhc=Dhc(:,ind);
end
[J,S,V]=svd(Dhc);
indzero=find(abs(diag(S))<1e-13);Y=V(:,indzero);ybarra=[];kbarra=[];
for jj=1:size(Y,2);
    ii=1;flag=1;
    while flag
        if abs(Y(ii,jj))>(1e-13);
            ind=find(kbarra==ii);
            if length(ind)==0;
                kbarra=[kbarra ii];ybarra=[ybarra Y(:,jj)];
            end
            flag=0;
        else
            ii=ii+1;
        end
    end
end
[kbarra,ind]=sort(kbarra);ybarra=ybarra(:,ind);
Uvec=matsplit(U);lUvec=size(Uvec,1);
Uvec=[zeros(lUvec,coldeg(kbarra(1))-coldeg(end))
Uvec];ncolUvec=size(Uvec,2);
for ii=1:length(kbarra)
```

```
    for jj=kbarra(ii):glin
        Uvec(glin*(kbarra(ii)-1)+jj,
        ncolUvec-(coldeg(kbarra(ii))-coldeg(jj)))=ybarra(jj,ii);
    end
end
U=vecmat(Uvec,glin);
```

A.17 Ss2tflev

```
function [n,d]=ss2tflev2(a,b,c,d)
n=size(a,1);p=size(c,1);q=size(b,2);dd=d;d=[1];
si=eye(n);ai=-trace(a);adja=si;d=[1 ai];
for i=2:n;
        si=si*a+ai*eye(n);ai=-trace(si*a)/i;adja=[adja si];d=[d ai]
end
n=matsum(convmat(c,convmat(adja,b,q),q),
vecmat(convmat(matsplit(dd,q),d,1),p,q)); n=cutzeros(n);
```

A.18 Polybases

```
function [F]=polybases(t,p,ncol); nlin=size(t,1);
if nargin==2;
    ncol=nlin;
end
nbase=0;j=0;flg=0;fhc=[];fvec=[]; esp_nulo=ncol-p;
if esp_nulo==0
    display('Espaco nulo: vetor origem'),return
end

while nbase<esp_nulo;
    cjt=convform(t,j+1,ncol);
```

```
    [xj,sj,yj]=svd(cjt);[lcjt,ccjt]=size(cjt);
    nj=NumSingNull(diag(sj),10^-4);
    if ccjt>lcjt;
        ncj=nj+ccjt-lcjt;
    else
        ncj=nj;
    end
    for i=1:ncj;
      if nbase<esp_nulo;
        if nbase==0;
            f=yj(:,ccjt-i+1);fhc=f(1:ncol);nbase=1;
        else
            fi=yj(:,ccjt-i+1);fhci=[fhc fi(1:ncol)];sfhci=svd(fhci);
            if NumSingNull(sfhci,10^-9)==0
                [mf,nf]=size(f);
                f=[[zeros(ccjt-mf,nf);f] fi];fhc=fhci;
                nbase=nbase+1;
            end
        end
      end
    end
    j=j+1;
end
F=[];
for i=1:ncol:size(f,1);
    F=[F f(i:i+ncol-1,:)];
end
```

A.19 Polymatposto

```
function posto=polymatposto(d,tol,ncol)
nlin=size(d,1);
if nargin==1;
    tol=1e-9;ncol=nlin;
else
    if nargin==2;
        ncol=nlin;
    else
        if nlin>ncol;
            d=transpol(d,ncol);aux=ncol;
            ncol=nlin;nlin=aux;
        end
    end
end
graus=coldeg(d,ncol);[ord_grau,ind]=sort(graus);
k=norm(ord_grau(1:nlin),1); i=0;rank=[];flag=1;
while flag
    dsk=polymatval(d,(-1)^i*i/2,ncol);sing=svd(dsk)';
    flag1=1;jj=nlin;num_sing0=0;
    num_sing0=NumSingNull(sing,tol,nlin);
    if i==0;
        posto=nlin-num_sing0;
    else
        posto=max([posto nlin-num_sing0]);
    end
    if i==k+1 | posto==nlin;
        flag=0;
    else
```

```
        i=i+1;
    end
end
```

A.20 Inverse

```
function [n,d]=inverse(g);
[glin,ld]=size(g); ldm=ld/glin;
if fix(ldm) ~=ldm;error('A matriz deve ser quadrada');end
gvec=matsplit(g); [m]=size(gvec,2); graucol=coldeg(g);
Dhc=[];Dlc=[]; for i=1:glin
    veci=[glin*(i-1)+1:glin*i];
    Dhc=[Dhc gvec(veci,m-graucol(i))];
end;
U_total=eye(size(g,1));
while numsingnull(svd(Dhc),10^-5)~=0,
    [U,E]=redumat(Dhc,graucol,glin);
    U_new=convmat(E,U);
    U_total=convmat(U_total,U_new);
    g=cutzeros(convmat(g,U_new));
    graucol=coldeg(g);
    gvec=matsplit(g);[m]=size(gvec,2);
    Dhc=[];
    for i=1:glin;
        veci=[glin*(i-1)+1:glin*i];Dhc=[Dhc gvec(veci,m-graucol(i))];
    end
end
ind=find(graucol==0);flag=0; if length(ind)~=0
lind=length(ind);deltavec=[zeros(glin,1) ones(glin,1)];
deltavec(ind,:)=[ones(lind,1) zeros(lind,1)];
```

```
delta=diaglamb(deltavec); g=cutzeros(convmat(g,delta));
flag=1;graucol=coldeg(g);gvec=matsplit(g);
end;
for i=1:glin;
     veci=[glin*(i-1)+1:glin*i];
    Dlc=[Dlc gvec(veci,m-graucol(i)+1:end)];
end
ac=[];
for i=1:length(graucol);
    ac=blkdiag(ac,[zeros(1,graucol(i));
    eye(graucol(i)-1,graucol(i))]);
end
Dhcinv=inv(Dhc);prod=Dhcinv*Dlc; acpos=[1];
for i=1:length(graucol)-1;
    acpos=[acpos norm(graucol(1:i),1)+1];
end
for i=1:glin;
    ac(acpos(i),:)=-prod(i,:);
end;
bc=[];
for i=1:glin;
    bc=[bc;[Dhcinv(i,:);zeros(graucol(i)-1,glin)]];
end;
 ord_col=graucol;
% Cria\c{c}ao de Cc
Cc=zeros(glin,sum(ord_col));tam=1;ind=0;
while tam<=glin
    ind=ind+ord_col(tam);
    Cc(tam,ind)=1;
    tam=tam+1;
```

```
end
%Cria\c{c}ao de Dc
Dc=zeros(glin);
%Utiliza\c{c}ao do algoritmo de Leverrier
% [n,d]=ss2tflev(ac,bc,Cc,Dc);
n2=[]; n=[];
for i=1:glin
    [n2,d]=ss2tf(ac,bc,Cc,Dc,i);
    n=[n;n2];
end;
n=vecmat(n,glin); if flag
    n=convmat(delta,n);
end;
n=convmat(U_total,n);
n=cutzeros(n);
```

A.21 Polydet

```
function [d] = polydet(g);
 [glin,ld]=size(g);ldm=ld/glin;
 if fix(ldm) ~=ldm;
    error('A matriz deve ser quadrada');
 end
Graucol=coldeg(g); gvec=matsplit(g);[m]=size(gvec,2);
Dhc=[];Dlc=[];
for i=1:glin
    veci=[glin*(i-1)+1:glin*i];
    Dhc=[Dhc gvec(veci,m-Graucol(i))];
end; E_new=eye(size(g,1)); U_total=eye(size(g,1)); detU=1;
while abs(det(Dhc))<1e-9
```

```
    [U,E]=redumat(Dhc,Graucol,glin);
    Udiag=diagpol(U);
    for i=1:glin,detU=detU*Udiag(i,end);end
    U_new=convmat(E,U);
    U_total=convmat(U_total,U_new);
    g=cutzeros(convmat(g,U_new));
    Graucol=graucolunas(g);
    gvec=matsplit(g);[m]=size(gvec,2);
    Dhc=[];
    for i=1:glin;
        veci=[glin*(i-1)+1:glin*i];
        Dhc=[Dhc gvec(veci,m-Graucol(i))];
    end
end
if find(coldeg(g))==[];
    d=det(Dhc)/detU;return;
end;
for i=1:glin;
    veci=[glin*(i-1)+1:glin*i];
    Dlc=[Dlc gvec(veci,m-Graucol(i)+1:end)];
end
ac=[];
for i=1:length(Graucol);
    ac=blkdiag(ac,[zeros(1,Graucol(i));
    eye(Graucol(i)-1,Graucol(i))]);
end
Dhcinv=inv(Dhc);prod=Dhcinv*Dlc; acpos=[1]; for
i=1:length(Graucol)-1;
    acpos=[acpos norm(Graucol(1:i),1)+1];
end
```

```
for i=1:glin;
    ac(acpos(i),:)=-prod(i,:);
end
d=det(Dhc)*poly(ac)/detU;
```

A.22 RCMFD

```
function [m,n]=rcmfd(ng,d,q);
[p,lng]=size(ng);
if nargin==2;
    q=p;
end
if size(d,1)==1
    dd=[];
    for i=1:p;
        dd=[dd;d];
    end;
    a=diaglamb(dd);
else
    a=d;
end
b=ng;t=matform(b,-a,q,p);
f=polybasesnew1(t,p,p+q);
m=f(1:q,:);n=f(q+1:p+q,:);
```

A.23 LCMFD

```
function [mtil,ntil]=lcmfd(m,n); p=size(n,1); q=size(m,1);
mt=transpol(m); nt=transpol(-n,q); t=matform(nt,mt,p,q);
f=polybasesnew1(t,q,q+p); ntilt=f(p+1:p+q,:);
ntil=transpol(ntilt,p); mtilt=f(1:p,:); mtil=transpol(mtilt);
```

A.24 MDC

```
function [mtil,ntil,mdcma]=mdc(p,q,ncol);
[mtil,ntil]=lcmfd(q,p);
mdcma=[];
[m,n]=rcmfd(ntil,mtil,ncol);
[nm,dm]=inverse(m);
cutzeros(convmat(nm,m)); mdcn=convmat(nm,q);
mdcnvec=matsplit(mdcn); mdcvec=[];
for i=1:size(mdcnvec,1);
       [qc,r]=rcmfd2(mdcnvec(i,:),dm,1);
       mdcvec=[ zeros(1,(size(r,2)-size(mdcvec,2))) mdcvec;
       zeros(1,(size(mdcvec,2)-size(r,2))) r/qc(end)];
end;
mdcma=vecmat(mdcvec,ncol);
```

A.25 Bezoutleft

```
function [xtil,ytil]=bezoutleft(m,n);
p=size(n,1);q=size(m,1);
m=transpol(m);n=transpol(n,q); t=matform(m,-n,q,p); nbase=0;j=1;
while nbase<q;
    cjt=convform(t,j,q+p);
    cjt=[cjt [zeros(size(cjt,1)-q,q);-eye(q)]];
    [lcjt,ccjt]=size(cjt);[xj,sj,yj]=svd(cjt);
    nj=NumSingNull(diag(sj),10^-9);
    if ccjt>lcjt;ncj=nj+ccjt-lcjt;else ncj=nj;end
    for i=1:ncj;
      if nbase<q;
        if nbase==0;
            f=yj(:,ccjt-i+1);fbot=f(end-q+1:end,:);nbase=1;
```

```
        else
             fi=yj(:,ccjt-i+1);fboti=[fbot fi(end-q+1:end,:)];
            if rank(fboti,1e-9)==nbase+1;
                [mf,nf]=size(f);f=[[zeros(ccjt-mf,nf);f] fi];
                nbase=nbase+1;
                fbot=[fbot fi(end-q+1:end,:)];
            end
        end
      end
    end
    j=j+1;
end
[lf,cf]=size(f);lff=lf-cf;
ff=f(1:lff,:)*inv(f(lff+1:lf,:));
c=f(lff+1:lf,:);ff
xyhat=[];
for i=1:p+q:lff;
    xyhat=[xyhat ff(i:i+p+q-1,:)];
end
xhat=xyhat(1:q,:);xtil=transpol(xhat,q);
yhat=xyhat(q+1:end,:);ytil=transpol(yhat,q);
```

A.26 Bezoutright

```
function [x,y]=bezoutright(m,n,q,p);
t=matform(m,-n,q,p);
nbase=0;j=1;
while nbase<p;
    cjt=convform(t,j,q+p);
    cjt=[cjt [zeros(size(cjt,1)-p,p);-eye(p)]];
```

```
    [lcjt,ccjt]=size(cjt);[xj,sj,yj]=svd(cjt);
    nj=NumSingNull(diag(sj),10^-9);
    if ccjt>lcjt;ncj=nj+ccjt-lcjt;else ncj=nj;end
    for i=1:ncj;
      if nbase<p;
        if nbase==0;
            f=yj(:,ccjt-i+1);fbot=f(end-p+1:end,:);nbase=1;
        else
             fi=yj(:,ccjt-i+1);fboti=[fbot fi(end-p+1:end,:)];
            if rank(fboti,1e-9)==nbase+1;
                [mf,nf]=size(f);f=[[zeros(ccjt-mf,nf);f] fi];
                nbase=nbase+1;fbot=[fbot fi(end-p+1:end,:)];
            end
        end
      end
    end
    j=j+1;
end
[lf,cf]=size(f);lff=lf-cf;
ff=f(1:lff,:)*inv(f(lff+1:lf,:));
c=f(lff+1:lf,:);ff
xyhat=[];
for i=1:p+q:lff;
    xyhat=[xyhat ff(i:i+p+q-1,:)];
end y=xyhat(1:q,:); x=xyhat(q+1:end,:);
```

A.27 Polybezout

```
function [m,n,mtil,ntil,x,y,xtil,ytil]=polybezout(n,d,ncol);
if nargin==3 [m,n]=rcmfd(n,d,ncol);else m=n;n=d;end
```

```
[lm,cm]=size(m);[ln,cn]=size(n);
[mtil,ntil]=lcmfd(m,n);mtil=cutzeros(mtil),ntil=cutzeros(ntil,lm)
[xtil,ytil]=bezoutleft(m,n) [lytil,cytil]=size(ytil);
[lxtil,cxtil]=size(xtil); [lmtil,cmtil]=size(mtil);
[lntil,cntil]=size(ntil); t4m=[zeros(lxtil,cntil-cxtil)
xtil;zeros(lntil,cxtil-cntil)-ntil];
 t4n=[zeros(lytil,cmtil-cytil) ytil;zeros(lmtil,cytil-cmtil) -mtil];
[x,y]=bezoutright(t4m,t4n,lm,ln)
```