xx (979C7.1)

CINVESTAV

Centro de Investigación y de Estudios Avanzados del IPN
Unidad Guadalajara

Desarrollo de Algoritmos Numéricos para Teoría de Control.

Tesis que presenta
Juan Carlos Zúñiga Anaya

Para obtener el grado de
Maestro en Ciencias

En la especialidad de
Ingeniería Eléctrica

Guadalajara, Jal., Julio de 2001.

Desarrollo de Algoritmos Numéricos para Teoría de Control.

Tesis de Maestría en Ciencias Ingeniería Eléctrica

Por

Juan Carlos Zúñiga Anaya

Ingeniero en Electrónica
Instituto Tecnológico y de Estudios Superiores de Occidente 1995-1999

Becario del CONACYT, expediente no. 143920

Director de tesis
Dr José Javier Ruiz León

CINVESTAV del IPN Unidad Guadalajara, Julio de 2001

INDICE GENERAL.

Notación utilizada.

$\Re$: Conjunto de los números reales.

$\Re[s]$: Conjunto de los polinomios en s con coeficientes reales.

$\Re(s)$: Conjunto de las funciones racionales en s con coeficientes reales.

$\Re_p(s)$: Conjunto de las funciones racionales propias en s con coeficientes reales.

$f : A \rightarrow B$ Función que mapea del espacio vectorial A al espacio vectorial B.

$f(x)$: Función con parámetros x.

$|a|$ Valor absoluto de a.

$\|a\|$ Norma de a.

A^T: Matriz A transpuesta.

I_n: Matriz identidad de nxn.

Im(A): Imagen de la matriz A.

ker(A): Kernel de la matriz A.

$rank(A)$: Rango de la matriz A.

$v(A)$: Nulidad de la matriz A.

$col_i A$: Columna i-esima de A.

$row_i A$: Fila i-esima de A.

$\deg x(s)$: Grado del polinomio $x(s)$.

$\deg_p f(s)$: Grado relativo de la función racional propia $f(s)$

$span\{v_1, v_2, ..., v_m\}$: Espacio vectorial generado por los vectores $v_1, v_2, ..., v_m$

$diag\{a_1, a_2, ..., a_n\}$: Matriz diagonal de $n \times n$ que tiene los valores a_i en la diagonal.

$sign(x)$: Función signo de x.

min: La función $\min\{a_1, a_2, ..., a_m\}$ es igual al menor de los elementos $a_1, a_2, ..., a_m$.

max: La función $\max\{a_1, a_2, ..., a_m\}$ es igual al mayor de los elementos $a_1, a_2, ..., a_m$

K $\times 10^n$: representa la notación científica del número K. Por ejemplo, $0.0345 = 34.5 \times 10^{-3}$

Introducción.

En la actualidad, los avances computacionales, tanto de software como de hardware permiten a diferentes ramas de la ciencia apoyarse en algoritmos computacionales para la solución y el análisis de un gran número de problemas.

Estos algoritmos computacionales capaces, en un principio de resolver únicamente problemas numéricos, han evolucionado y actualmente, gracias a la introducción de los lenguajes computacionales de matemáticas simbólicas, como por ejemplo MAPLE [23] o *Symbolic Math Toolbox* de MATLAB [25], es posible la solución de problemas totalmente simbólicos. Podemos entonces distinguir, a grandes rasgos, dos tipos de algoritmos, los que manejan unícamente números y que son llamados algoritmos númericos y aquellos que tienen la capacidad de trabajar con símbolos y con variables no valuadas. Este segundo tipo de algoritmo es llamado algoritmo simbólico.

En el campo del control automático, estos avances en el desarrollo de algoritmos computacionales no han sido la excepción. Actualmente son muchas las herramientas y algoritmos computacionales que se utilizan para resolver y analizar problemas de control desde un nivel teórico e incluso en la implementación real de los mismos. Por ejemplo, en la teoría de control clásico, donde los sistemas lineales monovariables e invariantes en el tiempo son representados por medio de una función de transferencia, el desarrollo de algoritmos numéricos computacionales nos permite hacer todo tipo de análisis, tales como: Gráficas de Bode, diagramas de Nyquist, gráficas del lugar de raíces, gráficas de la respuesta al escalón o al impulso unitario, etc. El avance y desarrollo del álgebra numérica, base de la representación de sistemas en el espacio de estado, también ha permitido la creación de un gran número de algoritmos numéricos para el análisis de sistemas de control desde este enfoque (las variables de estado), por ejemplo: la determinación de la controlabilidad y observabilidad de un sistema, obtención de valores propios y ubicación de polos, etc.

De igual forma, actualmente se cuenta con algoritmos numéricos desarrollados para casi todas las técnicas de control modernas, por ejemplo: Control no lineal, lógica difusa, redes neuronales, control robusto, control óptimo, etc.

Por otro lado, la creación de algoritmos computacionales para el enfoque polinomial de la teoría de control, es un campo de investigación que apenas desde hace algunos años, empezó a desarrollarse. El enfoque polinomial, iniciado por Kučera [19], es una elegante manera de analizar sistemas y problemas de control. En este enfoque, básicamente, la solución de un problema de control se reduce a la solución de alguna ecuación polinomial o ecuación Diofantina asociada al mismo [18]. Para esto se requiere la manipulación de polinomios y matrices polinomiales, es decir, matrices cuyos elementos no son números sino polinomios en la variable compleja s. Por lo tanto, los algoritmos del álgebra numérica aplicados en el enfoque de las variables de estado, por ejemplo, no pueden ser utilizados y es necesaria la creación de nuevos algoritmos capaces de manejar este tipo de matrices.

Una solución que parecería lógica, es la de desarrollar algoritmos simbólicos donde s sea visto como una variable no valuada, sin embargo, la cantidad de tiempo y recursos de memoria que estos algoritmos simbólicos pueden requerir para resolver problemas complicados como la solución de ecuaciones polinomiales o la obtención de formas canónicas, puede ser un gasto innecesario si el problema puede resolverse numéricamente. Así pues, la posibilidad de desarrollar algoritmos numéricos capaces de manejar matrices

polinomiales de una manera eficiente, con una exactitud aritmética comparable a la conseguida por los algoritmos simbólicos, y sin sacrificar recursos computacionales como tiempo de ejecución, y memoria, se ha vuelto un campo de investigación importante.

Algunos de los principales trabajos realizados recientemente en esta área son los siguientes:

State-Space Algorithms for Polynomial Matrix Operations [16]. Este trabajo realizado por Ferdinand Kraffer en 1998 se orienta a la solución de las operaciones con matrices polinomiales desde el espacio de estado. Este trabajo presenta algortimos numéricos para transformar una operación polinomial en un sistema en el espacio de estado, posteriormente presenta técnicas para la manipulación y solución de ese sistema (teórico) en el espacio de estado y finalmente, técnicas para regresar del espacio de estado y presentar la solución de la operación polinomial.

Reliable Algorithms for Polynomial Matrices [8]. Este trabajo, realizado por Didier Henrion también en 1998 presenta algoritmos numéricos basados en técnicas polinomiales como interpolación y matrices de Sylvester. El desarrollo de estos algoritmos está enfocado a realizar eficientemente, y sin pasar por el espacio de estado, operaciones como la evaluación del rango de matrices polinomiales, la triangularización de las mismas y la factorización espectral.

Numerical Algorithms for Polynomial Matrices [13]. Realizado en 1999 por Martin Hromčík. Aqui se presenta una nueva técnica de solución de problemas con matrices polinomiales, basada en la interpolación y la transformada rápida de Fourier. Se presentan una serie de algoritmos, algunos de ellos ya estudiados en trabajos anteriores, pero ahora resueltos con la transformada rápida de Fourier.

The Polynomial Toolbox [22]. Desarrollado por un grupo de investigadores encabezados por: Huibert Kwakernaak, Rens C.W. y Michael Sebek. The Polynomial Toolbox, está desarrollado para trabajar en MATLAB 5.x. Es un conjunto de nuevos algoritmos rápidos y confiables basados en avanzados métodos polinomiales que constituye la plataforma básica en la que se han programado los algoritmos de los otros trabajos.

El objetivo del presente trabajo es, primeramente, estudiar el estado del arte de esta clase de nuevos algoritmos numéricos que se han desarrollado y que se están desarrollando para el enfoque polinomial. Para esto, se presentarán también conceptos básicos del algebra numérica y los algoritmos numéricos en general. Un segundo objetivo que se planteó para este trabajo, fue el de desarrollar nuevos resultados y algoritmos para el estudio de sistemas lineales.

En particular, se logró el desarrollo de 3 nuevos algoritmos, los cuales fueron programados sobre la plataforma de MATLAB y el Polynomial Toolbox, y que constituyen la principal contribución de este trabajo.

El primero de estos algoritmos obtiene la matriz interactor de un sistema lineal multivariable. Esta matriz interactor es una matriz polinomial, invariante bajo retroalimentación de estado estática regular, que contiene la información al infinito del sistema y por lo tanto es un ingrediente fundamental en la solución de algunos problemas de control. El interactor de un sistema se puede obtener mediante operaciones elementales por columnas sobre el anillo de las funciones racionales propias, dada la relación que guarda con la forma de Hermite por columnas de la matriz de transferencia del mismo sistema. Sin embargo, estas operaciones elementales son numéricamente inestables e ineficientes al momento de utilizarlas en un algoritmo. En este trabajo se presenta un algoritmo alternativo que obtiene el interactor sin recurrir a las operaciones elementales. Este algoritmo basado únicamente en operaciones numéricamente estables es un algoritmo de los que llamaremos algoritmos confiables.

Con base en la información contenida en el interactor se pueden resolver algunos problemas de control, como el desacoplamiento de sistemas lineales, así como obtener la estructura al infinito de cualquier matriz

racional. La solución de este par de problemas se abordó en los otros dos algoritmos desarrollados en este trabajo.

El segundo algoritmo determina si el sistema lineal analizado puede desacoplarse mediante retroalimentación dinámica o estática. En el caso de sistemas cuadrados que sean desacoplables por retroalimentación estática, también encuentra una retroalimentación que logra el desacoplamiento. En este algoritmo, la matriz interactor juega un papel muy importante tanto en la prueba que indica si el sistema es desacoplable como en la determinación de la retroalimentación. Este algoritmo, basado en operaciones estables y en el algoritmo que obtiene el interactor, también podrá ser considerado como un algoritmo confiable.

El tercer y último algoritmo desarrollado resuelve el problema de obtener la estructura al infinito de cualquier matriz racional en general. Para este algoritmo se hace una generalización de la matriz interactor y una modificación práctica al algoritmo que la obtiene, de esta forma es posible obtener lo equivalente al interactor de cualquier matriz racional. Con base en este equivalente del interactor, se logra determinar la estructura al infinito de la matriz analizada.

Para la presentación de estos resultados asi como del estado del arte del desarrollo de algoritmos numéricos para el enfoque polinomial, se ha dividido el presente trabajo de la siguiente forma:

En el Capítulo 1 se presentan, primeramente, algunos preliminares. Se introduce el concepto de problema numérico y de algoritmo numérico como método de solución del primero. Se habla de algunos conceptos importantes como la estabilidad numérica, la precisión, exactitud y resolución y finalmente se introduce el concepto de algoritmo confiable y los métodos del algebra numérica en los que se basa este tipo de algoritmos.

En el Capítulo 2 se exponen los tipos de algoritmos que se utilizan en los distintos enfoques de la teoría de control y se presentan los problemas y el estado del arte de los algoritmos aplicados en el enfoque polinomial, base de este trabajo. Finalmente se exponen los métodos polinomiales avanzados en los que se basan estos algoritmos numéricos confiables para la teoría de control, vista desde el enfoque polinomial, y se presentan dos algoritmos representativos, uno para obtener el determinante [9] y el otro para obtener el rango [8] de matrices polinomiales.

A partir del Capítulo 3 se presentan los resultados originales obtenidos durante el desarrollo de este trabajo.

En el Capítulo 3 se presenta el algoritmo que obtiene la matriz interactor. Primeramente se introduce el concepto de interactor y la teoría necesaria para obtenerlo. En la segunda parte del capítulo se presenta el algoritmo desarrollado.

El algoritmo sobre el desacoplamiento de sistemas lineales es presentado en el Capítulo 4. Este algoritmo representa una aplicación directa del interactor y por lo tanto del algoritmo presentado en el Capítulo 3. De igual forma, primeramente se expone el problema del desacoplamiento de sistemas lineales así como la teoría necesaria para resolverlo y posteriormente, en una segunda parte del capítulo, se presenta el algoritmo.

En el Capítulo 5 se presenta otra aplicación del interactor más generalizada y no tan enfocada a un sistema físico real como en el caso del desacoplamiento. Esta aplicación es la obtención de la estructura al infinito de cualquier matriz racional por medio de un algoritmo basado en la obtención del interactor. Este capítulo también consta de dos secciones, en la primera se define la estructura al infinito de una matriz racional y se presenta la teoría necesaria que nos permitirá desarrollar el algoritmo para obtenerla. En la segunda sección se presenta el algoritmo en cuestión.

Finalmente, se formulan algunas conclusiones generales sobre el trabajo en sí y sobre los resultados obtenidos.

Capítulo 1

Preliminares.

En este capítulo se presenta la teoría básica de los algoritmos numéricos. También se definen algunos conceptos que se utilizarán a lo largo de este documento.

Se presenta el concepto de problema numérico y sus características y se introducen los algoritmos numéricos como métodos de solución de este tipo de problemas. Asi mismo, se plantean los conceptos de la estabilidad numérica, precisión, resolución y exactitud numérica, los cuales serán utilizados en los capítulos posteriores. Con base en estos conceptos se define un algoritmo numérico confiable y posteriormente se presentan las herramientas del álgebra lineal que son base de este tipo de algoritmos. El conocimiento de estas herramientas es importante ya que, si bien no serán directamente utilizadas, se encuentran como base de muchas de las operaciones numéricamente estables que realiza la computadora al momento de ejecutar alguno de los algoritmos numéricos aplicados al control que describiremos más tarde.

Es importante mencionar que los problemas y resultados sobre los aspectos numéricos, la estabilidad y las herramientas del álgebra numérica que serán presentados, han sido bien estudiados durante mucho tiempo y existe una gran cantidad de literatura sobre el tema. Sin embargo, en este trabajo solamente se presentan algunos resultados importantes sin demostración y sin profundizar demasiado en ellos, ya que su estudio está fuera de los alcances de esta tesis. Para más detalles sobre este tema se puede consultar por ejemplo [5, 31, 27, 7]

1.1 Problemas y algoritmos numéricos.

1.1.1 Los problemas numéricos y su solución.

Un problema numérico puede ser descrito como una función $f : \Re^m \longrightarrow \Re^n$ de la siguiente forma: $f(x) = y$, donde $x \in \Re^m$ son los datos que determinan el problema y $y \in \Re^n$ es la respuesta del mismo.

Si tomamos los datos x y resolvemos el problema obteniendo $f(x)$, es natural esperar que si se presenta una pequeña variación en x, el resultado del problema con los nuevos datos x' se encontrará cercano al resultado original, es decir, $f(x) \approx f(x')$. Este tipo de problemas numéricos son conocidos como problemas *bien condicionados (well conditioned)*. Sin embargo, no todos los problemas numéricos son bien condicionados, existen problemas en los que las soluciones $f(x)$ y $f(x')$ son muy distintas aún cuando $x \approx x'$. Este tipo de problemas es conocido como problemas *mal condicionados (ill conditioned)*. Es importante mencionar que los conceptos de cercanía o de lejanía entre dos números o resultados,

conceptos que seguiremos utilizando a lo largo de este trabajo, tienen una cuantificación formal que se introducirá cuando se hable, más adelante, de normas vectoriales y matriciales.

A continuación se presentan ejemplos de problemas mal condicionados y bien condicionados.

Ejemplo 1.1 *Consideremos la ecuación de quinto orden:*

$$x^5 + 7x^4 + 19.4x^3 + 26.6x^2 + 18.0384x + 4.8384 = 0$$

La solución de esta ecuación es nuestro problema numérico, es decir, la solución $y = f(x)$ son las raíces de la ecuación representada por f con los parámetros

$$x = \begin{bmatrix} 7 & 19.4 & 26.6 & 18.0384 & 4.8384 \end{bmatrix}$$

Para estos parámetros la solución es,

$$y = \begin{bmatrix} -1 & -1.2 & -1.4 & -1.6 & -1.8 \end{bmatrix}$$

Ahora supongamos que el vector de parámetros sufre una pequeña variación, es decir:

$$x' = \begin{bmatrix} 7.001 & 19.4 & 26.6 & 18.0384 & 4.8384 \end{bmatrix}$$

de esta forma:

$$y' = f(x') = \begin{bmatrix} -1.06 + 0.024j & -1.06 - 0.024j & -1.48 + 0.23j & -1.48 - 0.23j & -1.92 \end{bmatrix}$$

como se puede ver, las soluciones son muy diferentes, es decir, estamos hablando de un problema mal condicionado.

Ejemplo 1.2 *Ahora consideremos la ecuación de segundo orden:*

$$x^2 - 3.57x + 2.8782 = 0$$

La solución exacta de esta ecuación es $x_1 = 2.34$ y $x_2 = 1.23$.
Ahora consideremos una pequeña variación en los parámetros, es decir,

$$x^2 - 3.57x + 2.8783 = 0$$

la solución de esta ecuación es: $x_1 = 2.3399$ y $x_2 = 1.2301$. Este es un problema bien condicionado.

El hecho de que existan estos dos tipos de problemas numéricos toma importancia cuando en el proceso de solución se utilizan herramientas como computadoras o calculadoras. Estas herramientas generalmente trabajan con un sistema de punto decimal flotante en el cual el número de dígitos utilizados en la representación de los números y sus operaciones está limitado. Un número x en punto flotante es representado como sigue: $x = \pm 0.d_1d_2d_3...d_t \times B^e = MB^e$ Los dígitos $\pm d_1d_2d_3...d_t$ son conocidos como la *mantisa* M, la base B es generalmente 10 (numeración decimal), e es el exponente y t, que es el número de dígitos que representan a x, es lo que llamaremos resolución.

Es importante tener en mente que el hecho de que un problema sea mal condicionado depende más de la estructura del mismo y de la sensibilidad que tenga al cambio de parámetros y no tanto del hecho de que se resuelva por un algoritmo numérico con alguna resolución limitada.

Dada la limitación en la resolución, todo número debe ser aproximado según el número de dígitos t que se esté manejando, lo cual evidentemente ocasiona un error de redondeo que, en el caso de que el problema que se esté resolviendo sea mal condicionado, puede ser peligroso. El error de redondeo y la aproximación en punto flotante, se definen a continuación.

Definición 1.1 *Sea $x \in \Re$, y sea x_r su aproximación en punto flotante. El número x_r puede ser escrito como: $x_r = x(1+\partial)$. Por notación, la aproximación de un número o incluso del resultado de alguna operación en punto flotante, se escribe como: $fl(x) = x(1+\partial) = x_r$, donde ∂ es el error relativo que está acotado por $|\partial| \leq \varepsilon$. El valor de ε depende de la técnica de aproximación utilizada.*

a) Si la aproximación se realizó por redondeo (aproximación al entero inmediato superior), entonces: $\varepsilon \leq 0.5 \times 10^{1-t}$

b) Si la aproximación se realizó por truncamiento (aproximación al entero inmediato inferior), entonces: $\varepsilon \leq 1.0 \times 10^{1-t}$

El efecto del error de redondeo, el cual es inevitable, nos permite precisar dos conceptos más que seguiremos utilizando a lo largo de este trabajo. Estos son: la precisión y la exactitud[1]

Por exactitud entenderemos que tan cercano o lejano está un resultado de su valor real, es decir, si tenemos un problema numérico del cual conocemos el resultado exacto que es 3.78, por ejemplo, y nuestro algoritmo de solución nos entrega un resultado igual a 3.23 la primera vez que lo aplicamos y 3.77 la segunda vez, diremos que el segundo resultado es más exacto que el primero. También podemos decir que el algoritmo utilizado no es muy preciso, ya que nos dió un resultado diferente por cada vez que lo aplicamos. Por otro lado, si el algoritmo de solución nos entrega, cada vez que lo aplicamos, 2.34 como resultado, entonces diremos que es un algoritmo preciso pero que el resultado obtenido no es exacto.

En resumen, el conocimiento previo del tipo de problema que se va a resolver (mal o bien condicionado), los errores de redondeo en el algoritmo de solución y la resolución utilizada, nos darán idea de la confiabilidad, es decir, de la precisión y exactitud del resultado que se obtendrá.

1.1.2 Algoritmos numéricos y estabilidad.

La aplicación de un algoritmo numérico para resolver un problema $y = f(x)$ implica la definición de una nueva función f' la cual, para un vector de datos x, regrese una solución aproximada $f'(x)$ del problema. Una de las características más importantes del algoritmo f' es la estabilidad numérica. La estabilidad numérica de un algoritmo se define como sigue: se dice que un algoritmo es numéricamente estable si $\forall\, x \in D \subset \Re^m$, existe un x' tal que $f(x')$ es cercano a $f'(x)$. Lo anterior significa que si aplicamos un algoritmo numéricamente estable, la solución que obtengamos deberá estar cerca de la solución exacta del problema cuando los datos de entrada sufren una pequeña variación. Esta pequeña variación puede ser considerada como los errores generados por las aproximaciones numéricas en las operaciones propias del mismo algoritmo.

De esta forma, si utilizamos un algoritmo numéricamente estable para resolver un problema bien condicionado en el que consideramos una aproximación x' de los datos de entrada, sabemos que la solución $f(x')$ estará cerca de la solución exacta $f(x)$ y también, por definición de estabilidad numérica, que la solución $f(x')$ estará cerca de $f'(x)$. En otras palabras, obtendremos una solución aproximada $f'(x)$ cercana a la solución exacta $f(x)$. Lo anterior se muestra gráficamente en la figura 1.1.

[1] Cabe mencionar que los conceptos de precisión y exactitud asi como el concepto de resolución son muchas veces englobados, en la literatura, bajo el término único de precisión. Es en el campo de la medición y la instrumentación donde generalmente se hace la distinción y se nombra de una forma diferente a cada concepto [3]. Para efectos de claridad, en este trabajo, hemos decidido nombrar y utilizar estos tres conceptos también de manera independiente.

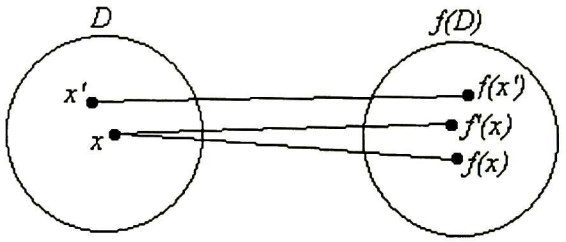

Figura 1.1. Aplicación de un algoritmo estable a un problema bien condicionado

Por otro lado, si utilizamos un algoritmo numéricamente estable para tratar de obtener la solución de un problema mal condicionado veremos como, aunque la solución $f'(x)$ obtenida por el algoritmo esté cerca de la solución $f(x')$ del problema con perturbación, dado que $f(x)$ está lejos de $f(x')$, la solución aproximada $f'(x)$ estará también alejada de la solución exacta $f(x)$. Sin embargo, la estabilidad del algoritmo asegura que no esté más alejada de lo que está $f(x')$. En otras palabras, la aplicación de un algoritmo estable en la solución de un problema mal condicionado, no asegura la obtención de un resultado cercano al resultado exacto, sin embargo tampoco introduce más error del que el propio problema presenta. Lo anterior se muestra gráficamente en la figura 1.2.

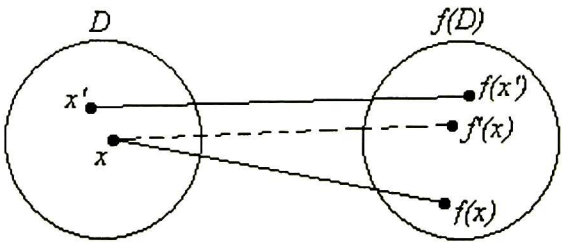

Figura 1.2. Aplicación de un algoritmo estable a un problema mal condicionado

Finalmente, con base en estos conceptos, podemos precisar lo que entenderemos como un *algoritmo numérico confiable*. Este concepto lo seguiremos utilizando a lo largo de este trabajo, ya que lograr algoritmos numéricos confiables para aplicarlos al estudio de sistemas de control es uno de los objetivos principales del mismo.

Llamaremos algoritmo numérico confiable a todo algoritmo que sea numéricamente estable, preciso y con una exactitud aceptable, es decir con una diferencia, con respecto al resultado real, muy pequeña. Es evidente que la exactitud del resultado depende también de la resolución numérica que se maneje en la representación de los números y sus operaciones (a mayor resolución, mayor exactitud), sin embargo, es importante que un algoritmo numérico confiable, maneje las operaciones y los errores de aproximación de una manera adecuada, de modo que la exactitud alcanzada en el resultado sea la máxima posible dada la resolución fijada por la computadora o herramienta donde se implementa el algoritmo.

1.2 Álgebra numérica y algoritmos confiables.

Muchas son las herramientas del álgebra numérica que se utilizan para el estudio de los problemas numéricos, el estudio de los errores de redondeo y la distancia o error entre dos números. También son muchos los resultados aplicados para manejar las matrices numéricas dentro de un algoritmo u operación numérica de modo que el error en el resultado final sea el mínimo posible. En esta parte del capítulo se presentan algunos de estos resultados.

Es importante recordar que estos resultados se presentan más bien como parte de los antecedentes y del estado del arte de los algoritmos numéricos y no tanto porque sean directamente utilizados en algunos de los algoritmos que presentaremos. Algunas referencias sobre estos temas son [5, 31, 27, 7].

1.2.1 Normas vectoriales y matriciales.

Dada la forma en la que un problema numérico es presentado, el álgebra de matrices y vectores y el álgebra numérica constituye la herramienta matemática principal para su análisis. En particular, los conceptos de lejanía o cercanía entre las soluciones, o entre los datos de un problema numérico, son cuantificados en base a los conceptos de error y norma de vectores y matrices.

Primeramente se presentan los conceptos de norma matricial y vectorial y algunos resultados interesantes.

Definición 1.2 *La norma de un vector es una función* $f : \Re^n \longrightarrow \Re$, *que satisface las siguientes propiedades:*

a) $f(x) > 0, \quad \forall x \in \Re^n, \quad x \neq 0$

b) $f(x+y) \leq f(x) + f(y), \quad \forall x, y \in \Re^n$

c) $f(ax) = |a|\, f(x) \quad \forall x \in \Re^n, \quad \forall a \in \Re$

d) Si $x = 0$,*entonces,* $f(x) = 0$

Por notación, $f(x)$ *se escribe como* $\|x\|$

Algunas de las normas más usadas en álgebra numérica son la *norma p*, definida como $\|x\|_p = (|x_1|^p + |x_2|^p + ... + |x_n|^p)^{1/p}$ y la *norma infinita* definida como $\|x\|_\infty = \max\{|x_i| : i = 1, 2, ..., n\}$

Sean $x, y \in \Re^n$, las siguientes propiedades se verifican:

a) $|x^T y| \leq \|x\|_p \|y\|_q \;$ con $\frac{1}{p} + \frac{1}{q} = 1$

b) $\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\, \|x\|_2$

c) $\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\, \|x\|_\infty$

d) $\|x\|_\infty \leq \|x\|_1 \leq \sqrt{n}\, \|x\|_\infty$

Un caso especial de la propiedad a) es la desigualdad $|x^T y| \leq \|x\|_2 \|y\|_2$ conocida como *Desigualdad de Cauchy*. La igualdad se verifica cuando $x = y$, es decir, $x^T x = \|x\|_2^2$.

La norma de una matriz puede definirse de manera análoga a la norma vectorial, es decir, como una función $f : \Re^{m \times n} \longrightarrow \Re$ que cumple con las propiedades mencionadas. La notación es similar, $f(A) = \|A\|$ donde $A \in \Re^{m \times n}$.

Una de las normas matriciales más utilizadas es la *norma p* definida como:

$$\|A\|_p = \max \left\{ \frac{\|Ax\|_p}{\|x\|_p} : x \neq 0, \ p = 1, 2, ..., \infty \right\}$$

Con base en esta definición, se puede probar que:

$$\begin{aligned} \|A\|_1 &= \max \left\{ \sum_{i=1}^{m} |a_{ij}| : j = 1, 2, ..., n \right\} \\ \|A\|_\infty &= \max \left\{ \sum_{j=1}^{n} |a_{ij}| : i = 1, 2, ..., n \right\} \end{aligned}$$

donde a_{ij} es el elemento (i, j) de la matriz A.

Otra norma muy utilizada es la *norma Frobenius* que se define como $\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}$.

Sean $A \in \Re^{m \times n}$ y $x \in \Re^n$, las siguientes propiedades se verifican:

a) $\|A\|_2 \leq \|A\|_F \leq \sqrt{n} \, \|A\|_2$

b) $\max\{|a_{ij}| : i = 1, 2, ..., m \ \ j = 1, 2, ..., n\} \leq \|A\|_2 \leq \sqrt{mn} \, \max\{|a_{ij}| : i = 1, 2, ..., m \ \ j = 1, 2, ..., n\}$

c) $\frac{1}{\sqrt{n}} \|A\|_\infty \leq \|A\|_2 \leq \sqrt{m} \, \|A\|_\infty$

d) $\frac{1}{\sqrt{m}} \|A\|_1 \leq \|A\|_2 \leq \sqrt{n} \, \|A\|_1$

e) $\|Ax\| \leq \|A\| \, \|x\|$

f) $\|Ax\|_2 \leq \|A\|_F \, \|x\|_2$

g) $\|x\|_2 = \|x\|_F$

Estos conceptos y resultados nos ayudan a definir de una manera más formal el error relativo entre dos números, dos vectores o dos matrices. A continuación se presenta esta definición y también algunos resultados importantes sobre los errores relativos.

Definición 1.3 *Sean α y β dos números donde β es una aproximación numérica de α.*

El residuo de β es: $\alpha - \beta$

El error absoluto es: $\|\alpha - \beta\|$

El residuo relativo es: $\frac{\alpha-\beta}{\alpha}$, $\alpha \neq 0$

El error relativo es: $\left\|\frac{\alpha-\beta}{\alpha}\right\|$, $\alpha \neq 0$

En el caso de que α o β sean matrices, esta definición se aplica a cada elemento de ellas, es decir, se hablará por ejemplo de la matriz de errores relativos, donde el elemento ij – ésimo será igual a:

$$\left\|\frac{\alpha_{ij} - \beta_{ij}}{\alpha_{ij}}\right\| \text{ con } \alpha_{ij} \neq 0$$

Dada la definición de error relativo, el orden de magnitud del mismo indica el número de cifras en las que la aproximación es igual al valor real. Es decir, si el error relativo es del orden de 10^{-n}, entonces α y β coinciden en n cifras significativas. En el siguiente ejemplo se ilustran algunos de estos conceptos.

Ejemplo 1.3 *Consideremos que el siguiente número 1.7347 es parte de algún algoritmo numérico con una resolución numérica de 4 digitos. Al representar este número en la resolución manejada por el algoritmo tendremos 1.735, es decir:*

$$fl(1.7347) = 1.7347(1 + \partial) = 1.735$$

donde ∂ es el error relativo que podemos acotar como: $|\partial| \leq \varepsilon$ donde ε dado que la aproximación se hizo por redondeo y no por truncamiento, está acotado como: $\varepsilon \leq 0.5 \times 10^{1-t} = 0.5 \times 10^{-3}$ Es decir:

$$|\partial| \leq 0.5 \times 10^{-3}$$

El valor exacto del error relativo ∂ lo podemos obtener como se indicó en la definición anterior, es decir:

$$\partial = \left|\frac{1.7347 - 1.735}{1.7347}\right| = 0.17294 \times 10^{-3}$$

para comprobar podemos observar que efectivamente:

$$1.7347(1 + \partial) = 1.7347(1 + .00017294) = 1.735$$

por último podemos verificar que, dado que el error relativo es del orden de 10^{-3}, efectivamente 1.7347 y su aproximación 1.735 coinciden en 3 cifras.

Por último, se considera el siguiente resultado interesante para matrices. Sea $A \in \Re^{m \times n}$ y su aproximación $\hat{A} = [\hat{a}_{ij}] = [a_{ij}(1 + e_{ij})] = A + E$, donde E es la matriz de los errores relativos. Entonces se cumple que:

a) $\|E\|_F \leq \varepsilon \|A\|_F$

b) $fl(A + B) = A + B + F$, donde $\|F\|_F \leq \varepsilon \|A + B\|_F$

c) $fl(AB) = AB + F$, donde $\|F\|_F \leq n\varepsilon \|A\|_F \|B\|_F$

La demostración de estos resultados se puede revisar en cualquier libro de álgebra numérica como [5, 27].

1.2.2 Matrices ortogonales.

Las matrices ortogonales son muy estudiadas dentro del álgebra numérica y los algoritmos numéricos ya que son la base de las transformaciones ortogonales las cuales tienen propiedades numéricas interesantes que mencionaremos más adelante.

Definición 1.4 *Un conjunto de vectores* $(x_1, x_2, ...x_p)$ *en* $\Re^n$ *se dice ortogonal si* $\forall i, j = 1, 2, ...p,\ x_i^T x_j = 0$. *Si además* $\|x\|_2 = 1$, *se dice que los vectores son ortonormales.*

Definición 1.5 *Una matriz* $A \in \Re^{m \times n}$ *se dice ortogonal o unitaria si* $A^T A = I_n$, *donde* I_n *es la matriz identidad de* $n \times n$.

Esto significa que las columnas de una matriz ortogonal forman una base ortonormal de $\Re^n$.

Una de las propiedades numéricas más importantes de las matrices ortogonales es que cuando son usadas para transformaciones, no modifican las normas principales de los vectores o matrices transformadas [5, 27], esto es:

a) $\|\hat{x}\|_2 = \|Qx\|_2 = \|x\|_2$ para cualquier matriz ortogonal Q.

b) $\left\|\hat{A}\right\|_2 = \|QAZ\|_2 = \|A\|_2$ para cualquier Q y Z ortogonales.

c) $\left\|\hat{A}\right\|_F = \|QAZ\|_F = \|A\|_F$ para cualquier Q y Z ortogonales.

1.2.3 Transformaciones Ortogonales.

Muchos de los problemas numéricos son resueltos reduciendo las matrices que los representan en formas más sencillas por medio de transformaciones ortogonales. Las transformaciones ortogonales aseguran además, la estabilidad numérica en el algoritmo de solución, tienen la propiedad de no aumentar los errores de redondeo y como mencionamos, mantienen invariante la norma de vectores y matrices, es por eso que son muy usadas.

Las transformaciones ortogonales básicas son dos: *Reflexiones Householder* y *Rotaciones Givens*. Estas transformaciones permiten entre otras cosas lograr un tipo de factorización de matrices muy usado en el álgebra numérica, *la factorización QR*. Además, la descomposición en valores singulares, como se verá más adelante, también es otra factorización importante basada en las matrices ortogonales.

Reflexiones Householder.

Definición 1.6 *Una matriz o transformación Householder es una matriz ortogonal de la forma:*

$$U = I - \frac{2}{u^T u} u u^T$$

donde $u \neq 0$ *es un n-vector conocido como vector Householder.*

Una de las principales utilidades de las transformaciones Householder es que pueden introducir la cantidad de ceros necesaria para transformar un vector x en la forma canónica e_i, y por lo tanto, por medio de una o varias transformaciones Householder podemos diagonalizar o triangularizar una matriz.

En otras palabras, para un vector dado x, es posible encontrar una reflexión U tal que: $Ux = -sign(x_1)\|x\|_2 e_1$. El vector Householder u puede escogerse como: $u = x + sign(x_1)\|x\|_2 e_1$

Ejemplo 1.4 *Consideremos* $x = [1, -2, 2, 4]^T$ *Obtengamos la matriz* U *que transforma a* x *en un vector canónico.*

Tomemos $u = x + \|x\|_2 e_1 = [6, -2, 2, 4]^T$ *por lo tanto la reflexión es:*

$$U = I - \frac{2}{u^T u} uu^T = \begin{bmatrix} -\frac{1}{5} & \frac{2}{5} & -\frac{2}{5} & -\frac{4}{5} \\ \frac{2}{5} & \frac{13}{15} & \frac{2}{15} & \frac{4}{15} \\ -\frac{2}{5} & \frac{2}{15} & \frac{13}{15} & -\frac{4}{15} \\ -\frac{4}{5} & \frac{4}{15} & -\frac{4}{15} & \frac{7}{15} \end{bmatrix}$$

Para comprobar obtengamos $Ux = \begin{bmatrix} -\frac{1}{5} & \frac{2}{5} & -\frac{2}{5} & -\frac{4}{5} \\ \frac{2}{5} & \frac{13}{15} & \frac{2}{15} & \frac{4}{15} \\ -\frac{2}{5} & \frac{2}{15} & \frac{13}{15} & -\frac{4}{15} \\ -\frac{4}{5} & \frac{4}{15} & -\frac{4}{15} & \frac{7}{15} \end{bmatrix} \begin{bmatrix} 1 \\ -2 \\ 2 \\ 4 \end{bmatrix}$

es decir, $Ux = \begin{bmatrix} -5 \\ 0 \\ 0 \\ 0 \end{bmatrix} = -sign(x_1)\|x\|_2 e_1$

Obviamente, si el ejemplo anterior se realizara numéricamente, es decir, evaluando las fracciones y tomando su aproximación en punto flotante con alguna resolución finita, sería de esperarse que el resultado no fuera exacto, sin embargo, otra de las buenas propiedades que tiene el uso de las transformaciones ortogonales es que introducen muy poco error de aproximación. De hecho, se puede demostrar que: si $\bar{U}$ es la aproximación en punto flotante de U, entonces, $\left\|U - \bar{U}\right\| \leq (4n + 22)\epsilon$, donde ϵ depende de la técnica de aproximación usada (redondeo o truncamiento) y n es la dimensión de U.

Un procedimiento similar puede extenderse a todos los vectores columna de una matriz y lograr con esto una triangularización (reducción Householder) de la misma, además, el uso de este método de triangularización conserva las buenas propiedades numéricas de las transformaciones ortogonales. La reducción Householder se realiza de la siguiente forma:

Algoritmo 1.1 *(Reducción Householder).*

Consideremos $A = \begin{bmatrix} a_1 & & a_m \end{bmatrix} \in \Re^{n \times m}$ *La matriz triangular o trapezoidal superior* $A_{k+1} = U_k U_{k-1} ... U_1 A$ *se obtiene de la siguiente forma:*

1) Se obtiene la matriz U_1 *de modo que* $U_1 a_1 = [r_{11}, 0, 0, ..., 0]^T$ *de esta forma:*

$$A_2 = U_1 A = \begin{bmatrix} r_{11} & & r_{1m} \\ 0 & & \\ 0 & A_2' & \\ 0 & & \end{bmatrix} \quad con\ A_2' = \begin{bmatrix} a_2' & & a_m' \end{bmatrix}$$

2) Se obtiene la matriz U_2 de modo que $U_2 a_2' = [r_{22}, 0, 0, ..., 0]^T$ de esta forma:

$$A_3 = U_2 U_1 A = \begin{bmatrix} r_{11} & & & r_{1m} \\ 0 & r_{22} & & r_{2m} \\ 0 & 0 & & \\ 0 & 0 & A_3' & \\ 0 & 0 & & \end{bmatrix} \quad con\ A_3' = \begin{bmatrix} a_3'' & & a_m'' \end{bmatrix}$$

Si $n > m$, este proceso deberá seguir hasta que $k = m$ y entonces, la matriz trapezoidal superior resultante será:

$$A_{m+1} = \begin{bmatrix} r_{11} & & r_{1m} \\ & r_{22} & r_{2m} \\ & & \\ & 0 & r_{mm} \\ & & \\ & & \end{bmatrix}$$

Si $n < m$, el proceso deberá seguir hasta que $k = n - 1$ y la matriz trapezoidal superior resultante será:

$$A_n = \begin{bmatrix} r_{11} & & r_{1n} & & \\ & r_{22} & r_{2n} & & \\ & & & a_{n+1}^* & a_m^* \\ & 0 & & & \\ & & r_{nn} & & \end{bmatrix}$$

Por último, si $m = n$, el proceso seguirá hasta que $k = m - 1$ y entonces la matriz resultante será triangular superior de dimensiones $m \times m$.

Rotaciones Givens.

Definición 1.7 *Una rotación o transformación Givens en el plano (i, j) es una matriz de la forma:*

$$R = \begin{bmatrix} & & & col.\ i & & & col.\ j & & \\ & 0 & & 0 & & & 0 & & 0 \\ fila\ i & 0 & & p & 0 & & q & 0 & 0 \\ & & & 0 & 1 & & 0 & & \\ fila\ j & 0 & & -q & 0 & & p & 0 & 0 \\ & & & 0 & & & 0 & 1 & \\ & 0 & & 0 & & & 0 & & 1 \end{bmatrix}$$

donde $p^2 + q^2 = 1$.

Dada la forma de una rotación Givens, el producto $y = Rx$ es fácilmente resuelto, ya que y solo difiere de x en la i-esima y j-esima componente,

$$\begin{aligned} y_j &= px_j - qx_i \\ y_i &= px_i + qx_j \end{aligned}$$

Por esta propiedad, las rotaciones Givens son usadas, generalmente, para hacer cero algún elemento de cierto vector. Si x es un vector dado de $\Re^n$ se puede construir una transformación Givens R con

$$\begin{aligned} p &= \frac{a_i}{\sqrt{a_i^2 + a_j^2}} \\ q &= \frac{a_j}{\sqrt{a_i^2 + a_j^2}} \end{aligned}$$

de modo que $p^2 + q^2 = 1$. Con esta transformación, $y = Rx$ tiene la siguiente forma: $y_i = \sqrt{a_i^2 + a_j^2}$ y $y_j = 0$.

Ejemplo 1.5 *Consideremos el vector* $x = [4, -3, 1, 4]^T$ *Obtengamos la rotación* R *tal que el segundo elemento de* x *sea igual a cero.*

Tomemos $i = 1$ *y* $j = 2$, *entonces:*

$$\begin{aligned} p &= \frac{a_i}{\sqrt{a_i^2 + a_j^2}} = \frac{4}{5} \\ q &= \frac{a_j}{\sqrt{a_i^2 + a_j^2}} = \frac{-3}{5} \end{aligned}$$

por lo tanto, la rotación Givens es:

$$R = \begin{bmatrix} \frac{4}{5} & -\frac{3}{5} & 0 & 0 \\ \frac{3}{5} & \frac{4}{5} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De esta forma, $y = Rx$ *tiene la siguiente forma:*

$$y = \begin{bmatrix} \frac{4}{5} & -\frac{3}{5} & 0 & 0 \\ \frac{3}{5} & \frac{4}{5} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ -3 \\ 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 1 \\ 4 \end{bmatrix}$$

De igual forma que con las transformaciones o reflexiones Householder, las rotaciones Givens cuentan con buenas propiedades numéricas y el uso de algoritmos con este tipo de matrices asegura errores pequeños de aproximación numérica.

Finalmente, se presentan algunos resultados básicos sobre dos tipos de factorizaciones, la factorización QR y la descomposición en valores singulares. Estas factorizaciones se basan en las transformaciones ortogonales y por lo tanto, guardan las buenas propiedades numéricas de estas transformaciones. La factorización QR permite, además de la triangularización de matrices, conocer algunas propiedades importantes

(ver teorema 1.1) sobre las mismas que son útiles para algunos algoritmos numéricos. La descomposición en valores singulares también es una factorización muy usada en el álgebra numérica y en la creación de algoritmos numéricos, por ejemplo, para obtener el rango de matrices reales.

Factorización QR.

Definición 1.8 *La factorización QR de una matriz* $A \in \Re^{m\times n}$ *está dada por:* $A = QR$. *La matriz* $Q \in \Re^{m\times m}$ *es ortogonal y la matriz* $R \in \Re^{m\times n}$ *es triangular superior.*

Muchos son los algoritmos que se han desarrollado para obtener la factorización QR de una matriz, a continuación presentamos algunos de ellos:

Algoritmo 1.2 *(Householder QR)*
La matriz triangular superior se obtiene de A realizando reducción Householder, es decir:

$$R = U_k^T U_{k-1}^T ... U_1 A$$

La matriz ortogonal está formada por la multiplicación de las transpuestas de las matrices Householder necesarias para la reducción, es decir:

$$Q = U_1 U_2 ... U_k$$

De este modo, la matriz A puede escribirse como:

$$A = QR$$

Algoritmo 1.3 *(Givens QR)*
De igual forma, podemos obtener una matriz triangular superior a partir de A por medio de varias transformaciones Givens, es decir:

$$R = G_k^T G_{k-1}^T ... G_1^T A$$

La matriz ortogonal está formada de la misma forma:

$$Q = G_1 G_2 ... G_k$$

Finalmente, A puede escribirse como:

$$A = QR$$

A continuación se presenta un ejemplo de factorización QR. El ejemplo es resuelto numéricamente con 5 dígitos de resolución y después se presenta la solución simbólica obtenida por MAPLE para comparar la exactitud aritmética del algoritmo.

Ejemplo 1.6 *Obtener la factorización Householder QR de la matriz:*

$$A = \begin{bmatrix} 2 & 2 & 3 \\ 2 & 0 & 2 \\ 1 & 1 & 2 \end{bmatrix}$$

Primeramente aplicamos la reducción Householder sobre la matriz $A = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix}$
Tomamos:

$$u_1 = a_1 + sign(a_{11}) \|a_1\|_2 e_1 = \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} + \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ 1 \end{bmatrix}$$

por lo tanto,

$$U_1^T = I - \frac{2}{u_1^T u_1} u_1 u_1^T = \begin{bmatrix} -.66667 & -.66667 & -.33333 \\ -.66667 & .73333 & -.13333 \\ -.33333 & -.13333 & .93333 \end{bmatrix}$$

entonces:

$$A_2 = U_1^T A = \begin{bmatrix} -.66667 & -.66667 & -.33333 \\ -.66667 & .73333 & -.13333 \\ -.33333 & -.13333 & .93333 \end{bmatrix} \begin{bmatrix} 2 & 2 & 3 \\ 2 & 0 & 2 \\ 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} -3.0 & -1.6667 & -4.0 \\ -.00001 & -1.4667 & -.80001 \\ .00001 & .26667 & .60001 \end{bmatrix}$$

Tomemos ahora:

$$u_2 = a_2' + sign(a_{21}') \|a_2'\| e_1 = \begin{bmatrix} -1.4667 \\ .26667 \end{bmatrix} - \begin{bmatrix} 1.4907 \\ 0 \end{bmatrix} = \begin{bmatrix} -2.9574 \\ .26667 \end{bmatrix}$$

por lo tanto,

$$U_2'^T = \begin{bmatrix} -.98387 & .17889 \\ .17889 & .98387 \end{bmatrix}$$

de este modo,

$$U_2^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -.98387 & .17889 \\ 0 & .17889 & .98387 \end{bmatrix}$$

entonces:

$$R = U_2^T U_1^T A = \begin{bmatrix} -3.0 & -1.6667 & -4.0 \\ 1.1627\ e10^{-5} & 1.4907 & .89444 \\ 8.0498\ e10^{-6} & -3.9834\ e10^{-6} & .44722 \end{bmatrix}$$

y la matriz Q está dada por:

$$Q = U_1 U_2 = \begin{bmatrix} -.66667 & .59629 & -.44721 \\ -.66667 & -.74535 & 6.0166\ e10^{-6} \\ -.33333 & .29814 & .89442 \end{bmatrix}$$

Finalmente, A es factorizada como:

$$A = \begin{bmatrix} -.66667 & .59629 & -.44721 \\ -.66667 & -.74535 & 6.0166\ e10^{-6} \\ -.33333 & .29814 & .89442 \end{bmatrix} \begin{bmatrix} -3.0 & -1.6667 & -4.0 \\ 1.1627\ e10^{-5} & 1.4907 & .89444 \\ 8.0498\ e10^{-6} & -3.9834\ e10^{-6} & .44722 \end{bmatrix} = QR$$

$$= \begin{bmatrix} 2.0 & 2.0 & 3.0 \\ 2.0 & 4.5644\ e10^{-5} & 2.0 \\ 1.0 & .99999 & 2.0 \end{bmatrix} \approx \begin{bmatrix} 2 & 2 & 3 \\ 2 & 0 & 2 \\ 1 & 1 & 2 \end{bmatrix}$$

Como se puede ver, el error introducido por el redondeo es considerablemente pequeño, lo cual confirma las buenas propiedades numéricas de las transformaciones ortogonales. Podemos comparar las matriz obtenidas por el algoritmo con las matrices que se obtienen de la solución simbólica que son las siguientes:

$$A = \begin{bmatrix} 2 & 2 & 3 \\ 2 & 0 & 2 \\ 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & \frac{4}{15}\sqrt{5} & -\frac{1}{5}\sqrt{5} \\ \frac{2}{3} & -\frac{1}{3}\sqrt{5} & 0 \\ \frac{1}{3} & \frac{2}{15}\sqrt{5} & \frac{2}{5}\sqrt{5} \end{bmatrix} \begin{bmatrix} 3 & \frac{5}{3} & 4 \\ 0 & \frac{2}{3}\sqrt{5} & \frac{2}{5}\sqrt{5} \\ 0 & 0 & \frac{1}{5}\sqrt{5} \end{bmatrix} = QR$$

Es importante recordar que entre más dígitos se utilicen en la aproximación, la exactitud aritmética será mejor. A continuación se presenta la solución numérica obtenida con Matlab con 15 dígitos en la aproximación.

$$\begin{aligned} A &= \begin{bmatrix} -0.66666666666667 & 0.59628479399994 & -0.44721359549996 \\ -0.66666666666667 & -0.74535599249993 & -0.00000000000000 \\ -0.33333333333333 & 0.29814239699997 & 0.89442719099992 \end{bmatrix} \\ &\quad \begin{bmatrix} -3.00000000000000 & -1.66666666666667 & -4.00000000000000 \\ 0 & 1.49071198499986 & 0.89442719099992 \\ 0 & 0 & 0.44721359549996 \end{bmatrix} \\ &= \begin{bmatrix} 2.0 & 2.0 & 3.0 \\ 2.0 & 0 & 2.0 \\ 1.0 & 1.0 & 2.0 \end{bmatrix} = QR \end{aligned}$$

Teorema 1.1 *Sea* $A = \begin{bmatrix} a_1 & & a_n \end{bmatrix} \in \Re^{m\times n}$ *con* $rank(A) = n$. *Sea* $Q = \begin{bmatrix} q_1 & & q_m \end{bmatrix} \in \Re^{m\times m}$ *y* $R = \begin{bmatrix} r_1 & & r_n \end{bmatrix} \in \Re^{m\times n}$ *Si existe la factorización* $A = QR$ *entonces:*

a) $span\{a_1, a_2, ..., a_n\} = span\{q_1, q_2, ..., q_n\}$

b) $\mathrm{Im}(A) = \mathrm{Im}(Q_1)$ *y* $\mathrm{Im}(A)^{\perp} = \mathrm{Im}(Q_2)$
donde $Q_1 = \begin{bmatrix} q_1 & & q_n \end{bmatrix}$ *y* $Q_2 = \begin{bmatrix} q_{n+1} & & q_m \end{bmatrix}$

c) $A = Q_1R_1$ *donde* $R_1 = \begin{bmatrix} r_1 \\ \\ \\ r_n \end{bmatrix}$

Esta última forma de factorizar A es conocida como Thin QR factorization

Ejemplo 1.7 *Obtengamos la Thin QR factorization de:*

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ -1 & -2 & 0 \\ 0 & 0 & 9 \end{bmatrix}$$

El rango de A es igual a 3 y además, existe la factorización:

$$A = QR = \begin{bmatrix} -0.40824829046386 & 0.57735026918963 & -0.16222142113076 & -0.68824720161169 \\ -0.81649658092773 & -0.57735026918963 & 0.00000000000000 & 0.00000000000000 \\ 0.40824829046386 & -0.57735026918963 & -0.16222142113076 & -0.68824720161169 \\ 0 & 0 & -0.97332852678458 & 0.22941573387056 \end{bmatrix}$$
$$\begin{bmatrix} -2.44948974278318 & -4.08248290463863 & -4.49073119510249 \\ 0 & 0.57735026918963 & -0.57735026918963 \\ 0 & 0 & -9.24662100445346 \\ 0 & 0 & 0 \end{bmatrix}$$

Por lo tanto, podemos encontrar:

$$Q_1 = \begin{bmatrix} -0.40824829046386 & 0.57735026918963 & -0.16222142113076 \\ -0.81649658092773 & -0.57735026918963 & 0.00000000000000 \\ 0.40824829046386 & -0.57735026918963 & -0.16222142113076 \\ 0 & 0 & -0.97332852678458 \end{bmatrix}$$

$$R_1 = \begin{bmatrix} -2.44948974278318 & -4.08248290463863 & -4.49073119510249 \\ 0 & 0.57735026918963 & -0.57735026918963 \\ 0 & 0 & -9.24662100445346 \end{bmatrix}$$

de modo que la Thin QR Factorization está dada por:

$$Q_1 R_1 = \begin{bmatrix} 1.0 & 2.0 & 3.0 \\ 2.0 & 3.0 & 4.0 \\ -1.0 & -2.0 & 0.0 \\ 0 & 0 & 9.0 \end{bmatrix} = A$$

Descomposición en valores singulares.

Teorema 1.2 *SVD (Singular Value Decomposition)*
Si $A \in \Re^{m\times n}$ y $rank(A) = r$, existen matrices ortogonales

$$U = \begin{bmatrix} u_1 & & u_m \end{bmatrix} \in \Re^{m\times m}, \quad V = \begin{bmatrix} v_1 & & v_n \end{bmatrix} \in \Re^{n\times n}$$

tales que:

$$\Sigma = U^T A V = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} \in \Re^{m\times n}$$

donde,

$$\Sigma_r = diag(\sigma_1 \sigma_2 ... \sigma_r)$$

con,

$$r = \min(m, n), \quad \sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r \geq 0$$

Los números σ_i se conocen como los valores singulares de A.
Los vectores u_i son llamados vectores singulares izquierdos.
Los vectores v_i son los vectores singulares derechos.

De este teorema se deduce que una matriz A cualquiera también puede ser factorizada como:

$$A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

Algunas propiedades importantes son:

i) Los valores singulares coinciden con la raíz cuadrada de los valores propios positivos comunes de AA^T y A^TA.

ii) Si los valores singulares son $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_p \geq \sigma_{p+1} = \sigma_{p+2} = ... = \sigma_r = 0$, entonces:
$Ker(A) = span\{v_{p+1}, ..., v_n\}$
$\text{Im}(A) = span\{u_1, ..., u_p\}$

iii) $Av_i = \sigma_i u_i$ y $A^T u_i = \sigma_i v_i$ con $i = \min(m, n)$

Ejemplo 1.8 *Obtengamos la SVD de la siguiente matriz:*

$$A = \begin{bmatrix} 1 & 2 \\ 0 & -1 \\ 1 & 3 \end{bmatrix}$$

usando 4 dígitos de resolución. Primeramente obtengamos los valores singulares.

$$eig\left(\begin{bmatrix} 1 & 2 \\ 0 & -1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} 1 & 0 & 1 \\ 2 & -1 & 3 \end{bmatrix}\right) = eig\left(\begin{bmatrix} 5 & -2 & 7 \\ -2 & 1 & -3 \\ 7 & -3 & 10 \end{bmatrix}\right) = 0, 15.81, 0.1898$$

$$eig\left(\begin{bmatrix} 1 & 0 & 1 \\ 2 & -1 & 3 \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 0 & -1 \\ 1 & 3 \end{bmatrix}\right) = eig\left(\begin{bmatrix} 2 & 5 \\ 5 & 14 \end{bmatrix}\right) = 15.81, 0.1898$$

por lo tanto,

$$\Sigma = \begin{bmatrix} 3.9762 & 0 \\ 0 & 0.4357 \\ 0 & 0 \end{bmatrix}$$

Ahora, si se desea conocer las matrices U y V, estas se pueden calcular, a partir de la propiedad iii, de la siguiente forma:

$$\begin{bmatrix} 1 & 2 \\ 0 & -1 \\ 1 & 3 \end{bmatrix}\begin{bmatrix} v_{11} \\ v_{21} \end{bmatrix} = \begin{bmatrix} v_{11} + 2v_{21} \\ -v_{21} \\ v_{11} + 3v_{21} \end{bmatrix} = 3.976\begin{bmatrix} u_{11} \\ u_{21} \\ u_{31} \end{bmatrix}$$

además, usando el hecho de que U y V deben ser ortogonales, es decir, la norma de sus vectores columna debe ser igual a 1, entonces tenemos el siguiente sistema de ecuaciones:

$$\begin{aligned} v_{11} + 2v_{21} &= 3.976u_{11} \\ -v_{21} &= 3.976u_{21} \\ v_{11} + 3v_{21} &= 3.976u_{31} \\ u_{11}^2 + u_{21}^2 + u_{31}^2 &= 1 \\ v_{11}^2 + v_{21}^2 &= 1 \end{aligned}$$

cuya solución es:

$$u_1 = \begin{bmatrix} -.55856 \\ .23647 \\ -.79504 \end{bmatrix} \qquad v_1 = \begin{bmatrix} -.34043 \\ -.94027 \end{bmatrix}$$

asi, sucesivamente se puede comprobar que la SVD de la matriz A es:

$$A = \begin{bmatrix} -.55856 & -.59554 & -.57735 \\ .23647 & -.7815 & .57735 \\ -.79504 & .18596 & .57735 \end{bmatrix} \begin{bmatrix} 3.9762 & 0 \\ 0 & .4356 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} -.34043 & -.94027 \\ -.94027 & .34043 \end{bmatrix}$$

Teorema 1.3 *(The thin SVD)*
Sea $A \in \Re^{m\times n}$ una matriz cualquiera de rango r. Si existe la descomposición SVD

$$A = U\Sigma V^T$$

donde

$$U = \begin{bmatrix} u_1 & & u_m \end{bmatrix} \in \Re^{m\times m}, \quad V^T = \begin{bmatrix} v_1 \\ \\ v_n \end{bmatrix} \in \Re^{n\times n}$$

entonces:

$$A = U_1\Sigma_1 V_1^T$$

donde,

$$U_1 = \begin{bmatrix} u_1 & & u_r \end{bmatrix}, \quad V_1^T = \begin{bmatrix} v_1 \\ \\ v_r \end{bmatrix}^T \quad y \quad \Sigma_1 = diag(\sigma_1 ... \sigma_r)$$

Capítulo 2

Algoritmos numéricos en la teoría de control.

En este capítulo, se pone atención a los algoritmos numéricos aplicados a la teoría de control. Se delimita el enfoque polinomial de la teoría de control como nuestro campo de trabajo. Se hablará de la importancia que tiene desarrollar algoritmos numéricos confiables para estudiar problemas de control desde el enfoque polinomial y se presenta un breve estado del arte sobre los nuevos algoritmos confiables que se han estado desarrollando para el enfoque polinomial. Finalmente se expondrán algunos de los métodos polinomiales avanzados que son base de estos nuevos algoritmos.

En la segunda parte de este capítulo se presentan dos algoritmos representativos de esta nueva clase de algoritmos confiables desarrollados para el enfoque polinomial. La explicación de estos algoritmos permitirá ejemplificar el uso concreto de algunas de las técnicas polinomiales avanzadas descritas en la primera sección.

Los problemas resueltos por este par de algoritmos son el de la obtención del determinante y el de la obtención del rango de una matriz polinomial. Como se verá más adelante, la información que entrega el determinante y el rango de un matriz polinomial es de gran importancia para la solución de problemas de control, por lo tanto, es importante presentar los algoritmos confiables que permiten obtenerla.

En esta parte del capítulo también es importante observar la forma en la que se tratará, numéricamente, un problema que implica el manejo de matrices cuyos elementos nos son constantes numéricas sino polinomios en la variable compleja s.

2.1 Los algoritmos numéricos y el enfoque polinomial.

Una vez presentado, en el capítulo anterior, un panorama general de los problemas numéricos, las herramientas matemáticas con que pueden analizarse y los algoritmos numéricos como una forma de resolverlos, se pondrá atención en la forma en que los algoritmos numéricos se han aplicado para resolver algunos problemas de la teoría de control.

2.1.1 El enfoque polinomial de la teoría de control.

A lo largo de los años, la teoría de control se ha analizado desde diferentes enfoques, por ejemplo, el enfoque clásico, que se basa en la descripción de un sistema lineal monovariable mediante una función de transferencia, el enfoque moderno, basado en la representación de sistemas mediante las variables de estado y el enfoque polinomial, sobre el cual gira el tema de este trabajo.

El enfoque polinomial (o algebraico) de los sistemas lineales de control fue iniciado por Kučera en los años 70. Inicialmente desarrollado para tratar problemas de control de sistemas discretos invariantes en el tiempo [19], el enfoque polinomial ha dado pie a muchas investigaciones. Actualmente se ha demostrado que el enfoque polinomial puede extenderse a los sistemas continuos [33], a los sistemas con parámetros desconocidos [17] y a los sistemas no lineales [18, 1] pudiendo resolver muchos de los problemas que el análisis y diseño de control de estos sistemas presenta.

El enfoque polinomial reformula, dentro del álgebra polinomial, muchas de las ideas del control en el campo de las variables de estado o en el campo de la frecuencia. La idea básica de este enfoque consiste en establecer una equivalencia entre resolver un problema de control y una ecuación polinomial asociada al mismo.

La presencia de polinomios y matrices polinomiales dentro de la teoría de control es evidente. Por ejemplo, las matrices polinomiales describen sistemas lineales multivariables [15] y ayudan en el análisis de sistemas singulares [20]. También se puede utilizar el enfoque polinomial para analizar el problema de la asignación de polos y la estabilización de sistemas por retroalimentación [19]. Problemas de control óptimo cuadrático [14] y control robusto [17], entre otros, también pueden estudiarse desde el enfoque polinomial. Es pues evidente la importancia de este enfoque.

Por lo que respecta a los algoritmos numéricos, en campo del control automático, son varios los investigadores que actualmente desarrollan e investigan métodos y algoritmos numéricos y computacionales enfocados al estudio de las propiedades de los sistemas así como a la solución de los problemas de control. Desde principios de los 50, se han desarrollado una gran cantidad de algoritmos para la solución de problemas de álgebra lineal, base de la representación de sistemas en variables de estado. También son actualmente conocidos muchos paquetes computacionales que resuelven problemas de control desde el enfoque clásico, obteniendo con gran exactitud y rapidez, diagramas de Bode, Nyquist, lugares de raíces, etc.

Sin embargo, como hemos dicho, el enfoque polinomial se basa en la solución de los problemas de control por medio de la solución de ecuaciones de matrices polinomiales asociadas a los mismos y los métodos numéricos mencionados en el párrafo anterior no permiten la manipulación de matrices cuyos elementos son polinomios y no números. De ahí la importancia de desarrollar nuevos algoritmos y herramientas computacionales para este nuevo enfoque de la teoría de control.

2.1.2 Algoritmos numéricos para el enfoque polinomial.

Tradicionalmente, los investigadores se han preocupado más por el planteamiento de ecuaciones de matrices polinomiales que representen problemas de control que por la solución de las mismas. Estas ecuaciones se han venido resolviendo con operaciones polinomiales básicas (como la división euclidiana) y operaciones elementales en el anillo de los polinomios o las funciones racionales. Sin embargo, los algoritmos numéricos basados en estas operaciones básicas, son inexactos y peor aún, generalmente son numéricamente inestables.

Hace unos años, con la introducción de los lenguajes de matemáticas simbólicas, el problema de generar herramientas computacionales para trabajar con matrices cuyos elementos no fueran constantes parecía resolverse. Incluso, el trabajar con algoritmos simbólicos basados en estos lengujes mejoran la exactitud del resultado obtenido. Al evaluar los números no enteros hasta el final del algoritmo y en una sola operación se evita la propagación de errores de aproximación y se consigue un resultado más exacto.

Pese a estas ventajas, se ha demostrado que los algoritmos simbólicos son ineficientes en el sentido de tiempo de ejecución y recursos de memoria utilizados para obtener una solución. De este modo, la exactitud y estabilidad numérica obtenida, en ocasiones puede pagarse caro y más aún cuando el problema puede resolverse numéricamente. Así pues, el desarrollo y aplicación de algoritmos numéricos estables más avanzados, que sustituyan los algoritmos polinomiales básicos y que logren una exactitud numérica igual a la obtenida por los algoritmos simbólicos, pero sin sacrificar memoria y tiempo de ejecución, se presenta actualmente como un campo muy grande para la investigación.

El enfoque polinomial mantiene estrechas similitudes con el enfoque del espacio de estado, sin embargo, el mayor desarrollo de algoritmos de solución para sistemas vistos desde el espacio de estado, le ha dado mayor reputación a este segundo. Esta situación ha provocado el desarrollo de algoritmos numéricos confiables para transformar una ecuación polinomial en un sistema representado en espacio de estado. Este sistema es analizado o solucionado con algoritmos del álgebra numérica y después se regresa la solución, por medio de otro algoritmo, de nuevo al campo de las matrices polinomiales. Este tipo de algoritmos se ha desarrollado en los últimos años [16], sin embargo, los algoritmos confiables que actuan directamente sobre las matrices polinomiales sin recurrir al espacio de estado son los que trataremos en este trabajo.

Varias son ya las investigaciones y los resultados obtenidos en este campo [8, 13], estos resultados así como los investigadores que han trabajado sobre ellos han contribuido al desarrollo de "The Polynomial Toolbox" [22]. Este es un conjunto de instrucciones desarrollado para trabajar en MATLAB 5.x específicamente con matrices y operaciones polinomiales. La versión 2.0 del Polynomial Toolbox cuenta con más de 220 instrucciones y con ellas es capaz de resolver todo tipo de ecuaciones polinomiales, de manipular matrices polinomiales y obtener una gran cantidad de informaciones de ellas como, por ejemplo, el rango, el determinante, los valores propios, etc. El Polynomial Toolbox cuenta también con algunas macros que resuelven problemas concretos de control como, por ejemplo, la obtención de factorizaciones coprimas de alguna matriz de transferencia, la solución de problemas LQG y asignación de polos, etc.

2.1.3 Métodos polinomiales y algoritmos confiables.

A continuación se presentan algunas de las técnicas polinomiales avanzadas que ayudan en el desarrollo de algoritmos numéricos confiables para matrices polinomiales. De igual forma que con las herramientas básicas del álgebra numérica, es importante presentar estas técnicas no porque sean directamente utilizadas sino porque son parte del estado del arte y ayudarán a entender como es que se realizan algoritmos numéricos con matrices cuyas entradas no son números sino polinomios. Algunas de estas operaciones y métodos polinomiales avanzados que se consideran a continuación son los siguientes:

- Matrices de Sylvester [8].
  Interpolación [8].
- Interpolación con transformada de Fourier [13].

Para explicar estos métodos, los usaremos para encontrar la solución de la siguiente ecuación polinomial,

$$A(s)X(s) = B(s) \tag{2.1}$$

donde $A(s) \in \Re^{m\times n}[s]$, $B(s) \in \Re^{m\times p}[s]$ y $X(s) \in \Re^{n\times p}[s]$. Las matrices $A(s)$ y $B(s)$ son conocidas y la incógnita es $X(s)$. Definamos como d_A el grado del polinomio de mayor orden de $A(s)$, lo mismo con d_B y d_X para las matrices $B(s)$ y $X(s)$ respectivamente. Además, supongamos que $d_B = d_A + d_X$

Matrices de Sylvester.

Las matrices de Sylvester es una técnica que reduce un problema con matrices polinomiales al problema de solucionar un sistema de ecuaciones numéricas. Esta técnica es muy útil en la solución de ecuaciones polinomiales como lo vemos a continuación.

Una forma de resolver la ecuación (2.1) consiste en expresar las matrices de la siguiente forma:

$$\begin{aligned} A(s) &= A_0 + sA_1 + s^2A_2 + ... + s^{d_A}A_{d_A} \\ B(s) &= B_0 + sB_1 + s^2B_2 + ... + s^{d_B}B_{d_B} \\ X(s) &= X_0 + sX_1 + s^2X_2 + ... + s^{d_X}X_{d_X} \end{aligned}$$

Con esta descomposición e igualando las potencias de s podemos obtener un sistema de ecuaciones lineales equivalente, es decir, un sistema que involucra únicamente las matrices constantes $A_0, ..., A_{d_A}, B_0, ..., B_{d_B}$ y cuya solución permite encontrar las matrices $X_0, ..., X_{d_X}$ que forman la matriz buscada $X(s)$. Este sistema lineal equivalente es:

$$\bar{A}\bar{X} = \bar{B}, \quad \begin{bmatrix} A_0 & & 0 \\ A_1 & A_0 & \\ & A_1 & \\ A_{d_A} & & A_0 \\ & A_{d_A} & A_1 \\ & & \\ 0 & & A_{d_A} \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ \\ X_{d_X} \end{bmatrix} = \begin{bmatrix} B_0 \\ B_1 \\ \\ B_{d_B} \end{bmatrix}$$

donde la matriz $\bar{A} \in \Re^{(d_A+d_X+1)m\times(d_X+1)n}$ se conoce como la matriz de Sylvester.

Ejemplo 2.1 *Resolvamos la ecuación polinomial*

$$\begin{bmatrix} 1 & s & s^2 \\ 0 & s+1 & s^2+3 \end{bmatrix} X(s) = \begin{bmatrix} s^2+s+1 & s^2+s \\ s^2+s+4 & s^2+3 \end{bmatrix}$$

Primeramente expresemos las matrices en la forma propuesta. Suponiendo que $d_B = d_A + d_X$, *entonces* $d_X = 0$, *pero esto evidentemente no puede ser, por tanto, supongamos que* $d_X = 1$, *y que entonces* $d_B = d_A + d_X = 3$, *es decir,* $B(s)$ *debe tener una matriz de coeficientes igual a cero.*

$$A(s) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 3 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} s + \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} s^2$$

$$B(s) = \begin{bmatrix} 1 & 0 \\ 4 & 3 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} s + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} s^2 + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} s^3$$

$$X(s) = \begin{bmatrix} x_{01} & x_{02} \\ x_{03} & x_{04} \\ x_{05} & x_{06} \end{bmatrix} + \begin{bmatrix} x_{11} & x_{12} \\ x_{13} & x_{14} \\ x_{15} & x_{16} \end{bmatrix} s$$

De esta forma, el sistema de ecuaciones lineal resultante es:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 3 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 3 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{01} & x_{02} \\ x_{03} & x_{04} \\ x_{05} & x_{06} \\ x_{11} & x_{12} \\ x_{13} & x_{14} \\ x_{15} & x_{16} \end{bmatrix} = \begin{bmatrix} x_{01} & x_{02} \\ x_{03}+3x_{05} & x_{04}+3x_{06} \\ x_{03}+x_{11} & x_{04}+x_{12} \\ x_{03}+x_{13}+3x_{15} & x_{04}+x_{14}+3x_{16} \\ x_{05}+x_{13} & x_{06}+x_{14} \\ x_{05}+x_{13} & x_{06}+x_{14} \\ x_{15} & x_{16} \\ x_{15} & x_{16} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 4 & 3 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Resolviendo este sistema encontramos que:

$$\bar{X} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Por lo tanto,

$$X(s) = \begin{bmatrix} 1 & s \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Comprobación:

$$\begin{bmatrix} 1 & s & s^2 \\ 0 & s+1 & s^2+3 \end{bmatrix} \begin{bmatrix} 1 & s \\ 1 & 0 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1+s+s^2 & s+s^2 \\ 4+s+s^2 & s^2+3 \end{bmatrix}$$

El problema que se presenta en este tipo de algoritmo es que no siempre será fácil estimar d_X y por lo tanto, no siempre será facil saber si hay matrices de coeficientes iguales a cero como sucedió en este ejemplo. Esta situación, si es resuelta por medio de la prueba y error, puede presentar un gran problema en la eficiencia de los algoritmos que utilicen este método. Entonces, para lograr un algoritmo confiable que resuelva, en este caso, la ecuación (2.1) se debe buscar una forma confiable de estimar, primeramente, el valor de d_X. Salvado el obstáculo de la determinación del grado d_X, las matrices de Sylvester representan un método confiable y eficiente.

Interpolación.

Ahora supongamos que las matrices $A(s) \in \Re^{m\times n}[s]$ y $B(s) \in \Re^{m\times p}[s]$ solo se conocen en algunos puntos s_i del plano complejo. Es decir, $A(s)$ y $B(s)$ se conocen únicamente por algunas matrices constantes $A(s_i)$ y $B(s_i)$ con $i = 0, 1, ..., d$. Ahora, la forma en que se puede resolver la ecuación polinomial es por medio de la interpolación.

La ecuación polinomial también puede escribirse como:

$$\begin{bmatrix} A(s) & sA(s) & \cdots & s^{d_X}A(s) \end{bmatrix} \bar{X} = B(s)$$

de modo que usando las matrices constantes que conocemos podemos plantear el sistema de ecuaciones como:

$$\begin{bmatrix} A(s_d) & s_d A(s_d) & & s_d^{d_x} A(s_d) \\ & & & \\ & & & \\ A(s_1) & s_1 A(s_1) & & s_1^{d_x} A(s_1) \\ A(s_0) & s_0 A(s_0) & & s_0^{d_x} A(s_0) \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ \\ \\ X_{d_x} \end{bmatrix} = \begin{bmatrix} B(s_d) \\ \\ \\ B(s_1) \\ B(s_0) \end{bmatrix}$$

El problema con este método es también, como con las matrices de Sylvester, la estimación del grado d_x y por lo tanto del número (d) de puntos de interpolación s_i necesarios para obtener un buen resultado.

El algoritmo de interpolación de una forma más general se plantea como sigue:

Supongamos una función f que se aplica a alguna matriz polinomial

$$A(s) = A_0 + A_1 s + A_2 s^2 + ... + A_{d_A} s^{d_A}$$

El resultado de esta operación es, en general otra matriz polinomial

$$Y(s) = f(A(s)) = Y_0 + Y_1 s + ... + Y_{d_Y} s^{d_Y}$$

Para encontrar numéricamente esta matriz $Y(s)$, y una vez salvado el obstáculo de estimar el grado d_Y, se puede utilizar el siguiente algoritmo:

Algoritmo 2.1 *(Algoritmo de Interpolación)*

1) Primeramente se evalua la matriz $A(s)$ en d_Y+1 puntos escogidos de alguna forma. Esta evaluación nos dará un conjunto de $d_Y + 1$ matrices constantes $A'(s_i)$ donde s_i con $i = 0, 1, 2, ..., d_Y$ son los puntos escogidos.

2) Se aplica la función f a las $d_Y + 1$ matrices constantes obteniendo otro conjunto de matrices constantes $Y'(s_i)$ donde s_i con $i = 0, 1, 2, ..., d_Y$ son los puntos escogidos.

3) Finalmente, se plantea el siguiente sistema de ecuaciones y se resuelve para $Y_0, ..., Y_{d_Y}$

$$\begin{aligned} Y'(s_0) &= Y_0 + Y_1 s_0 + ... + Y_{d_Y} s_0^{d_Y} \\ Y'(s_1) &= Y_0 + Y_1 s_1 + ... + Y_{d_Y} s_1^{d_Y} \end{aligned}$$

$$Y'(s_{d_Y}) = Y_0 + Y_1 s_{d_Y} + ... + Y_{d_Y} s_{d_Y}^{d_Y}$$

que en forma matricial se puede escribir como:

$$\begin{bmatrix} Y_0 & Y_1 & & Y_{d_Y} \end{bmatrix} V = \begin{bmatrix} Y'(s_0) & Y'(s_1) & & Y'(s_{d_Y}) \end{bmatrix}$$

donde V se conoce como matriz de Vandermonde y está dada por:

$$V = \begin{bmatrix} 1 & 1 & & 1 \\ s_0 & s_1 & & s_{d_Y} \\ & & & \\ \vdots & \vdots & & \vdots \\ s_0^{d_Y} & s_1^{d_Y} & & s_{d_Y}^{d_Y} \end{bmatrix}$$

Ejemplo 2.2 *Resolvamos el ejemplo 2.1 con este algoritmo.*

$$\begin{bmatrix} 1 & s & s^2 \\ 0 & s+1 & s^2+3 \end{bmatrix} X(s) = \begin{bmatrix} s^2+s+1 & s^2+s \\ s^2+s+4 & s^2+3 \end{bmatrix}$$

Sabemos, por el ejemplo 2.1, que el grado de $X(s)$ es 1, de modo que necesitamos evaluar las matrices $A(s)$ y $B(s)$ de la ecuación en 2 puntos distintos. Tomemos arbitrariamente los puntos $s_0 = 0$ y $s_1 = 1$, entonces,

$$A(s_0) = A(0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 3 \end{bmatrix} \qquad B(s_0) = B(0) = \begin{bmatrix} 1 & 0 \\ 4 & 3 \end{bmatrix}$$

$$A(s_1) = A(1) = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 4 \end{bmatrix} \qquad B(s_1) = B(1) = \begin{bmatrix} 3 & 2 \\ 6 & 4 \end{bmatrix}$$

Ahora, obtengamos una solución particular de las ecuaciones equivalentes para cada punto.
Para el punto $s_0 = 0$, tenemos,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 3 \end{bmatrix} X_0' = \begin{bmatrix} 1 & 0 \\ 4 & 3 \end{bmatrix}$$

cuya solución puede ser,

$$X_0' = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Para el punto $s_1 = 1$, tenemos,

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 2 & 4 \end{bmatrix} X_1' = \begin{bmatrix} 3 & 2 \\ 6 & 4 \end{bmatrix}$$

cuya solución particular puede ser,

$$X_1' = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Finalmente, el sistema de ecuaciones lineales a resolver es:

$$\begin{bmatrix} X_0 & X_1 \end{bmatrix} V = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

donde V es la matriz no singular

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

La solución de este sistema de ecuaciones es:

$$X_0 = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad X_1 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

de modo que,

$$X(s) = X_0 + X_1 s = \begin{bmatrix} 1 & s \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Del ejemplo anterior, podemos ver que los d_Y+1 puntos pueden ser escogidos arbitrariamente siempre y cuando la matriz V sea no singular. Sin embargo, está demostrado [13] que la arbitrariedad de esta selección puede ocasionar que el problema de la solución del sistema de ecuaciones resultante sea mal condicionado (ill conditioned). Asi pues, una mala aproximación numérica en la evaluación de las matrices constantes $Y'(s_i)$ puede resultar en matrices $Y_0, ..., Y_{d_Y}$ erróneas.

Para resolver este problema, se ha demostrado [13] que una buena selección de puntos pueden ser las d_Y+1 raíces complejas de la unidad, es decir,

$$s_k = e^{-j\frac{2\pi k}{d_Y+1}}$$

estos puntos son llamados puntos de Fourier. La idea de evaluar el algoritmo de interpolación en los puntos de Fourier da pie a la siguiente modificación del algoritmo de interpolación.

Interpolación con transformada de Fourier.

Primeramente se empieza con la definición y algunos resultados de la transformada de Fourier.

Definición 2.1 *Sea un vector de números complejos*

$$p = \begin{bmatrix} p_0 & p_1 & & p_n \end{bmatrix}$$

la DFT (discrete Fourier transform) de $m+1$ puntos del vector p está dada por el vector

$$q = \begin{bmatrix} q_0 & q_1 & & q_m \end{bmatrix}$$

conocido como la imagen de p donde

$$q_k = \sum_{i=0}^{m} p_i e^{-j\frac{2\pi k}{m+1}i}$$

La DFT inversa del vector q nos entrega el vector p original donde:

$$p_i = \frac{1}{m+1}\sum_{k=0}^{m} q_k e^{j\frac{2\pi i}{m+1}k}$$

La extensión de la definición 2.2 a matrices es análoga

Ejemplo 2.3 *Obtengamos la DFT de 4 puntos del siguiente vector:*

$$x = \begin{bmatrix} 1 & 1 & -1 \end{bmatrix}$$

Para q_0:

$$q_0 = \sum_{i=0}^{3} x_i = 1+1-1+0 = 1$$

Para q_1:

$$q_1 = \sum_{i=0}^{3} x_i e^{-j\frac{\pi}{2}i} = 1 + 1e^{-j\frac{\pi}{2}} - 1e^{-j\pi} + 0 = 2 - j$$

Para q_2:

$$q_2 = \sum_{i=0}^{3} x_i e^{-j\pi i} = 1 + 1e^{-j\pi} - 1e^{-2j\pi} + 0 = -1$$

Por último, para q_3:

$$q_3 = \sum_{i=0}^{3} x_i e^{-j\frac{3\pi}{2}i} = 1 + 1e^{-j\frac{3\pi}{2}} - 1e^{-j3\pi} + 0 = 2 + j$$

Por lo tanto:

$$y = \begin{bmatrix} 1 & 2-j & -1 & 2+j \end{bmatrix}$$

Algunos resultados útiles sobre la DFT se resumen en los siguientes teoremas [13]:

Teorema 2.1 *La DFT inversa puede expresarse como $\frac{1}{m+1}$ veces el complejo conjugado de la DFT aplicada al complejo conjugado de la imagen q, es decir:*

$$p_i = \frac{1}{m+1}\left(\sum_{k=0}^{m} q_k^* e^{-j\frac{2\pi i}{m+1}k}\right)$$

Teorema 2.2 *En el caso particular en que p tenga solo elementos reales, la imagen q cumple con la siguiente condición:*

$$q_{m+1-k} = q_k^*$$

Otra forma de escribir la definición de DFT es la siguiente:

$$q_k = \sum_{i=0}^{m} p_i s_k^i$$

donde

$$s_k = e^{-j\frac{2\pi k}{m+1}}$$

Es pues evidente que q_k es la evaluación del polinomio o matriz

$$p(s) = p_0 + p_1 s + p_2 s^2 + ... + p_n s^n$$

en los $m+1$ valores s_k.

Estos resultados también se aplican, de manera análoga en el caso de matrices.

Asi pues, si hacemos $m = d_Y$, el primer paso del algoritmo general de interpolación (algoritmo 2.1) se reduciría a encontrar la DFT del conjunto de matrices $A_0, A_1, ..., A_{d_A}$.

Por otro lado, como es de esperarse, dado un conjunto de valores (o matrices) $z_0, ..., z_{d_Y}$, la DFT inversa, nos permitiría encontrar el conjunto de valores (o matrices) constantes que forman el polinomio (o matriz):

$$x(s) = x_0 + x_1 s + x_2 s^2 + ... + x_{d_Y} s^{d_Y}$$

cuya evaluación en los $d_Y + 1$ valores s_k es igual a z_i con $i = 0, 1, ..., d_Y$. Así pues, el tercer paso del algoritmo general de interpolación se reduce a encontrar una DFT inversa.

Resumiendo, el algoritmo de interpolación con DFT puede ser escrito como sigue:

Algoritmo 2.2 *(Algoritmo de interpolación con DFT).*

1) Primeramente se evalua la matriz $A(s)$ en $d_Y + 1$ puntos de Fourier. Para hacer esta evaluación se obtiene la DFT del conjunto de matrices constantes $A_0, ..., A_{d_A}$. Esto nos dará un conjunto de $d_Y + 1$ matrices constantes $A'(s_i)$ donde s_i con $i = 0, 1, 2, ..., d_Y$ son los puntos de Fourier.

2) Se aplica la función f a las $d_Y + 1$ matrices constantes obteniendo otro conjunto de matrices constantes $Y'(s_i)$ donde s_i con $i = 0, 1, 2, ..., d_Y$ son los puntos de Fourier.

3) Finalmente, se obtiene la DFT inversa del conjunto de matrices $Y'(s_i)$ dando como resultado el conjunto de matrices $Y_0, ..., Y_{d_Y}$ que forman la matriz resultado

$$Y(s) = f(A(s)) = Y_0 + Y_1 s + ... + Y_{d_Y} s^{d_Y}$$

El problema con este método sigue siendo la estimación del grado d_Y de la solución, sin embargo, el problema de que las aproximaciones numéricas ocasionen resultados erróneos, por que el sistema de ecuaciones a resolver resulte mal condicionado, se puede evitar.

Para ejemplificar este método resolvamos también el ejemplo 2.1.

Ejemplo 2.4 *Tomemos el mismo problema,*

$$\begin{bmatrix} 1 & s & s^2 \\ 0 & s+1 & s^2+3 \end{bmatrix} X(s) = \begin{bmatrix} s^2+s+1 & s^2+s \\ s^2+s+4 & s^2+3 \end{bmatrix}$$

Considerando de nuevo que $d_X = 1$, obtengamos la DFT de dos puntos de las matrices

$$A(s) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 3 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} s + \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} s^2$$

$$B(s) = \begin{bmatrix} 1 & 0 \\ 4 & 3 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} s + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} s^2$$

Para $A(s)$ tenemos,

$$DFT\{A_0, A_1, A_2\} = \left\{ \begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 0 & 3 \end{bmatrix} \right\}$$

Para $B(s)$ tenemos,

$$DFT\{B_0, B_1, B_2\} = \left\{ \begin{bmatrix} 2 & 1 \\ 5 & 3 \end{bmatrix} \cdot \begin{bmatrix} 0 & -1 \\ 3 & 3 \end{bmatrix} \right\}$$

Ahora, resolviendo la ecuación para cada par de matrices, tenemos

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 3 \end{bmatrix} X(s_0) = \begin{bmatrix} 2 & 1 \\ 5 & 3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 & 0 \\ 0 & 0 & 3 \end{bmatrix} X(s_1) = \begin{bmatrix} 0 & -1 \\ 3 & 3 \end{bmatrix}$$

de donde,

$$X(s_0) = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \quad X(s_1) = \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Ahora, obteniendo la DFT inversa de $X(s_0)$ y $X(s_1)$ tenemos,

$$DFT^{-1}\left\{\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}\right\} = \left\{\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}\right\} = \{X_0, X_1\}$$

de modo que,

$$X(s) = X_0 + X_1 s = \begin{bmatrix} 1 & s \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

2.2 Algoritmos representativos.

Esta segunda sección está dividida en dos partes, en la primera parte se presenta el algoritmo que obtiene el determinante de una matriz polinomial y en la segunda parte un algoritmo para obtener el rango de una matriz polinomial.

Ambos algoritmos ya están desarrollados, el de la obtención del determinante en [9] y el de la obtención del rango en [8]. El objetivo de presentarlos en este trabajo es para ejemplificar el uso de las técnicas polinomiales avanzadas para resolver este par de problemas, los cuales, como se explica más adelante, tienen una gran importancia para el análisis y diseño de sistemas de control.

2.2.1 El Determinante de una matriz polinomial.

Dentro del enfoque polinomial de la teoría de control, la obtención del determinante de una matriz polinomial cuadrada así como el grado del mismo representa un problema típico de mucha importancia. Por ejemplo, a partir del determinante de una matriz polinomial se puede determinar el orden de una realización en espacio de estado de algún sistema lineal, conocer el número de ceros y polos del sistema y determinar la estructura al infinito de matrices polinomiales entre otras cosas [15, 2].

Preliminares.

Se presentan, primeramente, algunas definiciones y resultados básicos sobre el determinante de una matriz polinomial, posteriormente, se presenta el algoritmo desarrollado para obtenerlo.

Sea $A(s) \in \Re^{n\times n}(s)$ una matriz cuadrada polinomial, se dice que $A(s)$ es singular si $\det A(s) = 0$ para toda s, y no singular cuando $\det A(s) \neq 0$ para toda s excepto para los valores de s donde $A(s)$ pierde rango. Estos valores son conocidos, como veremos más adelente, como ceros de $A(s)$. El determinante $\det A(s)$, de una matriz polinomial es, en general, un polinomio en s de la forma:

$$d(s) = d_0 + d_1 s + ... + d_\partial s^\partial$$

La matriz $A(s)$ también puede ser escrita como:

$$A(s) = A_0 + A_1 s + ... + A_a s^a$$

donde las matrices A_i son conocidas como matrices de coeficientes. El número a indica el mayor orden de los elementos (polinomios) de la matriz $A(s)$. Las matrices A_i o matrices de coeficientes son las matrices con las que los algoritmos trabajarán.

Ahora bien, para obtener el determinante de una matriz polinomial, se puede realizar la expansión en cofactores sobre cualquier fila o columna de la misma y realizar las multiplicaciones entre polinomios necesarias. Sin embargo, para obtener el determinante numéricamente, es decir, utilizando solo las matrices de coeficientes de $A(s)$ es necesario estimar de alguna forma el grado ∂ del polinomio determinante para después (por algún método basado en interpolación) encontrar el polinomio determinante. Es la estimación de ∂ el problema principal que se presenta en el diseño de métodos numéricos confiables para obtener determinantes.

Resultados principales.

A continuación se presenta el teorema básico enunciado en [9] con respecto a la obtención del determinante. Este teorema nos proporciona un algoritmo confiable para obtener el grado ∂ del determinante, una vez salvado este problema, la obtención del polinomio determinante se puede lograr con alguno de los métodos de interpolación mostrados en la sección anterior.

Teorema 2.3 *Sea $A(s)$ una matriz polinomial cuadrada de nxn:*

$$A(s) = A_0 + A_1 s + ... + A_a s^a$$

y sea T_i la matriz de Toeplitz siguiente:

$$T_i = \begin{bmatrix} A_a & A_{a-1} & & A_{-i+1} & A_{-i} \\ 0 & A_a & & A_{-i+2} & A_{-i+1} \\ & & & & \\ & & & & \\ 0 & 0 & & A_a & A_{a-1} \\ 0 & 0 & & 0 & A_a \end{bmatrix}$$

donde A_p son las matrices de coeficientes de $A(s)$. Asumamos que $A_p = 0$ para $p < 0$, es decir, $A(s)$ no tiene potencias de s negativas.

Ahora, definiendo:

$$r_i = rankT_i - rankT_{i-1}$$

y tomando

$$k = \min\{i : r_i = n, i \geq 0\}$$

entonces, el grado ∂ del determinante de $A(s)$ está dado por:

$$\partial = rankT_k - n(k+1)$$

La prueba de este resultado se puede revisar en [9].

A continuación se presentan algunos ejemplos sobre la aplicación de este algoritmo.

Ejemplo 2.5 *Obtengamos el determinante de la siguiente matriz:*

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

$$= \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} s^3 + \begin{bmatrix} 2 & 2 & 1 \\ -2 & -1 & -1 \\ 0 & 0 & 0 \end{bmatrix} s^2 + \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} s + \begin{bmatrix} 1 & 0 & 0 \\ -3 & -1 & -1 \\ 2 & 2 & 1 \end{bmatrix}$$

Primeramente obtenemos las matrices T_{-1} y T_0,

$$T_{-1} = \begin{bmatrix} 0 & 1 & 0 & 2 & 2 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -2 & -1 & -1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & -2 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$T_0 = \begin{bmatrix} 0 & 1 & 0 & 2 & 2 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & -2 & -1 & -1 & 1 & 1 & 1 & -3 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 2 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -2 & -1 & -1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & -2 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

de esta forma, $r_0 = rankT_0 - rankT_{-1} = 6 - 4 = 2$. *Continuando con el método, podemos demostrar que* $r_4 = 3 = n$ *asi que, según el teorema 2.1,*

$$\partial = rankT_4 - n(4+1) = 15 - 15 = 0$$

Este resultado significa entonces que la matriz $A(s)$ *es una matriz unimodular y su determinante es igual a una constante distinta de cero que puede ser obtenida de la evaluación de* $A(s)$ *en cualquier punto.*

$$d(s) = \det A(0) = \det \begin{bmatrix} 1 & 0 & 0 \\ -3 & -1 & -1 \\ 2 & 2 & 1 \end{bmatrix} = 1$$

Es importante mencionar que la evaluación del rango de las matrices constantes T_i *se realiza con los algoritmos ya programados en MATLAB.*

Ejemplo 2.6 *Obtengamos ahora el determinante de la matriz:*

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

Obtengamos primeramente T_{-1} *y* T_0,

$$T_{-1} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_0 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 3 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

de esta forma, $r_0 = rankT_0 - rankT_{-1} = 2 = n$. *Finalmente:*

$$\partial = rankT_0 - 2 = 3$$

Ahora, para obtener el polinomio determinante de orden 3,

$$d(s) = d_0 + d_1 s + d_2 s^2 + d_3 s^3$$

se puede usar interpolación (tomando 4 diferentes puntos). Tomando los puntos $s = 0, 1, 2, 3$, *el sistema de ecuaciones resultante es el siguiente:*

$$\begin{aligned} \det A(0) &= d_0 \\ \det A(1) &= d_0 + d_1 + d_2 + d_3 \\ \det A(2) &= d_0 + 2d_1 + 4d_2 + 8d_3 \\ \det A(3) &= d_0 + 3d_1 + 9d_2 + 27d_3 \end{aligned}$$

es decir,

$$\begin{aligned} 1 &= d_0 \\ 9 &= d_0 + d_1 + d_2 + d_3 \\ 29 &= d_0 + 2d_1 + 4d_2 + 8d_3 \\ 67 &= d_0 + 3d_1 + 9d_2 + 27d_3 \end{aligned}$$

La solución de este sistema es: $d_0 = 1, d_1 = 4, d_2 = 3, d_3 = 1$. *Finalmente, el determinante de* $A(s)$ *es:*

$$d(s) = 1 + 4s + 3s^2 + s^3$$

Ahora resolvamos este mismo problema con el algoritmo de interpolación con DFT (Discrete Fourier Transform).

$$A(s) = A_0 + A_1 s + A_2 s^2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix} s + \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} s^2$$

obtengamos la DFT del conjunto de matrices $Q = \{A_0, A_1, A_2\}$ *para cuatro valores* $(\partial + 1)$. *El resultado de esta operación es el siguiente conjunto de matrices:*

$$R = \left\{ \begin{bmatrix} 2 & 1 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} 1-j & -1 \\ 1 & -3j \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1+j & -1 \\ 1 & 3j \end{bmatrix} \right\}$$

La evaluación del determinante de estas cuatro matrices nos da el siguiente vector de resultados:

$$r' = \begin{bmatrix} 9 & -2-3j & -1 & -2+3j \end{bmatrix}$$

Y finalmente, la DFT inversa de r' *es el siguiente vector:*

$$r = \begin{bmatrix} 1 & 4 & 3 & 1 \end{bmatrix}$$

Por lo tanto,

$$d(s) = 1 + 4s + 3s^2 + s^3$$

2.2.2 El rango de una matriz polinomial.

Determinar el rango de una matriz polinomial suele ser el primer paso en la mayoría de los algoritmos numéricos que se desarrollan para el estudio de propiedades de sistemas vistos desde el enfoque polinomial. De ahí la importancia de contar con algoritmos numéricos que permitan determinar ese rango de una manera confiable y eficiente.

Preliminares.

De igual forma que en la sección anterior, se comienza presentando algunas definiciones y resultados básicos, para después analizar los resultados y algoritmos desarrollados para obtener el rango de matrices polinomiales.

Se sabe que por definición, el rango de una matriz con elementos reales es la dimensión de su imagen, que también corresponde con el número de columnas linealmente independientes que tiene. Una definición análoga se puede aplicar para hablar del rango de matrices polinomiales. Algunas de las definiciones equivalentes de rango para matrices polinomiales son:

1) El mayor orden de todos los menores distintos de cero de la matriz $A(s)$.

2) El número,

$$\max_{\lambda \in C} rank A(\lambda)$$

Otro concepto importante es el concepto de cero de una matriz polinomial. El valor $s = s_o$ es un cero de la matriz $A(s)$ si

$$rank A(s_0) < r$$

donde r es el rango de $A(s)$.

Ejemplo 2.7 *El rango de la matriz polinomial,*

$$A(s) = \begin{bmatrix} s^2+1 & s & s^3+s \\ 2 & 0 & 2s \\ 3 & s+2 & 3s \end{bmatrix}$$

es 2, ya que evidentemente la columna 3 es igual a la columna 1 multiplicada por s, es decir, hay sólo dos columnas que son linealmente independientes (en sentido polinomial).

Comprobemos las definiciones equivalentes que acabamos de mencionar con esta matriz. Obtengamos los menores de $A(s)$,

$$M_{11} = \begin{bmatrix} 0 & 2s \\ s+2 & 3s \end{bmatrix} \quad \det M_{11} = -2s^2 - 4s$$

$$M_{12} = \begin{bmatrix} 2 & 2s \\ 3 & 3s \end{bmatrix} \quad \det M_{12} = 0$$

continuando de esta forma, veremos como el orden máximo de los menores de $A(s)$ distintos de cero es 2.

Ahora, obtengamos el número:

$$\max_{\lambda \in C} rank A(\lambda)$$

Por simple inspección de la matriz $A(s)$, podemos ver que no hay ningun valor de s que haga que el rango de esta sea menor de 2, es decir, $A(s)$ no tiene ceros. Por otro lado, también se puede ver que para cualquier valor de s, la columnas 1 y 3 seguirán siendo dependientes. Por lo tanto, para cualquier valor de λ, $rank A(\lambda) = 2$.

El rango de una matriz polinomial se puede ver, desde la teoría del álgebra lineal, como si se tratara de una matriz de coeficientes constantes es decir, para una matriz $A(s) \in \Re^{mxn}[s]$,

$$rank A(s) = n - \upsilon(A(s))$$

donde $\upsilon(A(s))$ se conoce como la nulidad de $A(s)$ y es la dimensión del kernel de $A(s)$, es decir, el conjunto de vectores $v_i(s)$ linealmente independientes, tales que:

$$A(s)v_i(s) = 0$$

Resultados principales.

A continuación se presentan dos algoritmos desarrollados en [8] para obtener el rango de una matriz polinomial, uno de ellos basado en la técnica de interpolación y otro basado en las matrices de Sylvester.

Algoritmo de Interpolación.

Un primer algoritmo para determinar el rango, se deriva de la técnica de interpolación y del hecho de que una matriz polinomial $A(s) \in \Re^{m\times n}[s]$ puede tener a lo más k ceros, donde

$$k = \min\left(\sum_{i=1}^{m} \deg row_i A, \sum_{j=1}^{n} \deg col_j A\right)$$

La idea básica de este algoritmo es la siguiente: si evaluamos el rango de $A(s)$ en determinados puntos, a lo más haremos k evaluaciones hasta encontrar un valor $s = s_0$ que no sea cero de la matriz y que por lo tanto

$$rankA(s_0) = \max_{\lambda\in C} rankA(\lambda) = rankA(s)$$

Este algoritmo puede modificarse también para utilizar la DFT, de esta forma, los k valores en los que se evaluará el rango de $A(s)$ serán los puntos de Fourier.

Ejemplo 2.8 *Obtengamos el rango de la matriz polinomial*

$$A(s) = \begin{bmatrix} s & s & 0 & s^2 \\ 2s & 2s & 2 & 2 \\ 1 & s & 0 & s \end{bmatrix}$$

Primero encontremos el número máximo de puntos a evaluar,

$$k = \min\left(\sum_{i=1}^{m} \deg row_i A, \sum_{j=1}^{n} \deg col_j A\right) = \min(4,4) = 4$$

y evaluemos el rango de $A(s)$ en cuatro puntos escogidos arbitrariamente,

$$rankA(0) = rank \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 1 & 0 & 0 & 0 \end{bmatrix} = 2$$

$$rankA(1) = rank \begin{bmatrix} 1 & 1 & 0 & 1 \\ 2 & 2 & 2 & 2 \\ 1 & 1 & 0 & 1 \end{bmatrix} = 1$$

$$rankA(2) = rank \begin{bmatrix} 2 & 2 & 0 & 4 \\ 4 & 4 & 2 & 2 \\ 1 & 2 & 0 & 2 \end{bmatrix} = 3$$

y claro, como el rango máximo que puede tener $A(s)$ es tres, significa que ese es el resultado, es decir,

$$rankA(s) = 3$$

y por lo tanto, los puntos $s_0 = 0$ y $s_1 = 1$ son ceros de $A(s)$.

Algoritmo con Matrices de Sylvester.

Este algoritmo, más elaborado que el primero, se basa en la idea de encontrar una base para el kernel de $A(s) \in \Re^{mxn}[s]$, es decir, las q columnas de la matriz $K(s) \in \Re^{nxq}[s]$ tal que

$$A(s)K(s) = 0_{mxq}$$

de esta forma, el rango de $A(s)$ estaría dado por

$$rankA(s) = n - q$$

Para resolver este problema numéricamente, se propone una equivalencia entre la ecuación polinomial $A(s)K(s) = 0_{mxq}$ y la ecuación de Sylvester asociada, es decir, considerando

$$\begin{aligned} A(s) &= A_0 + A_1 s + ... + A_{d_a} s^{d_a} \\ K(s) &= K_0 + K_1 s + ... + K_{d_k} s^{d_k} \end{aligned}$$

se forma la ecuación constante

$$\begin{bmatrix} A_0 & & 0 \\ A_1 & A_0 & \\ & A_1 & \\ A_{d_A} & & A_0 \\ & A_{d_A} & A_1 \\ & & \\ 0 & & A_{d_A} \end{bmatrix} \begin{bmatrix} K_0 \\ K_1 \\ \\ K_{d_k} \end{bmatrix} = 0_{mxq} = \bar{A}\bar{K}$$

Así pues, encontrar el espacio nulo de $A(s)$ es también equivalente a encontrar el espacio nulo de $\bar{A}$. El problema de nuevo es estimar el grado d_k para de esta forma determinar el tamaño de la matriz $\bar{A}$.

Para determinar el grado d_k, el siguiente lema y su corolario nos provee de una cota superior e inferior.

Lema 2.1 *Sea la matriz $A(s) \in \Re^{m\times n}[s]$. El grado d_k de la matriz $K(s)$, cuyas columnas son una base del espacio nulo derecho de $A(s)$, es menor o igual que d_R^*, donde,*

$$d_R^* = \sum_{i=1}^{n} \deg col_i A - \min_{i=1,2...,n} \deg col_i A$$

La prueba de este lema se puede revisar en [8].

Corolario 2.1 *Sea la matriz $A(s) \in \Re^{m\times n}[s]$, el grado d_k' de la matriz $K'(s) \in \Re^{r\times m}[s]$, cuyas filas son una base del espacio nulo izquierdo de $A(s)$, es menor o igual que d_L^*, donde,*

$$d_L^* = \sum_{i=1}^{m} \deg row_i A - \min_{i=1,2...,m} \deg row_i A$$

Nota 2.1 *De la teoría del álgebra de matrices podemos comprobar que obtener una base para el kernel izquierdo de una matriz es equivalente a obtener una base para el kernel derecho de la matriz transpuesta. Es decir, sea* $A \in \Re^{m\times n}[s]$, *definimos*

$$A^T(s)_{n\times m}K^*(s)_{m\times r} = 0_{n\times r}$$

donde $K^*(s)$ *está formada por la base del kernel derecho de* $A^T(s)$, *y*

$$K_i(s)_{r\times m}A(s)_{m\times n} = 0_{r\times n}$$

donde $K_i(s)$ *está formada por la base del kernel izquierdo de* $A(s)$, *entonces, podemos demostrar que:*

$$K_i(s) = (K^*(s))^T$$

Por otro lado, debido a la forma de la matriz, las columnas linealmente dependientes de $\bar{A}$ van aumentando en cada grupo de columnas. Esto es, sea q_0 el número de columnas dependientes dentro de las primeras n columnas, q_1 el número de columnas dependientes dentro de las $2n$ primeras columnas y así sucesivamente hasta que q_{d_k} es el número de columnas linealmente dependientes dentro de las $(d_k+1)n$ columnas de $\bar{A}$, entonces se verifica que:

$$0 \leq q_0 \leq q_1 \leq \ldots \leq q_{d_k}$$

De esta forma, el rango de $\bar{A}$, se puede escribir como:

$$rank\bar{A} = (n-q_0) + (n-q_1) + \ldots + (n-q_{d_k})$$

Ahora, considerando que solo sabemos que $d_k \leq d_R^*$, podemos escribir:

$$0 \leq q_0 \leq q_1 \leq \ldots \leq q_{d_R^*} = q_{d_R^*+1} = \ldots = q$$

donde q es la dimensión del espacio nulo de $A(s)$. Finalmente, el rango de $A(s)$ se puede encontrar como:

$$\begin{aligned} rankA(s) &= rank\bar{A}_{d_R^*} - rank\bar{A}_{d_R^*-1} \\ &= (n-q_0) + (n-q_1) + \ldots + (n-q_{d_R^*}) - (n-q_0) - (n-q_1) - \ldots - (n-q_{d_R^*-1}) \\ &= n - q_{d_R^*} = n-q \end{aligned}$$

Ahora, con base en el corolario 2.1 (nota 2.1), también podemos escribir:

$$\begin{aligned} rankA(s) &= rank\bar{A}'_{d_L^*} - rank\bar{A}'_{d_L^*-1} \\ &= (m-r_0) + (m-r_1) + \ldots + (m-r_{d_L^*}) - (m-r_0) - (m-r_1) - \ldots - (m-r_{d_L^*-1}) \\ &= m - r_{d_L^*} = m-r \end{aligned}$$

Veamos un ejemplo para ilustrar el algoritmo.

Ejemplo 2.9 *Obtengamos el rango de la matriz polinomial*

$$A = \begin{bmatrix} s & s & 1 & 1 \\ 2s & 2s & 2 & 2 \\ 1 & s & 0 & s \end{bmatrix}$$

Primeramente resolvamos este problema considerando el espacio nulo derecho

$$d_R^* = \sum_{i=1}^{n} \deg col_i A - \min_{i=1,2\ldots,n} \deg col_i A = (1+1+0+1) - 0 = 3$$

de esta forma, $\bar{A}_{d_R^*}$ y $\bar{A}_{d_R^*-1}$ están dadas por:

$$\bar{A}_{d_R^*} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad rank\bar{A}_{d_R^*} = 10$$

$$\bar{A}_{d_R^*-1} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \quad rank\bar{A}_{d_R^*-1} = 8$$

por lo tanto,

$$rankA(s) = 10 - 8 = 2$$

Ahora resolvamos el problema considerando la dimensión del espacio nulo izquierdo, es decir,

$$d_L^* = \sum_{i=1}^{m} \deg row_i A - \min_{i=1,2\ldots,m} \deg row_i A = (1+1+1) - 1 = 2$$

de esta forma, $\bar{A}'_{d^*_L}$ *y* $\bar{A}'_{d^*_L-1}$ *están dadas por:*

$$\bar{A}'_{d^*_L} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad rank\bar{A}'_{d^*_L} = 6$$

$$\bar{A}'_{d^*_L-1} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 1 & 2 & 0 \\ 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad rank\bar{A}'_{d^*_L-1} = 4$$

Finalmente:

$$rankA(s) = 6 - 4 = 2$$

Capítulo 3

Obtención del Interactor de un sistema lineal.

En este capítulo se presenta el primero de los algoritmos obtenidos en el desarrollo de este trabajo. Este algoritmo obtiene la matriz interactor de un sistema lineal multivariable.

Como se explicará más adelante, el interactor de un sistema lineal multivariable es una matriz polinomial invariante bajo retroalimentación de estado regular, que contiene la información sobre la estructura al infinito del sistema lineal. Dadas estas características, el interactor es utilizado para resolver diversos problemas de control, como por ejemplo, el desacoplamiento de sistemas lineales que se abordará en el siguiente capítulo.

De igual forma, dada una matriz racional cualquiera, sea propia o no, la información al infinito contenida en lo que sería su matriz interactor, puede ser utilizada, como se verá en el capítulo 5, para obtener la estructura al infinito de la matriz o su forma de Smith McMillan al infinito.

Como se verá más adelante, una forma natural de obtener el interactor sería por medio de operaciones elementales las cuales, como se mencionó en el capítulo anterior, son numéricamente inestables. Nuestro algoritmo, en cambio, es un algoritmo confiable el cual evita estas operaciones elementales. Cabe mencionar que existen algunos otros trabajos más recientes que presentan algoritmos para la obtención del interactor, tal es el caso de [28], el cual presenta un algoritmo para obtener una forma de interactor nilpotente de un sistema lineal. Existen, sin embargo, algunas diferencias entre este interactor nilpotente y la definición clásica de interactor, presentada en [34]. Por lo tanto, los problemas no son estrictamente los mismos, y sus métodos de solución son completamente distintos.

Este capítulo se divide de la siguiente manera, primeramente se presenta la definición general del interactor y algunos resultados importantes. En la segunda sección se presenta el algoritmo desarrollado para obtener este interactor.

3.1 El interactor de un sistema lineal multivariable.

Consideremos un sistema lineal multivariable de n estados, p salidas y m entradas descrito, en espacio de estado, por:

$$(A,B,C)\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) \end{cases}$$

donde $x \in \Re^n$, $u \in \Re^m$, $y \in \Re^p$, $A \in \Re^{n\times n}$, $B \in \Re^{n\times m}$ y $C \in \Re^{m\times n}$. Consideremos también que la matriz de transferencia de este sistema es la matriz racional propia $T(s) \in \Re_p^{p\times m}$ dada por:

$$T(s) = C(sI - A)^{-1}B$$

Ahora, definamos la matriz interactor de este sistema lineal. Esta definición asi como gran parte de los resultados que se presentan en este capítulo están basados en el artículo de Falb y Wolovich [34].

Definición 3.1 *Sea (A, B, C) un sistema lineal multivariable con matriz de transferencia $T(s)$ de rango igual a p y sea $\Phi(s)$ una matriz polinomial triangular inferior, única, de $p \times p$ y de la forma:*

$$\Phi(s) = \begin{bmatrix} s^{f_1} & & 0 \\ \phi_{21}(s) & s^{f_2} & \\ & & \\ \phi_{p1}(s) & \phi_{p2}(s) & s^{f_p} \end{bmatrix}$$

donde, para $i > j$, $\phi_{ij}(s) = 0$ o $\phi_{ij}(s)/s^{f_j}$ es divisible por s, tal que:

$$\lim_{s\to\infty} \Phi(s)T(s) = K$$

y donde K es una matriz constante de $p \times m$ y de rango pleno, es decir $rank K = p$.

La matriz $\Phi(s)$ es lo que se conoce como el interactor del sistema (A, B, C) cuya matriz de transferencia es $T(s)$.

La matriz interactor, como se verá más adelante, contiene toda la información de los ceros al infinito del sistema representado por $T(s)$. Además, se puede demostrar que $\Phi(s)$ es invariante bajo retroalimentación de estado, es decir, $\Phi(s)$ no se puede modificar mediante una retroalimentación de estado regular. Estas propiedades hacen que el interactor juegue un papel importante en la solución de muchos problemas de control. Más adelante se verá una de sus aplicaciones en el desacoplamiento de sistemas y también será utilizado indirectamente para obtener la estructura al infinito de una matriz racional en general.

A continuación se presentan algunos resultados sobre el interactor que nos ayudarán a demostrar algunas de sus propiedades y que nos darán idea de como es posible obtenerlo. Se comienza con algunas definiciones generales.

Definición 3.2 *Sea $f(s)$ una función racional. El grado relativo de $f(s)$, denotado por $\deg_p f(s)$, se define como la resta del grado del polinomio denominador menos el grado del polinomio numerador. Una función racional propia es aquella en la que $\deg_p f(s) \geq 0$.*

Definición 3.3 *Sea $A(s)$ una matriz racional. Se dice que $A(s)$ es propia cuando todos sus elementos son funciones racionales propias. Se dice que $A(s)$ es bipropia cuando es una matriz racional propia no singular y además su inversa es también racional propia.*

Si $A(s)$ es polinomial, se dice que es unimodular si es no singular y además su inversa es también polinomial.

Lema 3.1 *Una matriz $A(s)$ es bipropia si y solo si,*

$$\det\left(\lim_{s\to\infty} A(s)\right) = x \in \Re, \quad x \neq 0.$$

o equivalentemente, si y solo si,

$$\lim_{s\to\infty} A(s) = A'$$

donde A' es una matriz constante no singular.

Por otro lado, una matriz $A(s)$ es unimodular si y solo si,

$$\det\left(A(s)\right) = x \in \Re, \quad x \neq 0.$$

Ahora, para poder introducir la forma de Hermite por columnas de una matriz racional propia, la cual, como se verá, tiene una relación directa importante con la matriz interactor, se necesita definir lo que son las operaciones elementales en el anillo[1] de las funciones racionales propias.

Definición 3.4 *Las operaciones elementales por columnas, en el anillo de las funciones racionales propias $\Re_p(s)$, sobre la matriz $T(s)$ se definen como:*

1. *Intercambiar dos columnas de $T(s)$.*
2. *Multiplicar una columna de $T(s)$ por una función racional bipropia.*
3. *Sumar a una columna de $T(s)$ otra columna de $T(s)$ multiplicada por una función racional propia.*

Estas operaciones son una extensión de las operaciones elementales en el anillo de los polinomios $\Re(s)$, las cuales se definen a continuación ya que también son utilizadas para la obtención del interactor.

Definición 3.5 *Las operaciones elementales por columnas, en el anillo de los polinomios $\Re(s)$, sobre la matriz $T(s)$ se definen como:*

1. *Intercambiar dos columnas de $T(s)$.*
2. *Multiplicar una columna de $T(s)$ por una constante distinta de cero.*
3. *Sumar a una columna de $T(s)$ otra columna de $T(s)$ multiplicada por un polinomio.*

De manera análoga se pueden definir las operaciones elementales por filas en $\Re_p(s)$ o en $\Re(s)$.

La forma de Hermite por columnas de una matriz racional propia se define como sigue.

Definición 3.6 *Sea $T(s) \in \Re_p^{m\times m}(s)$ una matriz racional propia de rango pleno y sea $B(s)$ una matriz bipropia tal que:*

$$T(s)B(s) = H(s) = \begin{bmatrix} \frac{1}{s^{f_1}} & & 0 \\ & & \\ & & \\ h_{m1}(s) & & \frac{1}{s^{f_m}} \end{bmatrix} \tag{3.1}$$

es una matriz triangular inferior única, conocida como la forma de Hermite por columnas de $T(s)$ en el anillo de las funciones racionales propias $\Re_p(s)$, donde $h_{ij}(s)$ son funciones racionales propias con un único polo en $s = 0$ y que satisfacen, $h_{ij}(s) = 0$ o $\deg_p h_{ij}(s) < f_i$ para toda $i > j$.

[1] Es importante mencionar que, para no desviarnos de nuestro objetivo, muchos de los conceptos del álgebra lineal que se utilizan en este capítulo, como por ejemplo, el concepto de anillo, unidad en el anillo, etc, no son definidos en este trabajo. Para ver estas definiciones se puede revisar por ejemplo [7, 2, 12].

Se puede demostrar que la matriz bipropia $B(s)$ está dada por:

$$B(s) = B_1(s)B_2(s)...B_k(s)$$

donde $B_i(s)$ con $i = 1, 2, ..., k$ son las matrices resultantes de aplicar a la identidad I_m la misma operación elemental por columnas, en el anillo de las funciones racionales propias $\Re_p(s)$, que la que se aplicó a $T(s)$ para llegar a su forma de Hermite por columnas. Las matrices $B_i(s)$ son llamadas matrices elementales en el anillo $\Re_p(s)$.

También se puede demostrar que aplicar una serie de operaciones elementales por filas en $\Re_p(s)$ a una matriz $T(s)$, es equivalente a multiplicarla, por la izquierda, por una matriz bipropia.

Veamos un ejemplo para ilustrar las definiciones anteriores.

Ejemplo 3.1 *Obtengamos la forma de Hermite por columnas de:*

$$T(s) = \begin{bmatrix} \frac{1}{s+1} & \frac{1}{s+2} \\ \frac{1}{s+3} & \frac{1}{s+4} \end{bmatrix}$$

utilizando operaciones elementales para llegar a la forma $H(s)$ presentada en la Definición 3.4.

Primeramente multipliquemos la columna 1 por la función bipropia $\frac{s+1}{s}$, es decir, con $c_1 \left(\frac{s+1}{s}\right) \rightarrow c_1$ tenemos,

$$T(s)B_1(s) = \begin{bmatrix} \frac{1}{s} & \frac{1}{s+2} \\ \frac{s+1}{s(s+3)} & \frac{1}{s+4} \end{bmatrix}$$

donde $B_1(s)$ viene de realizar a la identidad la misma operación elemental, es decir,

$$B_1(s) = \begin{bmatrix} \frac{s+1}{s} & 0 \\ 0 & 1 \end{bmatrix}$$

Ahora, con $c_2 - \frac{s}{s+2}c_1 \rightarrow c_2$ tenemos,

$$T(s)B_1(s)B_2(s) = \begin{bmatrix} \frac{1}{s} & 0 \\ \frac{s+1}{s(s+3)} & \frac{2}{(s+2)(s+3)(s+4)} \end{bmatrix} \qquad B_2(s) = \begin{bmatrix} 1 & -\frac{s}{s+2} \\ 0 & 1 \end{bmatrix}$$

Después, con $c_2 \frac{(s+2)(s+3)(s+4)}{2s^3} \rightarrow c_2$ tenemos,

$$T(s)B_1(s)B_2(s)B_3(s) = \begin{bmatrix} \frac{1}{s} & 0 \\ \frac{s+1}{s(s+3)} & \frac{1}{s^3} \end{bmatrix} \qquad B_3(s) = \begin{bmatrix} 1 & 0 \\ 0 & \frac{(s+2)(s+3)(s+4)}{2s^3} \end{bmatrix}$$

Finalmente, con $c_1 - \frac{6s}{s+3}c_2 \rightarrow c_1$ tenemos,

$$H(s) = T(s)B_1(s)B_2(s)B_3(s)B_4(s) = \begin{bmatrix} \frac{1}{s} & 0 \\ \frac{s-2}{s^2} & \frac{1}{s^3} \end{bmatrix} \qquad B_4(s) = \begin{bmatrix} 1 & 0 \\ -\frac{6s}{s+3} & 1 \end{bmatrix}$$

De esta forma, la matriz $B(s)$ está dada por:

$$B(s) = B_1(s)B_2(s)B_3(s)B_4(s) = \begin{bmatrix} \frac{4s^2+16s+12}{s^2} & -\frac{0.5s^3+4s^2+9.5s+6}{s^3} \\ -\frac{3s^2+18s+24}{s^2} & \frac{0.5s^3+4.5s^2+13s+12}{s^3} \end{bmatrix}$$

de donde podemos comprobar que es bipropia y además que,

$$T(s)B(s) = \begin{bmatrix} \frac{1}{s+1} & \frac{1}{s+2} \\ \frac{1}{s+3} & \frac{1}{s+4} \end{bmatrix} \begin{bmatrix} \frac{4s^2+16s+12}{s^2} & -\frac{0.5s^3+4s^2+9.5s+6}{s^3} \\ -\frac{3s^2+18s+24}{s^2} & \frac{0.5s^3+4.5s^2+13s+12}{s^3} \end{bmatrix} = \begin{bmatrix} \frac{1}{s} & 0 \\ \frac{s-2}{s^2} & \frac{1}{s^3} \end{bmatrix} = H(s)$$

En el siguiente lema, presentado en [26], se establece la relación entre el interactor y la forma de Hermite por columnas de una matriz $T(s)$.

Lema 3.2 *Sea $T(s) \in \Re_p^{m\times m}(s)$ una matriz racional propia no singular y sea $H(s)$ su forma de Hermite por columnas. Entonces, el interactor de $T(s)$ se puede encontrar como:*

$$\Phi(s) = H(s)^{-1}$$

De este lema, de la Definición 3.1 y considerando la ecuación (3.1), se puede ver que:

$$\Phi(s)T(s) = H(s)^{-1}T(s) = B(s)^{-1}T(s)^{-1}T(s) = B^{-1}(s)$$

de modo que:

$$\lim_{s\to\infty} \Phi(s)T(s) = \lim_{s\to\infty} B(s)^{-1} = K \tag{3.2}$$

Esta última expresión (3.2), se puede demostrar con el Lema 3.1, ya que si $B(s)$ es bipropia, entonces $B^{-1}(s)$ también es bipropia, de modo que

$$\lim_{s\to\infty} B(s)^{-1}$$

es una matriz constante no singular.

De estos resultados se puede concluir que ya que $B(s)$ es bipropia y no contiene ceros ni polos al infinito, la forma de Hermite por columnas $H(s)$, y por lo tanto también el interactor $\Phi(s)$, contienen toda la información al infinito de $T(s)$. Este resultado también representa una posible forma de encontrar $\Phi(s)$, utilizar operaciones elementales para llegar a $H(s)$ y obtener su inversa.

Ejemplo 3.2 *Obtengamos el interactor de:*

$$T(s) = \begin{bmatrix} \frac{1}{s+1} & \frac{1}{s+2} \\ \frac{1}{s+3} & \frac{1}{s+4} \end{bmatrix}$$

Sabemos que la forma de Hermite por columnas de $T(s)$ es:

$$H(s) = \begin{bmatrix} \frac{1}{s} & 0 \\ \frac{s-2}{s^2} & \frac{1}{s^3} \end{bmatrix}$$

de modo que, según el Lema 3.1, podemos encontrar $\Phi(s)$ *como:*

$$\Phi(s) = H(s)^{-1} = \begin{bmatrix} s & 0 \\ 2s^2 - s^3 & s^3 \end{bmatrix}$$

Comprobemos que este es el interactor de $T(s)$ *verificando la ecuación (3.2).*

$$\lim_{s\to\infty} \Phi(s)T(s) = \lim_{s\to\infty} \begin{bmatrix} s & 0 \\ 2s^2 - s^3 & s^3 \end{bmatrix} \begin{bmatrix} \frac{1}{s} & 0 \\ \frac{s-2}{s^2} & \frac{1}{s^3} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 6 & 8 \end{bmatrix} = K$$

por otro lado,

$$\lim_{s\to\infty} B(s)^{-1} = \lim_{s\to\infty} \begin{bmatrix} \frac{24s^3+26s^4+9s^5+s^6}{24s^2+50s^3+35s^4+10s^5+s^6} & \frac{12s^3+19s^4+8s^5+s^6}{24s^2+50s^3+35s^4+10s^5+s^6} \\ \frac{48s^4+36s^5+6s^6}{24s^2+50s^3+35s^4+10s^5+s^6} & \frac{24s^4+32s^5+8s^6}{24s^2+50s^3+35s^4+10s^5+s^6} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 6 & 8 \end{bmatrix} = K$$

donde K *es evidentemente no singular.*

Hasta ahora se ha considerado que la matriz $T(s)$ es cuadrada, ahora se verán algunos resultados útiles para el caso de que no lo sea. El siguiente resultado resuelve el caso de que $T(s)$ tenga más columnas que filas.

Lema 3.3 *Sea* $T(s) \in \Re_p^{p\times m}(s)$ *una matriz racional propia de rango* p *y sea*

$$H(s) = \begin{bmatrix} \frac{1}{s^{f_1}} & & 0 \\ & & \\ & & 0 \\ & & \\ h_{p1}(s) & & \frac{1}{s^{f_p}} \end{bmatrix} = \begin{bmatrix} H_1(s) & 0 \end{bmatrix}$$

su forma de Hermite por columnas. Entonces, el interactor de $T(s)$ *se puede encontrar como:*

$$\Phi(s) = H_1(s)^{-1}$$

Por último, para el caso de que $T(s)$ tenga más filas que columnas, $\Phi(s)$ debe de formarse por partes, como se indica en el siguiente lema.

Lema 3.4 *Sea* $T(s) \in \Re_p^{p\times m}(s)$ *una matriz racional propia de rango* m *con las primeras* m *filas linealmente independientes. Entonces, el interactor de* $T(s)$ *está dado por:*

$$\Phi(s) = \begin{bmatrix} \Phi_m(s) & 0 \\ -y_1(s) & y_2(s) \end{bmatrix}$$

donde, si consideramos una factorización coprima derecha de $T(s)$,

$$T(s) = \begin{bmatrix} T_m(s) \\ T_{p-m}(s) \end{bmatrix} = \begin{bmatrix} N_m(s) \\ N_{p-m}(s) \end{bmatrix} D(s)^{-1} = N(s)D(s)^{-1}$$

entonces, $\Phi_m(s)$ *es el interactor de* $T_m(s)$ *y las matrices* $y_1(s)$ *y* $y_2(s)$ *son una factorización coprima izquierda de* $N_{p-m}(s)N_m(s)^{-1}$ *es decir,*

$$N_{p-m}(s)N_m(s)^{-1} = y_2(s)^{-1}y_1(s)$$

donde además, para asegurar la unicidad del interactor, $y_2(s)$ debe ser triangular inferior y en forma de Hermite (ver [34]).

Para este caso, la matriz

$$\lim_{s\to\infty} \Phi(s)T(s) = K$$

no sólo tendrá rango pleno, sino que además las $p-m$ últimas filas serán iguales a cero.

Veamos un ejemplo de este último caso.

Ejemplo 3.3 *Obtengamos el interactor de la matriz*

$$T(s) = \begin{bmatrix} \frac{1}{s+2} & 0 \\ \frac{2}{s+3} & \frac{s+1}{s+3} \\ \frac{1}{(s+2)(s+3)} & \frac{1}{s+3} \\ \frac{2s+7}{(s+2)(s+3)} & \frac{1}{s+3} \end{bmatrix} = \begin{bmatrix} T_m(s) \\ T_{p-m}(s) \end{bmatrix}$$

Por el Lema 3.1 podemos obtener,

$$\Phi_m(s) = \begin{bmatrix} s & 0 \\ 0 & 1 \end{bmatrix}$$

Ahora, tomando una factorización coprima derecha de $T(s)$ tenemos:

$$T(s) = \begin{bmatrix} 1 & 0 \\ 1 & s+1 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} s+2 & 0 \\ -1 & s+3 \end{bmatrix}^{-1} = \begin{bmatrix} N_m(s) \\ N_{p-m}(s) \end{bmatrix} D^{-1}(s)$$

de esta forma,

$$N_{p-m}(s)N_m^{-1}(s) = \begin{bmatrix} -\frac{1}{s+1} & \frac{1}{s+1} \\ \frac{2s+1}{s+1} & \frac{1}{s+1} \end{bmatrix}$$

Ahora, tomemos una factorización coprima izquierda de $N_{p-m}(s)N_m^{-1}(s)$,

$$\begin{bmatrix} -\frac{1}{s+1} & \frac{1}{s+1} \\ \frac{2s+1}{s+1} & \frac{1}{s+1} \end{bmatrix} = D_1^{-1}(s)N_1(s)$$

y finalmente, por medio de operaciones elementales por filas en $\Re(s)$, hacemos que $D_1(s)$ sea triangular inferior y en forma de Hermite. Así, las matrices y_1 y y_2 buscadas son,

$$y_2(s) = U(s)D_1(s) \qquad y_1(s) = U(s)N_1(s)$$

donde $U(s)$ es una matriz unimodular. Es evidente que, haciendo esto, las matrices y_1 y y_2 siguen siendo una factorización coprima izquierda de $N_{p-m}(s)N_m^{-1}(s)$ ya que:

$$y_2^{-1}(s)y_1(s) = D_1^{-1}(s)U^{-1}(s)U(s)N(s) = D_1^{-1}(s)N_1(s)$$

Asi pués, haciendo estas operaciones obtenemos,

$$y_1(s) = \begin{bmatrix} -1 & 1 \\ 2 & 0 \end{bmatrix} \quad y_2(s) = \begin{bmatrix} s+1 & 0 \\ -1 & 1 \end{bmatrix}$$

de modo que,

$$\Phi(s) = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & -1 & s+1 & 0 \\ -2 & 0 & -1 & 1 \end{bmatrix}$$

Comprobemos este resultado,

$$\lim_{s\to\infty} \Phi(s)T(s) = \lim_{s\to\infty} \begin{bmatrix} \frac{s}{s+2} & 0 \\ \frac{2}{s+3} & \frac{s+1}{s+3} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = K$$

lo cual es correcto según el Lema 3.4.

3.2 El algoritmo interactor.

De la definición 3.1, se han derivado tres casos particulares. Uno cuando $T(s)$ es cuadrada, otro cuando tiene más filas que columnas ($p > m$) y un último caso para cuando $T(s)$ tiene más columnas que filas ($p < m$). Para estos casos también se ha visto que el interactor o una parte de él (en el caso de que $p > m$), puede ser obtenido de la inversa de la forma de Hermite por columnas de la matriz $T(s)$ (o $T_m(s)$ para el caso de que $p > m$). Sin embargo, como se mencionó en el Capítulo 2, un algoritmo numérico basado en las operaciones elementales dista mucho de ser un algoritmo confiable ya que estas operaciones elementales, al implicar divisiones euclidianas entre polinomios, son numéricamente inestables y difíciles de programar.

En esta sección se presenta el algoritmo numérico confiable que se desarrolló para obtener el interactor de una forma alternativa y sin recurrir a las operaciones elementales. El siguiente algoritmo está basado en el procedimiento presentado por W. A. Wolovich y P. L. Falb. Para mayor detalle sobre algunos aspectos de lo que se presentará más adelante se puede consultar [34].

Para una explicación más clara del algoritmo, este se dividirá en dos casos, el primero para cuando $p \leq m$ y el segundo para cuando $p > m$, sin embargo, como se puede ver en el Anexo 1, en un solo programa se han implementado ambos casos.

3.2.1 Caso 1. Para T(s) de rango pleno por filas.

La idea básica del algoritmo consiste en proponer la i-ésima fila $\Phi_i(s)$ del interactor cuidando que la $i - ésima$ fila K_i sea linealmente independiente de las $i - 1$ filas anteriores, donde,

$$K_i = \lim_{s\to\infty} \Phi_i(s)T(s)$$

La forma de proponer las m filas del interactor es la siguiente:

Primeramente se obtienen los números μ_i que es el órden del cero al infinito de la $i-ésima$ fila de $T(s)$ (ver Capítulo 5), es decir μ_i es tal que:

$$\lim_{s\to\infty} s^{\mu_i} T_i(s) = \tau_i$$

con τ_i constante y distinto de cero y donde $T_i(s)$ es la $i-ésima$ fila de $T(s)$. Ahora se proponen las filas del interactor como,

$$\Phi'_i(s) = \begin{bmatrix} 0 & \cdots & s^{\mu_i} & 0 & \cdots & 0 \end{bmatrix} \tag{3.3}$$

y se evalua lo que sería la fila $i-ésima$ de K, esto es,

$$\tau_i = \lim_{s\to\infty} \Phi'_i(s) T(s)$$

Se toma efectivamente $\Phi_1(s) = \Phi'_1(s)$, de modo que $K_1 = \tau_1$. Para la siguiente fila se observa si τ_2 es linealmente independiente de K_1, si lo es entonces se toma $\Phi_2(s) = \Phi'_2(s)$ y $K_2 = \tau_2$. Si no son linealmente independientes, se toma $\Phi'_{21}(s) = \Phi'_2$, $\tau_{21} = \tau_2$ y se propone una nueva fila

$$\Phi'_{22}(s) = s^{\mu_{22}} \left[\Phi'_{21}(s) - \alpha_{22}\Phi_1(s)\right]$$

con,

$$\tau_{21} = \alpha_{22} K_1$$

y tal que:

$$\lim_{s\to\infty} \Phi'_{22}(s) T(s) = \tau_{22}$$

es constante distinto de cero.

Ahora, si τ_{22} es linealmente independiente de K_1, se toma $\Phi_2(s) = \Phi'_{22}(s)$ y $K_2 = \tau_{22}$. Si no son linealmente independientes, se propone una nueva fila $\Phi'_{23}(s)$ de forma similar. Este proceso es iterativo hasta lograr la independencia lineal. Una vez lograda la independencia lineal se pasa a las siguientes filas y se realiza el mismo proceso con todas hasta encontrar el interactor.

En el paso general del proceso de construcción, consideremos el índice i como el número de fila que se está construyendo y el índice j como el número de veces que se ha intentado obtener la independencia lineal, entonces:

$$\Phi'_{ij}(s) = s^{\mu_{ij}} \left[\Phi'_{i,j-1}(s) - \alpha_{ij}\Phi_{i-1}(s)\right] \tag{3.4}$$

con

$$\tau_{i,j-1} = \alpha_{ij} K_{i-1} \tag{3.5}$$

y tal que:

$$\lim_{s\to\infty} \Phi'_{ij}(s) T(s) = \tau_{ij} \tag{3.6}$$

es constante distinto de cero.

El siguiente ejemplo ilustra la aplicación de este algoritmo.

Ejemplo 3.4 *Obtengamos, por medio del algoritmo anterior, el interactor de:*

$$T(s) = \begin{bmatrix} \frac{1}{s+1} & \frac{1}{s+2} \\ \frac{1}{s+3} & \frac{1}{s+4} \end{bmatrix}$$

Primeramente obtengamos los valores μ_i. Primero, para la fila 1,

$$\lim_{s\to\infty} sT_1(s) = \begin{bmatrix} 1 & 1 \end{bmatrix} = \tau_1$$

por lo tanto, $\mu_1 = 1$. Para la fila 2,

$$\lim_{s\to\infty} sT_2(s) = \begin{bmatrix} 1 & 1 \end{bmatrix} = \tau_2$$

por lo tanto, también, $\mu_2 = 1$.

Ahora, según (3.3), la primera fila de $\Phi(s)$ está dada por,

$$\Phi_1(s) = \begin{bmatrix} s & 0 \end{bmatrix}$$

y entonces,

$$\lim_{s\to\infty} \Phi_1(s)T(s) = \lim_{s\to\infty} \begin{bmatrix} s & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{s+1} & \frac{1}{s+2} \\ \frac{1}{s+3} & \frac{1}{s+4} \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix} = K_1 = \tau_1$$

La segunda fila de $\Phi(s)$ estaría dada, según (3.3) por,

$$\Phi_2'(s) = \begin{bmatrix} 0 & s \end{bmatrix}$$

sin embargo,

$$\tau_2 = \lim_{s\to\infty} \Phi_2'(s)T(s) = \lim_{s\to\infty} \begin{bmatrix} 0 & s \end{bmatrix} \begin{bmatrix} \frac{1}{s+1} & \frac{1}{s+2} \\ \frac{1}{s+3} & \frac{1}{s+4} \end{bmatrix} = \lim_{s\to\infty} \begin{bmatrix} \frac{s}{s+3} & \frac{s}{s+4} \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

es linealmente dependiente de K_1, por lo tanto hay que proponer otra fila según la expresión (3.4), es decir:

$$\Phi_{22}'(s) = s^{\mu_{22}} [\Phi_{21}'(s) - \alpha_{22}\Phi_1(s)] = s^{\mu_{22}} \left(\begin{bmatrix} 0 & s \end{bmatrix} - \alpha_{22} \begin{bmatrix} s & 0 \end{bmatrix} \right)$$

donde, según (3.5), $\alpha_{22} = 1$, y donde, para cumplir con (3.6), $\mu_{22} = 1$, esto es,

$$\Phi_{22}'(s) = s \begin{bmatrix} -s & s \end{bmatrix}$$

Con esta nueva fila tenemos que,

$$\tau_{22} = \lim_{s\to\infty} \Phi_{22}'(s)T(s) = \lim_{s\to\infty} \begin{bmatrix} -\frac{s}{s+1} + \frac{s}{s+3} & -\frac{s}{s+2} + \frac{s}{s+4} \end{bmatrix} = \begin{bmatrix} -2 & -2 \end{bmatrix}$$

que sigue siendo linealmente dependiente de K_1, por lo tanto, se debe proponer una nueva fila 2,

$$\Phi_{23}'(s) = s^{\mu_{23}} [\Phi_{22}'(s) - \alpha_{23}\Phi_1(s)] = s^{\mu_{23}} \left(\begin{bmatrix} -s & s \end{bmatrix} - \alpha_{23} \begin{bmatrix} s & 0 \end{bmatrix} \right)$$

donde de nuevo, según (3.5) y (3.6), podemos ver que $\alpha_{23} = -2$ y que $\mu_{23} = 1$ y por tanto,

$$\Phi_{23}'(s) = s \begin{bmatrix} -s^2 + 2s & s^2 \end{bmatrix}$$

Con esta nueva fila,

$$\tau_{23} = \lim_{s\to\infty} \Phi'_{23}(s)T(s) = \lim_{s\to\infty} \left[\begin{array}{cc} \frac{6s^2}{(s+1)(s+3)} & \frac{8s^2}{(s+2)(s+4)} \end{array} \right] = \left[\begin{array}{cc} 6 & 8 \end{array} \right]$$

que es linealmente independiente de K_1, por lo tanto, $\Phi_2(s) = \Phi'_{23}(s)$ y $K_2 = \tau_{23}$.

Finalmente, el interactor es

$$\Phi(s) = \left[\begin{array}{c} \Phi_1(s) \\ \Phi_2(s) \end{array} \right] = \left[\begin{array}{cc} s & 0 \\ 2s^2 - s^3 & s^3 \end{array} \right]$$

el cual corresponde exáctamente a lo obtenido, por medio de operaciones elementales, en el Ejemplo 3.2.

Comprobemos este resultado,

$$\lim_{s\to\infty} \left[\begin{array}{c} \Phi_1(s) \\ \Phi_2(s) \end{array} \right] T(s) = \lim_{s\to\infty} \left[\begin{array}{cc} \frac{s}{s+1} & \frac{s}{s+2} \\ \frac{6s^2}{(s+1)(s+3)} & \frac{8s^2}{(s+2)(s+4)} \end{array} \right] = \left[\begin{array}{cc} 1 & 1 \\ 6 & 8 \end{array} \right] = \left[\begin{array}{c} K_1 \\ K_2 \end{array} \right]$$

Para la programación, en MATLAB, de este algoritmo, se utilizaron funciones del Polynomial Toolbox para introducir y manejar las matrices polinomiales. Sin embargo, esta plataforma, no permite el manejo directo de matrices racionales como el caso de $T(s)$ ni el cálculo numérico de los límites al infinito de una función racional en s. Asi pues, para programar este algoritmo, se formularon los siguientes resultados que ayudan a resolver estos problemas.

Primeramente, para manejar las matrices racionales lo que se hizo fue transformar la matriz original en una matriz polinomial y trabajar con ella. Para eso se multiplicaron todos los denominadores y el resultado se multiplicó por cada numerador obteniendo así una matriz polinomial. Cabe mencionar que obviamente, en lugar de multiplicar todos los denominadores se pudo haber obtenido el mínimo común múltiplo de ellos, sin embargo, esto requería de más tiempo de ejecución.

El problema con esta operación es que modifica el orden del cero al infinito por fila de la matriz $T(s)$ original y por lo tanto el valor de los enteros μ_i. El siguiente lema nos garantiza la forma de recuperar estos enteros, que como hemos explicado, son base del algoritmo.

Lema 3.5 *Sea $T(s)$ una matriz racional propia de la siguiente forma:*

$$T(s) = \left[\begin{array}{cc} \frac{a_{11}(s)}{b_{11}(s)} & \frac{a_{1n}(s)}{b_{1n}(s)} \\ & \\ \frac{a_{m1}(s)}{b_{m1}(s)} & \frac{a_{mn}(s)}{b_{mn}(s)} \end{array} \right]$$

y sea $m(s) = \prod_{i=1}^{m} \prod_{j=1}^{n} b_{ij}(s)$, entonces:

$$\mu_i = \deg m(s) - \deg T'_i(s)$$

donde $\deg T'_i(s)$ es el grado por filas de $T'_i(s)$ y donde

$$T'(s) = m(s)T(s)$$

es una matriz polinomial.

Demostración. Recordemos que,

$$\lim_{s\to\infty} s^{\mu_i} T_i(s) = \tau_i \tag{3.7}$$

donde τ_i es constante distinto de cero.

Consideremos que $T_i(s)$ es de un sólo elemento, es decir,

$$T_i(s) = \frac{a_n s^n + a_{n-1}s^{n-1} + ... + a_0}{b_p s^p + b_{p-1}s^{p-1} + ... + b_0}$$

entonces,

$$\lim_{s\to\infty} T_i(s) = \lim_{s\to\infty} \frac{a_n s^n + a_{n-1}s^{n-1} + ... + a_0}{b_p s^p + b_{p-1}s^{p-1} + ... + b_0} = 0$$

siempre que $p > n$. Para hacer este límite distinto de cero, se debe cumplir que $p = n$.

Por lo tanto, ya que $T_i(s)$ es racional propia, para cumplir con (3.7),

$$\mu_i = p - n$$

de modo que,

$$\lim_{s\to\infty} s^{p-n} T_i(s) = \lim_{s\to\infty} \frac{a_n s^p + a_{n-1}s^{p-1} + ... + a_0}{b_p s^p + b_{p-1}s^{p-1} + ... + b_0} = \frac{a_n}{b_p}$$

Ahora, si $T_i(s)$ tiene más de un elemento,

$$T_i(s) = \left[\begin{array}{ccc} \frac{a_i(s)}{b_1(s)} & \frac{a_2(s)}{b_2(s)} & \frac{a_k(s)}{b_k(s)} \end{array} \right]$$

se puede ver fácilmente que,

$$\mu_i = \min \{\deg b_j(s) - \deg a_j(s)\} \quad \text{con} \quad j = 1, 2, ..., k \tag{3.8}$$

Supongamos que el elemento $r - ésimo$ cumple con (3.8), es decir,

$$\mu_i = \deg b_r(s) - \deg a_r(s)$$

entonces, el elemento $r - ésimo$ de $T_i'(s)$ tendrá el mayor grado de la fila, ya que,

$$\deg a_r'(s) = \deg a_r(s) - \deg b_r(s) + \deg m(s)$$

por lo tanto, el grado por fila de $T_i'(s)$ será $\deg a_r'(s)$.

Finalmente,

$$\mu_i = \deg m(s) - \deg T_i'(s) = \deg m(s) - \deg a_r(s) + \deg b_r(s) - \deg m(s)$$

$$\mu_i = \deg b_r(s) - \deg a_r(s)$$

■

Para obtener los límites cuando s tiende al infinito de las distintas funciones racionales que aparecen a lo largo del desarrollo del algoritmo, también se formuló el siguiente resultado.

Lema 3.6 *Sea $T(s)$ una matriz racional propia de la siguiente forma:*

$$T(s) = \begin{bmatrix} \frac{a_{11}(s)}{b_{11}(s)} & & \frac{a_{1n}(s)}{b_{1n}(s)} \\ & & \\ \frac{a_{m1}(s)}{b_{m1}(s)} & & \frac{a_{mn}(s)}{b_{mn}(s)} \end{bmatrix}$$

sea $m(s) = \prod_{i=1}^{m} \prod_{j=1}^{n} b_{ij}(s)$, y sea

$$T'(s) = m(s)T(s)$$

donde t_i es el vector de coeficientes de mayor grado en s de la fila $T_i'(s)$, entonces:

$$\lim_{s\to\infty} s^{\mu_i} T_i(s) = t_i$$

Demostración. Este resultado se puede demostrar, con base en la prueba del lema anterior, ya que suponiendo que el elemento $r - ésimo$ de

$$T_i(s) = \begin{bmatrix} \frac{a_i(s)}{b_1(s)} & \frac{a_2(s)}{b_2(s)} & & \frac{a_k(s)}{b_k(s)} \end{bmatrix}$$

cumple con (3.8), entonces,

$$\lim_{s\to\infty} s^{p_r - n_r} T_i(s) = \begin{bmatrix} 0 & & \frac{a_{rn}}{b_{rp}} & 0 & & 0 \end{bmatrix}$$

lo cual corresponde con el vector de coeficientes de mayor grado en s de $T_i'(s)$ ya que el elemento de mayor grado es efectivamente el elemento $r - ésimo$. ■

Resumiendo, con este par de resultados, se puede ver como los enteros μ_i pueden ser obtenidos a partir del grado del polinomio $m(s)$ y del grado por filas de $T_i'(s)$, además, el límite

$$\lim_{s\to\infty} s^{\mu_i} T_i(s)$$

puede ser obtenido como los coeficientes líderes de otro vector polinomial. Así pues, las operaciones con matrices racionales necesarias, en un principio, para obtener el interactor, son reducidas a operaciones con polinomios y matrices polinomiales, las cuales son fácilmente manejadas por MATLAB y el Polynomial Toolbox.

Observe que si los vectores

$$\tau_i = \lim_{s\to\infty} \Phi_i'(s)T(s)$$

con $\Phi_i'(s)$ dado por (3.3) fueran siempre linealmente independientes, los números μ_i podrían ser obtenidos directamente, de la expresión (3.8), a partir de la matriz original $T(s)$. El uso de los Lemas 3.5 y 3.6 se justifica ya que en general siempre hay alguna dependencia lineal y es necesario proponer otras filas del interactor distintas a las propuestas en (3.3). Para hacer esto es necesaria la manipulación de $T(s)$ lo cual implica, como se ha dicho, que deba ser transformada a matriz polinomial de modo que pueda ser manejada en MATLAB [30].

Las otras operaciones necesarias para desarrollar este algoritmo, como lo son: la evaluación de la independencia lineal entre vectores y la determinación de los múltiplos α_{ij} de (3.5) son realizadas con funciones de MATLAB y del Polynomial Toolbox ya existentes.

Veamos entonces como se resolvería el ejemplo 3.4 con nuestro algoritmo.

Ejemplo 3.5 *Obtengamos el interactor de:*

$$T(s) = \begin{bmatrix} \frac{1}{s+1} & \frac{1}{s+2} \\ \frac{1}{s+3} & \frac{1}{s+4} \end{bmatrix}$$

utilizando el algoritmo anterior.

Primeramente obtengamos la matriz polinomial equivalente. Con $m(s) = (s+1)(s+2)(s+3)(s+4)$, *tenemos,*

$$T'(s) = \begin{bmatrix} (s+2)(s+3)(s+4) & (s+1)(s+3)(s+4) \\ (s+1)(s+2)(s+4) & (s+1)(s+2)(s+3) \end{bmatrix}$$

Ahora, utilizando el Lema 3.4, obtengamos los valores μ_i, *para la fila 1,*

$$\mu_1 = \deg m(s) - \deg T_1'(s) = 4 - 3 = 1$$

y para la fila 2,

$$\mu_2 = \deg m(s) - \deg T_2'(s) = 4 - 3 = 1$$

Por lo tanto, usando el Lema 3.5,

$$\begin{aligned} \tau_1 &= \begin{bmatrix} 1 & 1 \end{bmatrix} \\ \tau_2 &= \begin{bmatrix} 1 & 1 \end{bmatrix} \end{aligned}$$

Como se puede ver, los resultados son los mismos que si se trabajara con la matriz racional.
Ahora, según (3.3), las filas del interactor serían

$$\Phi_1(s) = \begin{bmatrix} s & 0 \end{bmatrix}, \quad \Phi_2(s) = \begin{bmatrix} 0 & s \end{bmatrix}$$

pero, τ_2 *es dependiente de* $K_1 = \tau_1$, *ya que si probamos, como lo haría el programa interactor,*

$$rank \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

el resultado es uno, lo que significa que la segunda fila es dependiente de la primera, entonces debemos proponer otra fila

$$\Phi_{22}'(s) = s^{\mu_{22}} [\Phi_{21}'(s) - \alpha_{22}\Phi_1(s)] = s^{\mu_{22}} \left(\begin{bmatrix} 0 & s \end{bmatrix} - \alpha_{22} \begin{bmatrix} s & 0 \end{bmatrix} \right)$$

Para encontrar α_{22}, *el programa interactor utiliza la función axb del Polynomial. Para encontrar* μ_{22} *se vuelve a aplicar el Lema 3.5. Asi sucesivamente, se puede ver como con los Lemas 3.5, 3.6 y con las funciones de MATLAB y el Polynomial Toolbox es posible la obtención del interactor,*

$$\Phi(s) = \begin{bmatrix} s & 0 \\ 2s^2 - s^3 & s^3 \end{bmatrix}$$

3.2.2 Caso 2. Para T(s) de rango pleno por columnas.

Para este caso, un vez programado el caso anterior, $\Phi_m(s)$ se encuentra sin ningún problema.

Las matrices $y_1(s)$ y $y_2(s)$ se encuentran con las funciones del Polynomial Toolbox de la siguiente forma:

Primeramente se obtiene un factorización coprima derecha de $T(s)$,

$$T(s) = \begin{bmatrix} N_m(s) \\ N_{p-m}(s) \end{bmatrix} D^{-1}(s)$$

con la función *rat2rmf*. Después se obtiene una factorización coprima izquierda de $N_{p-m}(s)N_m^{-1}(s)$,

$$N_{p-m}(s)N_m^{-1}(s) = D_1^{-1}(s)N_1(s)$$

con la función *rat2lmf*. Ahora, para transformar $D_1(s)$ en la forma requerida necesitamos, como se explicó en el Ejemplo 3.3, realizar operaciones por filas sobre $D_1(s)$. La función *hermite* del Polynomial realiza operaciones por filas para obtener la forma de Hermite (por filas) de la matriz que se le de como entrada, el problema es que esta forma de Hermite es triangular superior y se requiere que $y_2(s)$ sea triangular inferior.

Para resolver este problema se hace lo siguiente. Se cambian de orden las columnas de $D_1(s)$ (la última se pone como primera, la penúltima como segunda, etc...) y se obtiene, con la función *hermite*, $D_2(s)$ que es la forma de Hermite por filas de esta nueva matriz. Finalmente, la matriz $y_2(s)$ se obtiene de cambiar de orden filas y columnas de $D_2(s)$. Con esto, al realizar dos veces cambio de columnas, efectivamente se esta obteniendo $y_2(s)$ en la forma requerida y únicamente con operaciones por filas. Ahora, para que $y_2(s)$ y $y_1(s)$ sigan siendo coprimas izquierdas, tomemos $y_1(s) = y_2(s)D_1^{-1}(s)N(s)$ de modo que,

$$y_2^{-1}(s)y_1(s) = y_2^{-1}(s)y_2(s)D_1^{-1}(s)N(s) = D_1^{-1}(s)N1(s)$$

Ejemplo 3.6 *Veamos el ejemplo 3.3 con este procedimiento.*
Tenemos que:

$$N_{p-m}(s)N_m^{-1}(s) = \begin{bmatrix} -\frac{1}{s+1} & \frac{1}{s+1} \\ \frac{2s+1}{s+1} & \frac{1}{s+1} \end{bmatrix}$$

Ahora, tomemos una factorización coprima izquierda de $N_{p-m}(s)N_m^{-1}(s)$,

$$\begin{bmatrix} -\frac{1}{s+1} & \frac{1}{s+1} \\ \frac{2s+1}{s+1} & \frac{1}{s+1} \end{bmatrix} = D_1^{-1}(s)N_1(s) = \begin{bmatrix} -0.41 & 0.41 \\ 0.46+0.76s & 0.46+0.15s \end{bmatrix}^{-1} \begin{bmatrix} 0.82 & 0 \\ 0.3s & 0.91 \end{bmatrix}$$

Cambiando de orden las columnas de $D_1(s)$ *y obteniendo su forma de Hermite por filas tenemos,*

$$D_2(s) = \begin{bmatrix} 1 & -1 \\ 0 & s+1 \end{bmatrix}$$

y por lo tanto, cambiando filas y columnas,

$$y_2(s) = \begin{bmatrix} s+1 & 0 \\ -1 & 1 \end{bmatrix}$$

Finalmente,

$$y_1(s) = y_2(s)D_1^{-1}(s)N_1(s) = \begin{bmatrix} -1 & 1 \\ 2 & 0 \end{bmatrix}$$

de modo que,

$$\Phi(s) = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & -1 & s+1 & 0 \\ -2 & 0 & -1 & 1 \end{bmatrix}$$

Las funciones *rat2lmf*, *rat2rmf* y *hermite* están basadas en las técnicas de Sylvester e Interpolación vistas en el Capítulo 2, por lo tanto también son funciones confiables.

Resumiendo, el método presentado en [34] y que hemos reformulado en este capítulo, permite obtener el interactor de un sistema lineal sin recurir a las operaciones elementales. Por otra parte, los Lemas 3.5 y 3.6 permiten la implementación de este algoritmo en MATLAB de una forma confiable ya que tampoco implican la utilización de operaciones elementales ni operaciones polinomiales básicas como la división euclidiana. Además, el uso de funciones confiables del Polynomial Toolbox y funciones numéricas de MATLAB permite también la solución, de manera confiable, de otros problemas relacionados con la obtención del interactor como lo son: la prueba de la independencia lineal entre vectores, o la obtención de las matrices y_1 y y_2. De esta forma, podemos decir que el algoritmo *interactor* es, para cualquier caso, un algoritmo confiable.

Capítulo 4

Desacoplamiento de sistemas lineales.

Como mencionamos en el capítulo anterior, el interactor de un sistema lineal multivariable contiene la información al infinito del sistema, y es un invariante completo en el sentido de que es la parte del sistema que no puede ser modificada mediante una retroalimentación de estado regular. Desde este punto de vista, la información contenida en esta matriz juega un papel determinante en la existencia de la solución de algunos problemas de control.

En este capítulo se estudia uno de estos problemas, a saber, el desacoplamiento de sistemas lineales multivariables.

Se comenzará con la definición del problema, posteriormente se presentan las condiciones de solución, existentes en la literatura, en términos del interactor del sistema. El objetivo es aplicar el algoritmo desarrollado para obtener el interactor y con él determinar si el sistema en cuestión es desacoplable y obtener una retroalimentación que logre el desacoplamiento, en caso de que el problema tenga solución. En la segunda sección de este capítulo se presenta el algoritmo desarrollado para resolver este problema de desacoplamiento.

4.1 El problema del desacoplamiento de sistemas lineales.

El desacoplamiento de sistema dinámicos es uno de los tópicos más ampliamente tratados en teoría de control, y existe una gran cantidad de publicaciones y resultados al respecto. Básicamente, se dice que un sistema dinámico es desacoplable si existe una retroalimentación de estado tal que en el sistema en lazo cerrado, cada nueva entrada controle de manera independiente a cada salida del sistema.

La ventaja de lograr sistemas desacoplados es evidente. El hecho de que la $i-ésima$ entrada afecte solamente a la $i-ésima$ salida permite controlar cada salida como si se tratasen de sistemas independientes.

El problema de desacoplamiento puede mostrar diversas variantes, dependiendo por ejemplo, de la estructura del sistema en lazo cerrado, del tipo de retroalimentación que se utilice, del hecho de que el sistema tenga el mismo número de entradas que salidas, etc. No es la intención de este trabajo presentar un estudio general de este problema, por lo que nos restringiremos a casos particulares. Se considera,

primeramente, el desacoplamiento línea por línea de sistemas cuadrados y no cuadrados mediante retroalimentación estática de estado, y posteriormente el desacoplamiento línea por línea de sistemas cuadrados y no cuadrados mediante retroalimentación dinámica.

4.1.1 Desacoplamiento por retroalimentación estática.

Se dice que el sistema lineal multivariable de n estados, p salidas y m entradas descrito, en espacio de estado, por:

$$(A,B,C)\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) \end{cases} \tag{4.1}$$

donde $x \in \Re^n$, $u \in \Re^m$, $y \in \Re^p$, $A \in \Re^{n\times n}$, $B \in \Re^{n\times m}$ y $C \in \Re^{m\times n}$, y con matriz de transferencia la matriz racional propia $T(s) \in \Re_p^{p\times m}$ obtenida por:

$$T(s) = C(sI - A)^{-1}B$$

es desacoplable por retroalimentación de estado estática si existe una retroalimentación de la forma

$$u(t) = Fx(t) + Gv(t) \tag{4.2}$$

donde $F \in \Re^{m\times n}$ y $G \in \Re^{m\times p}$ son matrices constantes, tal que en el sistema retroalimentado $(A + BF, BG, C)$ la entrada $v_i(t)$ controle solamente a la salida $y_i(t)$ sin afectar a las otras salidas.

En términos de función de transferencia, observe que la formulación anterior es equivalente a la existencia de una retroalimentación de estado (4.2) tal que la función de transferencia en lazo cerrado es una matriz diagonal no singular.

Cuando $p = m$ es decir, $T(s)$ es cuadrada, podemos hablar de desacoplamiento regular (donde G deberá ser no singular). Por otro lado, cuando $m > p$, es decir, $T(s)$ tiene más columnas que filas, entonces se hablará de desacoplamiento no regular.

Existen varios resultados equivalentes que establecen las condiciones bajo las cuales un sistema es desacoplable mediante retroalimentación estática regular. Dichas condiciones pueden establecerse a partir de los ceros al infinito del sistema, de los ceros al infinito por líneas, de los órdenes esenciales y del interactor del sistema. La condición que utilizaremos en nuestro algoritmo será la siguiente, la cual está en función del interactor del sistema.

Lema 4.1 *Un sistema lineal (A,B,C) de n estados, p entradas y p salidas y cuya matriz de transferencia es $T(s) \in \Re_p^{p\times p}(s)$, es desacoplable por retroalimentación regular estática si y solo si el interactor $\Phi(s)$ de $T(s)$ es diagonal.*

En el caso del desacoplamiento estático no regular, es decir, cuando $T(s)$ no es cuadrada, las condiciones de este lema son suficientes pero no necesarias. Cabe mencionar que, de hecho, el problema del desacoplamiento estático no regular, continua sin ser resuelto completamente hasta la fecha.

Evidentemente que, con el algoritmo del interactor desarrollado, hacer un programa que resuelva la parte de saber si un sistema es desacoplable está resuelta. Ahora, el problema es como encontrar la retroalimentación estática adecuada para desacoplar dicho sistema. Para este problema se han desarrollado varios resultados, sin embargo, sólo para el caso regular estático se ha expresado el problema como una

ecuación polinomial cuyas soluciones proporcionan la retroalimentación $u(t) = Fx(t) + Gv(t)$ buscada. Este resultado se presenta a continuación.

Consideremos un sistema (A, B, C) con matriz de transferencia $T(s)$ cuadrada e interactor $\Phi(s)$ y tomemos una factorización coprima derecha $N_1(s)D^{-1}(s)$ de (A, B, I_n) donde $D(s)$ sea reducida por columnas.

Ahora, el problema de encontrar la retroalimentación estática que desacopla al sistema se reduce a encontrar una solución particular constante con X no singular, de la ecuación

$$XD(s) + YN_1(s) = \Phi(s)T(s)D(s) \tag{4.3}$$

donde la retroalimentación está dada por:

$$G = X^{-1} \quad \text{y} \quad F = -X^{-1}Y$$

Se puede demostrar que la acción de $u(t) = Fx(t) + Gv(t)$, obtenida de esta forma, es equivalente a la acción de un compensador $Co(s)$ que multiplicando por la derecha a $T(s)$ produce un lazo cerrado

$$C(sI - A - BF)^{-1}BG = T(s)Co(s) = \Phi^{-1}(s) = diag\left\{\frac{1}{s^{n_i}}\right\} = T_{LC}(s)$$

donde n_i son los órdenes de los ceros al infinito del sistema (ver Capítulo 5)

Ejemplo 4.1 *Sea el siguiente sistema* (A, B, C), *donde,*

$$A = \begin{bmatrix} 2 & 8 & 0 & 6 & 12 \\ 1 & 0 & 0 & 0 & 0 \\ -25 & -50 & -10 & -53 & -74 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & -5 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 6 & 0 & 1 & 8 \\ 1 & -9 & 0 & 0 & -12 \end{bmatrix}$$

y cuya matriz de transferencia es:

$$T(s) = \begin{bmatrix} \frac{1}{(s+1)^2} & \frac{1}{(s+1)(s+2)} \\ \frac{-6}{(s+1)(s+2)^2} & \frac{s-3}{(s+2)^2} \end{bmatrix}$$

El interactor de $T(s)$ *obtenido con el algoritmo del capítulo anterior es*

$$\Phi(s) = \begin{bmatrix} s^2 & 0 \\ 0 & s \end{bmatrix}$$

como podemos ver, $\Phi(s)$ es diagonal y por lo tanto, según el Lema 4.1, el sistema es desacoplable por retroalimentación estática.

Para obtener la retroalimentación que desacopla a este sistema, tomemos la factorización,

$$(sI-A)^{-1}B = N_1(s)D^{-1}(s) = \begin{bmatrix} 0 & s \\ 0 & 1 \\ s^2 & 0 \\ s & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 14+23s+10s^2+s^3 & 10+15s+5s^2 \\ -12-6s & -8-2s+s^2 \end{bmatrix}^{-1}$$

Con estas matrices, la ecuación polinomial

$$XD(s)+YN_1(s) = \Phi(s)T(s)D(s)$$

tiene como una solución particular,

$$X = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} -13 & -2 & -2 & -17 & -2 \\ -7 & 8 & 0 & -6 & 12 \end{bmatrix}$$

Y por lo tanto, finalmente,

$$G = X^{-1} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

$$F = -X^{-1}Y = \begin{bmatrix} 6 & 10 & 2 & 11 & 14 \\ 7 & -8 & 0 & 6 & -12 \end{bmatrix}$$

Ahora, para demostrar que F y G efectivamente desacoplan al sistema, tomemos el lazo cerrado,

$$C(sI-A-BF)^{-1}BG = \begin{bmatrix} \frac{1}{s^2} & 0 \\ & \\ 0 & \frac{1}{s} \end{bmatrix} = T_{LC} = \Phi^{-1}(s)$$

4.1.2 Desacoplamiento por retroalimentación dinámica.

La formulación del problema de desacoplamiento dinámico es análoga a la del desacoplamiento estático. Se dice que el sistema 4.1 es desacoplable por retroalimentación dinámica si existe una retroalimentación, en función de la frecuencia,

$$u(s) = F(s)x(s) + Gv(s)$$

donde $F(s) \in \Re_p^{m\times n}(s)$ es una matriz racional propia y $G \in \Re^{m\times p}$ es una matriz constante tal que en el sistema retroalimentado $(A+BF, BG, C)$ la entrada $v_i(t)$ controle solamente a la salida $y_i(t)$ sin afectar a las otras.

De igual forma, cuando $p = m$ es decir, $T(s)$ es cuadrada, podemos hablar de desacoplamiento dinámico regular (donde G deberá ser no singular). Por otro lado, cuando $m > p$, es decir, $T(s)$ tiene más columnas que filas, hablaremos de desacoplamiento dinámico no regular.

Los resultados del lema 4.1 también son análogos a este caso, es decir, un sistema cuadrado es desacoplable por retroalimentación regular dinámica si y sólo si su interactor es diagonal. Las condiciones, del Lema 4.1 también son únicamente suficientes para el caso no regular (sistemas no cuadrados), sin embargo, para este caso si existe otro resultado que expresa las condiciones necesarias y suficientes.

Consideremos la siguiente definición.

Definición 4.1 *Sea $A(s)$ una matriz polinomial de $p \times m$. El rango al infinito por columnas de $A(s)$ se define como el rango (sobre los reales) de la matriz $A_0 = \begin{bmatrix} a_1 & & a_m \end{bmatrix}$ donde*

$$a_i = \lim_{s \to \infty} A_i(s) s^{-r_i}$$

son vectores constantes distintos de cero y donde $A_i(s)$ es la $i-$ésima columna de $A(s)$,

Ahora, las condiciones necesarias y suficientes para que un sistema no cuadrado sea desacoplable por retroalimentación dinámica son las siguientes [4]:

Lema 4.2 *Un sistema lineal (A, B, C) de n estados, m entradas y p salidas donde $p < m$ y cuya matriz de transferencia es $T(s) \in \Re_p^{p \times m}(s)$ es desacoplable por retroalimentación dinámica no regular si y solo si,*

$$m \geq 2p - k \tag{4.4}$$

donde k es el rango al infinito por columnas del interactor $\Phi(s)$ de $T(s)$.

Estas condiciones son únicamente necesarias para retroalimentación estática, es decir, para que un sistema no cuadrado sea desacoplado con retroalimentación estática se debe cumplir (4.4).

Ejemplo 4.2 *Analicemos si el siguiente sistema no cuadrado, cuya matriz de transferencia es*

$$T(s) = \begin{bmatrix} s^{-1} & 0 & 0 & s^{-2} & 0 \\ 0 & s^{-1} & 0 & s^{-3} & 0 \\ -s^{-1} & -s^{-1} & s^{-2} & -s^{-2} - s^{-3} & s^{-3} \end{bmatrix}$$

es desacoplable.

Primeramente obtengamos el interactor,

$$\Phi(s) = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ s^2 & s^2 & s^2 \end{bmatrix}$$

y probemos la condición (4.4)

Según la Definición 4.1,

$$\Phi_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

por lo tanto, $k = 1$ y la condición (4.4)

$$m \geq 2p - k$$

$$5 \geq 2(3) - 1 = 5$$

se cumple, lo que significa que el sistema es desacoplable por retroalimentación dinámica.

Observe que no puede asegurarse si el sistema es desacoplable por retroalimentación estática, ya que se cumple la condición (4.4) (condición necesaria) pero el interactor no es diagonal (condición suficiente).

El problema de encontrar la retroalimentación dinámica de un sistema desacoplable, cuadrado o no cuadrado, tampoco está totalmente resuelto, existen algunos resultados pero aún no se ha expresado el problema como la solución de una ecuación polinomial como en el caso de retroalimentación regular estática.

4.2 El algoritmo desacop.

Resumiendo lo que se presentó en la sección anterior, si el sistema a estudiar es cuadrado (desacoplamiento regular), el sistema será desacoplable si y solo si su interactor es diagonal y se podrá utilizar retroalimentación estática, para la cual, el problema de como determinarla ya está resuelto.

Si el sistema a estudiar tiene más entradas que salidas (desacoplamiento no regular), el sistema será desacoplable por retroalimentación dinámica si y sólo si se cumpla la condición (4.4). Si además se cumple que el interactor es diagonal, se puede asegurar que el sistema también podrá ser desacoplado con retroalimentación estática. Si por otro lado, no se cumple la condición (4.4), entonces el sistema no será desacoplable por ningun tipo de retroalimentación. El problema de como determinar la retroalimentación no regular tampoco tiene hasta el momento una solución completa.

Con base en estos resultados, se desarrolló el algoritmo *desacop* el cual, por lo tanto, tiene las siguientes características:

1) Es capaz de determinar si un sistema cuadrado es desacoplable por retroalimentación estática y dinámica. En el caso de que el sistema estudiado sea desacoplable, el algoritmo es capaz de determinar también las matrices F y G de la retroalimentación estática (4.2) que lo desacopla.

2) Es capaz de determinar si un sistema no cuadrado es desacoplable por retroalimentación dinámica comprobando la condición (4.4). Si además se cumple que el interactor es diagonal, se asegura que el sistema también es desacoplable por retroalimentación estática.

3) Si el sistema no es desacoplable, es decir, no es diagonal para el caso cuadrado y no se cumple (4.4) para el caso no cuadrado, el programa *desacop* regresará cero como resultado.

El programa desacop se basa en otros programa de MATLAB, del Polynomial Toolbox y claro en el programa interactor.

Para presentar este algoritmo se va a dividir el problema en dos partes, la prueba de desacoplamiento que determina si el sistema tratado es desacoplable y la obtención de la retroalimentación para el caso regular estático.

4.2.1 La prueba de desacoplamiento.

Para saber si $\Phi(s)$ es diagonal, simplemente se obtiene el interactor y con funciones de MATLAB se observa si es efectivamente diagonal. Para probar la condición (4.4), el problema es como obtener el rango al infinito de $\Phi(s)$.

Para resolver este problema consideremos lo siguiente: $\Phi(s)$ es siempre polinomial ya que $T(s)$ es racional propia, por lo tanto, los vectores columna

$$\phi_j = \lim_{s\to\infty} \Phi_j(s)s^{-r_j}$$

donde $\Phi_j(s)$ es la $j-ésima$ columna de $\Phi(s)$, serán constantes y distintos de cero cuando r_j sea igual a la mayor potencia en s de la columna $\Phi_j(s)$, es decir, al grado por columna de $\Phi_j(s)$. Ahora, si tomamos r_j igual al grado por columna de $\Phi_j(s)$, entonces se puede ver como ϕ_j será igual al vector de coeficientes líderes de $\Phi_j(s)$.

Asi pues, el rango al infinito de $\Phi(s)$ se puede obtener como el rango de la matriz

$$\Phi_0 = [\ \phi_1 \qquad\qquad \phi_p\ ]$$

donde

$$\phi_j = g_j^{r_j}$$

con

$$\Phi_j(s) = g_j^{r_j} s^{r_j} + g_j^{r_j-1} s^{r_j-1} + ... + g_j^0$$

Resolvamos el Ejemplo 4.3 aplicando este algoritmo.

Ejemplo 4.3 *Tomemos el sistema*

$$T(s) = \begin{bmatrix} s^{-1} & 0 & 0 & s^{-2} & 0 \\ 0 & s^{-1} & 0 & s^{-3} & 0 \\ -s^{-1} & -s^{-1} & s^{-2} & -s^{-2} - s^{-3} & s^{-3} \end{bmatrix}$$

cuyo interactor es,

$$\Phi(s) = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ s^2 & s^2 & s^2 \end{bmatrix}$$

Ahora, según el algoritmo anterior,

$$\Phi_1(s) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} s^2 + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} s$$

$$\Phi_2(s) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} s^2 + \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} s$$

$$\Phi_3(s) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} s^2$$

por lo tanto,

$$\Phi_0 = [\ \phi_1 \qquad\qquad \phi_p\ ] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

4.2.2 Obtención de la retroalimentación.

Como se ha dicho, el programa *desacop* puede determinar únicamente la retroalimentación regular estática. Encontrar esta retroalimentación se reduce a encontrar la solución de

$$XD(s) + YN_1(s) = \Phi(s)T(s)D(s)$$

donde,

$$(sI - A)^{-1}B = N_1(s)D^{-1}(s)$$

Para solucionar esta ecuación, se puede utilizar la función *xaybc* del Polynomial Toolbox, donde obviamente,

$$a = D(s), \quad b = N_1(s) \quad \text{y} \quad c = \Phi(s)T(s)D(s)$$

Sin embargo, como se puede ver, para encontrar c es necesaria la multiplicación entre matrices polinomiales, como $D(s)$ y $\Phi(s)$, con $T(s)$ que es racional. Esta multiplicación no se puede realizar ya que, como se ha mencionado, MATLAB no puede manejar directamente las matrices racionales.

Para resolver este problema, consideremos la factorización

$$C(sI - A)^{-1}B = N(s)D^{-1}(s) = CN_1(s)D^{-1}(s) = T(s) \tag{4.5}$$

de este modo,

$$c = \Phi(s)T(s)D(s) = \Phi(s)CN_1(s)D^{-1}(s)D(s) = \Phi(s)CN_1(s)$$

ya puede ser obtenido fácilmente.

Por último, resolvamos el Ejemplo 4.1 aplicando este algoritmo.

Ejemplo 4.4 *Desacoplemos el mismo sistema* (A, B, C), *donde*

$$A = \begin{bmatrix} 2 & 8 & 0 & 6 & 12 \\ 1 & 0 & 0 & 0 & 0 \\ -25 & -50 & -10 & -53 & -74 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & -5 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 6 & 0 & 1 & 8 \\ 1 & -9 & 0 & 0 & -12 \end{bmatrix}$$

cuya matriz de transferencia es:

$$T(s) = \begin{bmatrix} \frac{1}{(s+1)^2} & \frac{1}{(s+1)(s+2)} \\ \frac{-6}{(s+1)(s+2)^2} & \frac{s-3}{(s+2)^2} \end{bmatrix}$$

y cuyo interactor es

$$\Phi(s) = \begin{bmatrix} s^2 & 0 \\ 0 & s \end{bmatrix}$$

Para obtener la retroalimentación, tomemos la factorización,

$$(sI - A)^{-1}B = N_1(s)D^{-1}(s) = \begin{bmatrix} 0 & s \\ 0 & 1 \\ s^2 & 0 \\ s & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 14 + 23s + 10s^2 + s^3 & 10 + 15s + 5s^2 \\ -12 - 6s & -8 - 2s + s^2 \end{bmatrix}^{-1}$$

De este modo, la factorización del sistema completo es, según (4.5),

$$C(sI - A)^{-1}B = CN_1(s)D^{-1}(s) = \begin{bmatrix} 8 + s & 6 \\ -12 & s - 9 \end{bmatrix} \begin{bmatrix} 14 + 23s + 10s^2 + s^3 & 10 + 15s + 5s^2 \\ -12 - 6s & -8 - 2s + s^2 \end{bmatrix}$$

Con estas matrices, podemos escribir la ecuación polinomial como

$$XD(s) + YN_1(s) = \Phi(s)T(s)D(s) = \Phi(s)CN_1(s)$$

donde,

$$\Phi(s)CN_1(s) = \begin{bmatrix} s^2(8 + s) & 6s^2 \\ -12s & s(s - 9) \end{bmatrix}$$

puede ser fácilmente obtenido por MATLAB ya que es una multiplicación de solamente matrices polinomiales.

Ahora, ya que

$$\Phi(s)CN_1(s) = \Phi(s)T(s)D(s)$$

Esta ecuación también tiene como una solución particular,

$$X = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad Y = \begin{bmatrix} -13 & -2 & -2 & -17 & -2 \\ -7 & 8 & 0 & -6 & 12 \end{bmatrix}$$

y por lo tanto, finalmente,

$$G = X^{-1} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

$$F = -GY = \begin{bmatrix} 6 & 10 & 2 & 11 & 14 \\ 7 & -8 & 0 & 6 & -12 \end{bmatrix}$$

4.2.3 Conclusiones de este algoritmo.

El algoritmo desacop, como se mencionó anteriormente, verifica las condiciones de desacoplamiento para sistemas lineales por retroalimentación estática o dinámica, regular y no regular. En el caso de sistemas cuadrados desacoplables, también calcula una retroalimentación que desacopla al sistema.

Por lo que respecta al cálculo de la retroalimentación desacoplante, observe que hablar de una solución particular de (4.3) significa que puede haber más de una de ellas, es decir, la retroalimentación que

desacopla al sistema no es única y la forma final de esta depende de como sea obtenida la solución particular X y Y

A pesar de la facilidad con que puede desacoplarse un sistema con este procedimiento, este enfoque tiene algunas desventajas que es importante mencionar.

En primer lugar, se debe notar que la retroalimentación obtenida con este procedimiento da como resultado una función de transferencia, en lazo cerrado, que es igual a la inversa del interactor. Esta situación, desde el punto de vista de ubicación de polos, significa que la retroalimentación produce una cancelación de polos con ceros finitos (en caso de que los hubiera), mientras que los polos restantes son ubicados todos en $s = 0$. Esto significa que el sistema está desacoplado, sin embargo, no se puede asegurar su estabilidad ni tampoco que las salidas tengan algún comportamiento deseado. Para lograr esto, es necesaria la aplicación de una retroalimentación posterior que permita ubicar estos polos de una manera adecuada y sin perder la estructura de sistema desacoplado que se consiguió con la primera retroalimentación. El problema de obtener un sistema desacoplado e internamente estable se conoce como desacoplamiento con estabilidad, pero su estudio está fuera de los alcances de este trabajo. Para revisar este tema ver, por ejemplo, [24, 29].

Otra problema de este procedimiento para desacoplar un sistema, sobre todo si se pretende aplicarlo al desacoplamiento de un sistema real, es el de la robustez. Este es un inconveniente propio de la falta de resultados en esta area, ya que hasta donde sabemos, el problema de desacoplamiento robusto no ha sido resuelto. Los resultados presentados sobre retroalimentación estática no son robustos en el sentido de que, por ejemplo, las condiciones de desacoplamiento para un sistema que era desacoplable, pueden no cumplirse más si se presenta una pequeña variación en alguno de sus parámetros.

Como ilustración considere el siguiente ejemplo.

Ejemplo 4.5 *Consideremos el siguiente sistema* (A, B, C) *con matrices,*

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & \varepsilon \\ 1 & 1 \\ 0 & 1 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Si ε *es igual a cero, el interactor de este sistema es*

$$\Phi_1(s) = \begin{bmatrix} s^2 & 0 \\ 0 & s \end{bmatrix}$$

que es diagonal y por lo tanto, el sistema es desacoplable.

Ahora, si $\varepsilon \neq 0$, *por ejemplo* $\varepsilon = 0.001$, *el interactor de este sistema es*

$$\Phi_2(s) = \begin{bmatrix} s & 0 \\ -1000s^2 & s^2 \end{bmatrix}$$

y entonces la condición de desacoplamiento ya no se cumple.

Estas desventajas son, naturalmente, heredadas al programa *desacop*, sin embargo, este primer algoritmo confiable puede ser base de algoritmos futuros que resuelvan el problema incluyendo la estabilidad y una ubicación deseada de polos. Por lo anterior, podemos afirmar que el algoritmo puede ser utilizado para estudiar el problema de desacoplamiento de sistemas lineales multivariables, pero hay que considerar algunos aspectos de estabilidad y robustez si se quisiera aplicar este algoritmo al control de sistemas reales.

Capítulo 5

Obtención de la estructura al infinito.

En este capítulo se presenta el algoritmo desarrollado para obtener la estructura al infinito de una matriz de rango pleno racional en general, es decir, no necesariamente polinomial o racional propia.

La estructura al infinito de una matriz racional comprende el conjunto de los órdenes de los polos y ceros al infinito de la matriz. Esta información se puede definir, como se mostrará más adelante, a partir de la forma de Smith McMillan al infinito de la matriz, que es básicamente una forma canónica bajo equivalencia con matrices bipropias. De lo anterior, es evidente que una forma de obtener la estructura al infinito de una matriz es mediante operaciones elementales por filas y por columnas en el anillo $\Re_p(s)$ sobre la matriz. Como se mencionó anteriormente en el cálculo del interactor de un sistema lineal, dichas operaciones no son numéricamente estables, por lo que no pueden ser usadas para desarrollar un algoritmo numérico confiable.

Nuestra intención es entonces, desarrollar un método confiable y el algoritmo correspondiente para obtener los polos y los ceros al infinito de una matriz racional. La idea es muy simple y está relacionada también con el método presentado en el Capítulo 3 para obtener el interactor de un sistema lineal multivariable.

Primeramente se define la estructura al infinito de una matriz racional, mencionando brevemente la importancia y posible interpretación que tiene esta información. Después se mostrarán los resultados que relacionan al interactor con esta estructura al infinito y finalmente se presenta el algoritmo desarrollado.

5.1 La estructura al infinito de matrices racionales.

La estructura al infinito de una matriz racional se puede definir como sigue.

Definición 5.1 *Sea $A(s) \in \Re^{p\times m}(s)$ una matriz racional de rango r. Entonces existen matrices bipropias $B_1(s)$ y $B_2(s)$, y una matriz única,*

$$B_1(s)A(s)B_2(s) = M_\infty(s) = \begin{bmatrix} s^{n_1} & & & \\ & & & 0 \\ & & & \\ & & s^{n_r} & \\ & 0 & & 0 \end{bmatrix} \tag{5.1}$$

conocida como la forma de Smith McMillan al infinito de $A(s)$ o la forma de Smith sobre el anillo $\Re_p(s)$ de $A(s)$, donde $n_i \geq n_{i+1}$, $i = 1, 2, ..., r-1$ son enteros positivos, negativos o cero.

La matriz $M_\infty(s)$ despliega la estructura al infinito de $A(s)$, la cual puede tener en general tanto polos como ceros al infinito de distintos órdenes. Si $n_i < 0$, entonces n_i es el orden de un cero al infinito de $A(s)$, si $n_i > 0$, entonces n_i es el orden de un polo al infinito. Evidentemente, las matrices $B_1(s)$ y $B_2(s)$ representan las operaciones elementales en $\Re_p(s)$ por filas y por columnas, respectivamente, que se tuvieron que aplicar a $A(s)$ para llevarla a la forma $M_\infty(s)$.

Si en particular, la matriz estudiada es racional propia, es decir, puede ser la matriz de transferencia que representa a un sistema lineal multivariable, entonces la estructura al infinito tendrá solamente ceros al infinito. En este caso, el orden de estos ceros al infinito se puede interpretar como el número de veces que se debe derivar cada una de las salidas para obtener un componente del vector de entradas. Esta información es de gran importancia en el estudio de algunos problemas de control como son: desacoplamiento, seguimiento de modelo, rechazo de perturbaciones, etc.

Por otro lado, la estructura al infinito de matrices polinomiales también es de gran importancia para la teoría de control [15, 2, 10]. Por ejemplo, en caso de que el sistema lineal sea descrito en espacio de estado, es decir por las matrices (A, B, C), los ceros al infinito del sistema, corresponden también a los ceros al infinito de la matriz polinomial

$$\begin{bmatrix} sI - A & B \\ C & 0 \end{bmatrix}$$

Son muchas las aplicaciones que puede tener el encontrar la estructura al infinito de una matriz racional en general, de ahí la importancia de desarrollar este algoritmo. Cabe mencionar que ya existen otros algoritmos confiables desarrollados para obtener la estructura al infinito de matrices polinomiales como por ejemplo el desarrollado en [10]. Este algoritmo está basado en matrices de Toeplitz formadas con las matrices de coeficientes de la matriz analizada. Otro algoritmo es el presentado en [32], el cual presenta una técnica en espacio de estado basada en la correspondencia entre la forma de Smith McMillan de la matriz analizada y la forma canónica de Kronecker de un pencil de matrices asociado.

Existen algunos otros algoritmos que obtienen la estructura al infinito de matrices racionales propias pero, por sus características no pueden considerarse algoritmos confiables. Un ejemplo de estos es el algoritmo desarrollado en [21], el cual está basado en los grados relativos de todos los menores de la matriz analizada. Algunos algoritmos simbólicos como por ejemplo el presentado en [6] también han sido desarrollados.

Los procedimientos utilizados en esta serie de algoritmos, son fundamentalmente diferentes del propuesto en nuestro algoritmo, y la comparación entre estos está fuera de los alcances de este trabajo.

La importancia de desarrollar este algoritmo para obtener la estructura al infinito de una matriz racional, es mostrar una aplicación más de la matriz interactor y lo importante que puede ser tener una forma para obtenerla. Además, nuestro algoritmo, como veremos más adelante, tiene la ventaja de que puede trabajar igualmente con matrices polinomiales, matrices racionales propias o matrices con algunos elementos como funciones racionales propias y otros como polinomios.

Antes de presentar el algoritmo desarrollado, se presentan algunos resultados importantes que involucran al interactor y que por lo tanto son base de este algoritmo. Es importante mencionar que el término interactor está definido, como se vió en el Capítulo 3, únicamente para sistemas lineales, es decir, cuando $T(s)$ es racional propia y es la representación matemática de algun sistema físico con entradas y salidas.

En este capítulo consideraremos que la matriz a analizar es cualquier matriz racional $A(s)$, no necesariamente propia ni tampoco necesariamente relacionada con algún sistema lineal. Por lo tanto, hablar de su interactor no sería estrictamente correcto, sin embargo, como veremos más adelante, la relación entre la forma de Hermite por columnas de $A(s)$ y su inversa $\Phi(s)$ se puede establecer sin importar si $A(s)$ es racional o no. Entonces por analogía, durante el desarrollo de este capítulo, seguiremos llamando a esta inversa $\Phi(s)$ la matriz interactor de $A(s)$.

Comencemos con la definición de una matriz reducida por filas.

Definición 5.2 *Sea $A(s)$ una matriz polinomial no singular de $p \times p$, y sea $\{q_i\}$, con*

$$q_i = \max\{\deg a_{ij}(s),\ j = 1, 2, ..., p\},\quad i = 1, 2, ..., p$$

el conjunto de los grados por filas de $A(s)$. Se dice que $A(s)$ es reducida por filas si

$$\deg\{\det A(s)\} = \sum_{i=1}^{p} q_i$$

Ahora, para explicar como es que el interactor de una matriz racional nos ayuda a encontrar su estructura al infinito, se considera, primeramente, una matriz $A(s)$ racional propia y cuadrada.

De la Definición 3.5, sabemos que la forma de Hermite por columnas de $A(s)$ es

$$A(s)B(s) = H(s)$$

donde $B(s)$ es bipropia. Ahora, del Lema 3.2 sabemos también que

$$H^{-1}(s) = \Phi(s) = \begin{bmatrix} s^{f_1} & & \\ & 0 & \\ & & \\ \phi_{p1} & & s^{f_p} \end{bmatrix}$$

Ahora, supongamos que la matriz polinomial $\Phi(s)$ es reducida por filas, esto significa que, según la Definición 5.2, el grado de cualquier elemento ϕ_{ij} es menor o igual que el grado del elemento en la diagonal de esa fila, es decir, ϕ_{ii}. Esta relación de grados implica que, para cualquier $i < j$, ϕ_{ij} sea divisible por ϕ_{ii} en el anillo $\Re_p(s)$ en el sentido de que existe una función racional propia $x(s) \in \Re_p(s)$ tal que,

$$\phi_{ii} = \phi_{ij} x(s)$$

Asi pues, es posible aplicar una serie de operaciones elementales por columnas de tipo 3 en $\Re_p(s)$, de modo que se llegue a una forma diagonal de $\Phi(s)$, es decir,

$$\Phi(s)\bar{B}(s) = diag\left\{s^{f_1}, ..., s^{f_p}\right\} = P(s)$$

donde $\bar{B}(s)$ representa las operaciones elementales en $\Re_p(s)$ aplicadas. Con esto podemos ver que,

$$A(s)B(s) = \Phi^{-1}(s) = \left(P(s)\bar{B}^{-1}(s)\right)^{-1} = \bar{B}(s)P^{-1}(s) \tag{5.2}$$

finalmente,

$$\bar{B}^{-1}(s)A(s)B(s) = P^{-1}(s) \tag{5.3}$$

Ahora, si se cumple que $f_i \geq f_{i+1}$, $i = 1, 2, ..., p-1$ (si no se cumple desde el principio, se puede lograr haciendo intercambio de filas), y haciendo $\bar{B}^{-1}(s) = B_1(s)$, $B(s) = B_2(s)$, y $P^{-1}(s) = M_\infty(s)$, esta última expresión (5.3) es equivalente a la definición de la forma de Smith McMillan (5.1), en otras palabras podemos decir que, si el interactor de $A(s)$ es reducido por filas, entonces los órdenes de los ceros al infinito están sobre la diagonal del interactor.

Con base en este procedimiento podemos formular el siguiente resultado, base de nuestro algoritmo.

Lema 5.1 *Sea $A(s) \in \Re_p^{p\times p}(s)$ una matriz racional propia de rango pleno y sea $\Phi(s) \in \Re^{p\times p}(s)$ su matriz polinomial interactor. Si $\Phi(s)$ es reducido por filas, entonces, los órdenes de los ceros al infinito de $A(s)$ corresponden a los enteros $\deg \phi_{ii}(s)$ donde $\phi_{ii}(s)$ son los elementos en la diagonal de $\Phi(s)$.*

Si el interactor de $A(s)$ no es reducido por filas, siempre existe una matriz de permutación P que equivale a un cambio de filas en $A(s)$ (y que por lo tanto no modifica la estructura al infinito de $A(s)$), tal que el interactor de $PA(s)$ es reducido por filas [11]. La forma en la que se debe hacer este cambio de filas se presenta más adelante.

Para el caso de que $A(s)$ tenga más columnas que filas, se puede ver fácilmente que el mismo resultado que acabamos de describir también puede aplicarse, ya que, como se formuló en el Lema 3.3, el interactor de $A(s)$ también viene de la inversa de su forma de Hermite por columnas. En el caso de que $A(s)$ tenga más filas que columnas y dado que la estructura al infinito de $A(s)$ es la misma que la de $A^T(s)$, se puede transponer la matriz y considerar que hay más columnas que filas, de esta forma, se puede aplicar también el mismo resultado.

Ahora bien, supongamos el caso de que $A(s)$ sea racional en general, es decir, no necesariamente racional propia o polinomial. Para este caso se puede encontrar el entero positivo más pequeño d tal que $\frac{1}{s^d}A(s) = A'(s)$ sea racional propia y entonces el Lema 5.1 sea aplicable. Haciendo esto, y suponiendo que el interactor de $A'(s)$ sea reducido por filas, entonces las expresiones equivalentes de 5.2 y 5.3 serían

$$A'(s)B(s) = \Phi^{-1}(s) = \left(P(s)\bar{B}^{-1}(s)\right)^{-1} = \bar{B}(s)P^{-1}(s)$$

$$\bar{B}^{-1}(s)A'(s)B(s) = P^{-1}(s)$$

y los órdenes n_i de los polos y ceros al infinito de la matriz original $A(s)$ serían

$$n_i = -f_i + d$$

donde f_i, en este caso, son los órdenes de los ceros al infinito de la matriz racional $A'(s)$.

A continuación se reformula el lema 5.1 de una manera general.

Lema 5.2 *Sea $A(s) \in \Re^{p\times m}(s)$ una matriz racional en general de rango pleno por filas, sea d el número entero positivo más pequeño tal que $A'(s) = \frac{1}{s^d}A(s)$ es racional propia y sea $\Phi(s) \in \Re^{p\times p}(s)$ la matriz polinomial interactor de $A'(s)$. Si el interactor*

$$\Phi(s) = \begin{bmatrix} s^{f_1} & & \\ & 0 & \\ & & \\ \phi_{p1} & & s^{f_p} \end{bmatrix}$$

es reducido por filas, entonces, los órdenes n_i de los ceros y polos al infinito de $A(s)$ son los enteros

$$n_i = -f_i + d \tag{5.4}$$

Observe que aunque la estructura al infinito está definida (Definición 5.1) para cualquier matriz de rango r, en nuestro algoritmo y dado a que está basado en el algoritmo *interactor* del Capítulo 3, solo consideraremos matrices de rango pleno. Para poder considerar el caso de que la matriz analizada no fuera de rango pleno, se debería de empezar por la modificación del algoritmo *interactor*. Esta modificación no es muy complicada y consiste en poner la fila Φ_i del interactor igual a cero cuando no se logre después de un número n de veces la independencia lineal entre las filas $K_1, K_2, ..., K_i$. Este número n tiene una cota superior (ver [34]).

Obtengamos ahora la estructura al infinito de algunas matrices para ejemplificar estos resultados.

Ejemplo 5.1 *Obtengamos la estructura al infinito de la matriz racional propia,*

$$T(s) = \begin{bmatrix} \frac{1}{s+1} & \frac{1}{s+2} \\ \frac{1}{s+3} & \frac{1}{s+4} \end{bmatrix}$$

Primeramente utilicemos operaciones elementales en $\Re_p(s)$ por columnas.
Multipliquemos la columna 1 por la función bipropia $\frac{s+1}{s}$, es decir, con $c_1\left(\frac{s+1}{s}\right) \rightarrow c_1$ tenemos,

$$T(s)B_1(s) = \begin{bmatrix} \frac{1}{s} & \frac{1}{s+2} \\ \frac{s+1}{s(s+3)} & \frac{1}{s+4} \end{bmatrix}$$

Ahora, con $c_2 - \frac{s}{s+2}c_1 \rightarrow c_2$ tenemos,

$$T(s)B_1(s)B_2(s) = \begin{bmatrix} \frac{1}{s} & 0 \\ \frac{s+1}{s(s+3)} & \frac{2}{(s+2)(s+3)(s+4)} \end{bmatrix}$$

Después, con $c_2\frac{(s+2)(s+3)(s+4)}{2s^3} \rightarrow c_2$ tenemos,

$$T(s)B_1(s)B_2(s)B_3(s) = \begin{bmatrix} \frac{1}{s} & 0 \\ \frac{s+1}{s(s+3)} & \frac{1}{s^3} \end{bmatrix}$$

Finalmente, con $c_1 - \frac{6s}{s+3}c_2 \rightarrow c_1$ tenemos,

$$H(s) = T(s)B_1(s)B_2(s)B_3(s)B_4(s) = \begin{bmatrix} \frac{1}{s} & 0 \\ \frac{s-2}{s^2} & \frac{1}{s^3} \end{bmatrix}$$

donde $H(s)$ sería la forma de Hermite por columnas y las matrices $B_i(s)$ las matrices elementales en $\Re_p(s)$ correspondientes.

Ahora apliquemos operaciones elementales en $\Re_p(s)$ por filas para llegar a la forma de Smith McMillan (5.1).

Sumando a la segunda fila $\left(-\frac{s-2}{s}\right)$ veces la fila 1, es decir, $r_2 - \frac{s-2}{s}r_1 \rightarrow r_2$, tenemos,

$$M_\infty(s) = \begin{bmatrix} \frac{1}{s} & 0 \\ 0 & \frac{1}{s^3} \end{bmatrix}$$

De esta forma, podemos ver que los órdenes de los ceros al infinito son $n_1 = 1$ y $n_2 = 3$.

Ahora tomemos el interactor,

$$\Phi(s) = H^{-1}(s) = \begin{bmatrix} s & 0 \\ 2s^2 - s^3 & s^3 \end{bmatrix}$$

el cual es evidentemente reducido por filas, por lo tanto, los órdenes de los ceros al infinito de $T(s)$ están sobre la diagonal, es decir, efectivamente $n_1 = 1$ y $n_2 = 3$.

Ejemplo 5.2 *Obtengamos la estructura al infinito de la matriz racional,*

$$A(s) = \begin{bmatrix} s^3 & 1 & 0 & 0 \\ 0 & \frac{1}{s} & 0 & s^2 \\ 0 & 0 & \frac{1}{s^2} & 0 \end{bmatrix}$$

Siguiendo el Lema 5.2, tenemos que $d = 3$, entonces,

$$A'(s) = \frac{1}{s^3} \begin{bmatrix} s^3 & 1 & 0 & 0 \\ 0 & \frac{1}{s} & 0 & s^2 \\ 0 & 0 & \frac{1}{s^2} & 0 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{s^3} & 0 & 0 \\ 0 & \frac{1}{s^4} & 0 & \frac{1}{s} \\ 0 & 0 & \frac{1}{s^5} & 0 \end{bmatrix}$$

El interactor de $A'(s)$ está dado por,

$$\Phi(s) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s^5 \end{bmatrix}$$

el cual es evidentemente reducido por filas, por lo tanto, los órdenes de los ceros al infinito de $A'(s)$ son, $f_1 = 0$, $f_2 = 1$ y $f_3 = 5$.

Finalmente, según (5.4), $A(s)$ tiene dos polos al infinito de órdenes $n_1 = 3$ y $n_2 = 2$, y un cero al infinito de orden $n_3 = 2$.

5.2 El algoritmo structinf.

El algoritmo *structinf*, al igual que el algoritmo *desacop* presentado en el Capítulo 4, se basa en algunas funciones de MATLAB, en algunas manipulaciones para poder manejar las matrices racionales y claro en el algoritmo *interactor* presentado en el Capítulo 3.

La idea básica del algoritmo *structinf* se basa en el Lema 5.2 presentado en la sección anterior. A continuación se presenta este algoritmo.

Supongamos que se desea obtener la estructura al infinito de $A(s)$ que es racional en general, es decir, se necesita encontrar el número d tal que $A'(s) = \frac{1}{s^d} A(s)$ es racional propia. Posteriormente se ordenan las filas de $A'(s)$ de modo que los grados relativos por filas, es decir, la menor diferencia entre el grado de denominador menos el del numerador de cada elemento de la fila, vayan del menor en la primera fila al mayor en la última fila.

Al sacar el interactor de esta matriz ordenada, aseguramos que, dado que la primera fila del interactor es siempre

$$\Phi_1(s) = \begin{bmatrix} s^{f_1} & 0 & 0 \end{bmatrix}$$

este f_i sea efectivamente el menor de todos los órdenes de los ceros al infinito de $A'(s)$, de hecho, si el interactor resulta reducido por filas, entonces todos los órdenes de los ceros al infinito estarán sobre la diagonal. Ahora consideremos que el interactor no es reducido por filas, entonces, sabiendo que el orden f_1 es correcto, escribamos al interactor de la siguiente forma

$$\Phi(s) = \begin{bmatrix} s^{f_1} & 0 & & 0 \\ & & & \\ & & \bar{\Phi}(s) & \\ \phi_{p1} & & & \end{bmatrix}$$

y tomemos la inversa de las $p-1$ filas y columnas siguientes, es decir $\bar{\Phi}^{-1}(s)$. Ordenando los grados relativos de $\bar{\Phi}^{-1}(s)$ de menor a mayor y haciendo ese mismo cambio de filas sobre la matriz $A'(s)$ de la que sacamos el interactor, se logra entonces que, al sacar de nuevo el interactor de la matriz reordenada, ahora las dos primeras filas tengan en la diagonal los dos primeros órdenes de los ceros al infinito.

Haciendo este proceso iterativamente y un máximo de $p-1$ veces finalmente se llegará a que el interactor sea reducido por filas y los órdenes de los ceros al infinito estén sobre la diagonal del mismo. Finalmente, para recuperar el orden de los polos y ceros al infinito de la matriz original $A(s)$, al valor d se le resta los valores obtenidos hasta este punto.

Ahora, recordando que MATLAB sólo maneja matrices polinomiales, sabemos que al sacar el interactor de $A'(s)$ primeramente se transforma $A'(s)$ a una matriz polinomial $A''(s)$ multiplicándola por el producto $p(s)$ de todos los denominadores de sus elementos. Después se utilizarán los Lemas 3.6 y 3.6 para que, a partir del grado $\deg p(s)$ se recupere la información original de $A'(s)$ (ver Capítulo 3). Dada esta situación, resulta ineficiente que se transforme $A(s)$ a una matriz racional propia $A'(s)$ y después regresar a una matriz polinomial $A''(s)$.

Para evitar esta ineficiencia, el algoritmo sigue el procedimiento presentado en el siguiente ejemplo.

Ejemplo 5.3 *Obtengamos la estructura al infinito de la matriz racional,*

$$A(s) = \begin{bmatrix} s^3 & 1 & 0 & 0 \\ 0 & \frac{1}{s} & 0 & s^2 \\ 0 & 0 & \frac{1}{s^2} & 0 \end{bmatrix}$$

Siguiendo el Lema 5.2, sabemos que $A(s)$ debe ser multiplicada por $\frac{1}{d}$ donde $d = 3$, es decir,

$$A'(s) = \frac{1}{s^3}\begin{bmatrix} s^3 & 1 & 0 & 0 \\ 0 & \frac{1}{s} & 0 & s^2 \\ 0 & 0 & \frac{1}{s^2} & 0 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{s^3} & 0 & 0 \\ 0 & \frac{1}{s^4} & 0 & \frac{1}{s} \\ 0 & 0 & \frac{1}{s^5} & 0 \end{bmatrix}$$

y después, para poder aplicar el algoritmo interactor, $A'(s)$ debería ser pasada a una forma polinomial.

Sin embargo, en el algoritmo lo que se hace es pasar directamente $A(s)$ a una forma polinomial multiplicándola por:

$$p(s) = s^3$$

haciendo lo anterior tenemos,

$$p(s)A(s) = A''(s) = \begin{bmatrix} s^6 & s^3 & 0 & 0 \\ 0 & s^2 & 0 & s^5 \\ 0 & 0 & s & 0 \end{bmatrix}$$

Ahora se toma como $\deg p(s)$*, para la aplicación del algoritmo interactor (ver Capítulo 3, Lemas 3.5 y 3.6), el número* $g = 6$ *que es el mayor grado en* s *de la matriz* $A''(s)$ *y se obtiene el interactor reducido por filas,*

$$\Phi(s) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s^5 \end{bmatrix}$$

de lo que sería, por lo tanto, la matriz

$$\frac{1}{s^9}A''(s) = \frac{1}{s^6}A''(s) = \begin{bmatrix} 1 & \frac{1}{s^3} & 0 & 0 \\ 0 & \frac{1}{s^4} & 0 & \frac{1}{s} \\ 0 & 0 & \frac{1}{s^5} & 0 \end{bmatrix}$$

que es efectivamente la matriz $A'(s)$ *que necesitábamos.*

Ahora, según el Lema 5.2, los órdenes de los ceros y polos al infinito de $A(s)$ *son,*

$$n_i = -f_i + d$$

donde f_i *son las potencias en* s *de la diagonal de* $\Phi(s)$*, sin embargo el número* d *no lo conocemos ya que en realidad nunca obtuvimos* $A'(s)$*. A pesar de lo anterior, sabemos que* $g = 6$ *y que el verdadero valor de* $\deg p(s) = 3$*, por lo tanto, se puede ver fácilmente que,*

$$n_i = -f_i + g - \deg p(s) \tag{5.5}$$

es decir,

$$\begin{aligned} n_1 &= 0 + 6 - 3 = 3, \\ n_2 &= -1 + 6 - 3 = 2, \quad y \\ n_3 &= -5 + 6 - 3 = -2 \end{aligned}$$

lo cual corresponde con lo obtenido en el Ejemplo 5.2.

Ahora veamos un ejemplo en el que sea necesaria la reordenación de filas.

Ejemplo 5.4 *Obtengamos la estructura al infinito de la matriz racional,*

$$A(s) = \begin{bmatrix} \frac{1}{s} & \frac{1}{s^4} & \frac{1}{s} \\ \frac{1}{s} & 0 & \frac{1}{s} \\ \frac{1}{s^2} & \frac{1}{s^3} & \frac{s^2+s^5}{s^7} \end{bmatrix}$$

Primeramente, se pasa esta matriz a una forma polinomial multiplicándola por

$$p(s) = s^{20}$$

es decir,

$$A_1''(s) = \begin{bmatrix} s^{19} & s^{16} & s^{19} \\ s^{19} & 0 & s^{19} \\ s^{18} & s^{17} & s^{15} + s^{18} \end{bmatrix}$$

y se toma $g = 19$, *ahora, según el Lema 3.5 los grados por filas de* $A''(s)$ *serán 0 para la fila 1, 0 para la fila 2 y 1 para la fila 3. Es decir, las filas están ordenadas correctamente y se puede obtener el interactor de* $\frac{1}{s^{19}} A_1''(s)$ *que es,*

$$\Phi_1(s) = \begin{bmatrix} 1 & 0 & 0 \\ -s^3 & s^3 & 0 \\ -s^3 - s^5 & s^5 & s^4 \end{bmatrix}$$

Ahora, como se puede ver, este interactor no es reducido por filas, por lo tanto, se debe obtener la inversa de

$$\bar{\Phi}_1(s) = \begin{bmatrix} s^3 & 0 \\ -s^5 & s^4 \end{bmatrix}$$

que es,

$$\bar{\Phi}_1^{-1}(s) = \begin{bmatrix} \frac{1}{s^3} & 0 \\ \frac{1}{s^2} & \frac{1}{s^4} \end{bmatrix}$$

De esta inversa se ve que las filas 2 y 3 deben ser intercambiadas ya que el grado relativo de la fila 3 es menor que el de la fila 2. Entonces, se toma

$$A_2''(s) = \begin{bmatrix} s^{19} & s^{16} & s^{19} \\ s^{18} & s^{17} & s^{15} + s^{18} \\ s^{19} & 0 & s^{19} \end{bmatrix}$$

y se obtiene el interactor de $\frac{1}{s^{19}} A_1''(s)$ *que es,*

$$\Phi_2(s) = \begin{bmatrix} 1 & 0 & 0 \\ -s & s^2 & 0 \\ -s^3 - s^5 & s^4 & s^5 \end{bmatrix}$$

Finalmente, dado que $\Phi_2(s)$ *ya es reducido por filas, el algoritmo término. Ahora, dado que* $\deg p(s) = 20$ *y según (5.5) . los órdenes de los ceros al infinito de* $A(s)$ *son:*

$$\begin{aligned} n_1 &= 0 + 19 - 20 = -1 \\ n_2 &= -2 + 19 - 20 = -3 \\ n_3 &= -5 + 19 - 20 = -6 \end{aligned}$$

En resumen, este algoritmo es entonces un algoritmo confiable pero que tiene la desventaja de que para algunos casos en los que las matrices sean demasiado grandes, puede darse la situación de que sea necesario calcular varias veces (a lo más $p - 1$ veces) el interactor de una matriz. Por otro lado, este algoritmo permite trabajar indistintamente, y con el mismo procedimiento, matrices polinomiales, racionales propias o matrices racionales en general lo cual es una ventaja importante.

Conclusiones.

El actual desarrollo que el enfoque polinomial de la teoría de control está teniendo, ha abierto nuevos campos de investigación como lo es el desarrollo de algoritmos para trabajar con matrices polinomiales y resolver problemas de control desde este enfoque.

A pesar de la introducción de los lenguajes de matemáticas simbólicas, con los cuales el manejo de matrices y ecuaciones polinomiales se reduciría a considerar la variable compleja s como una variable no valuada, muchos investigadores trabajan por encontrar algoritmos numéricos que manejen este tipo de matrices. Un algoritmo numérico puede resultar más eficiente en términos de memoria y tiempo de ejecución que un algoritmo simbólico, además, aplicando los métodos adecuados estos algoritmos numéricos pueden lograr la precisión numérica obtenida simbólicamente. Esta posibilidad de lograr algoritmos eficientes además de la elegancia con que un problema, que involucra matrices polinomiales, puede ser resuelto al aplicar un algoritmo numérico hace que estos últimos sean muy atractivos para la investigación.

En el desarrollo de estos algoritmos numéricos confiables, aplicados al enfoque polinomial de la teoría de control, varios son los resultados que se han obtenido y que han permitido, entre otras cosas, la creación de The Polynomial Toolbox. En este trabajo se han desarrollado 3 algoritmos numéricos confiables programados sobre la plataforma que proporcionan MATLAB y el Polynomial Toolbox.

Uno de estos algoritmos, el algoritmo *interactor* presentado en el capítulo 3, es capaz de obtener la matriz interactor de un sistema lineal multivariable con matriz de transferencia de rango pleno, ya sea por filas o por columnas. El procedimiento para obtener este interactor sin recurrir a las operaciones elementales que, como mencionamos, se ha demostrado son numéricamente inestables, es una modificación al procedimiento presentado inicialmente en el artículo de Falb y Wolovich [34]. En este trabajo, se modificó en lo necesario este procedimiento para poder programarlo utilizando MATLAB y The Polynomial Toolbox. El algoritmo programado, basado únicamente en operaciones estables, es entonces un algoritmo confiable y, hasta donde sabemos, único. La importancia de este algoritmo es evidente cuando se estudian los problemas en los que el interactor forma parte importante. Algunos de estos problemas, como el desacoplamiento de sistemas lineales y la obtención de la estructura al infinito de matrices racionales, fueron tratados también en este trabajo, otros problemas quedan como posibilidad de trabajo futuro. Uno de estos problemas es el de la equivalencia de sistemas bajo compensación dinámica. Para este problema el interactor también forma parte importante de la solución, tanto para probar si los dos sistemas analizados son equivalentes bajo compensación dinámica como para encontrar el compensador en caso de que lo sean.

El algoritmo *desacop*, desarrollado para el desacoplamiento de sistemas lineales, se basó, obviamente en el algoritmo *interactor* y en otras funciones de MATLAB y del Polynomial Toolbox (ver Capítulo 4). Este algoritmo determina si el sistema lineal analizado el desacoplable mediante retroalimentación dinámica o estática. En el caso de sistemas cuadrados que sean desacoplables por retroalimentación estática, también encuentra una retroalimentación que logra el desacoplamiento. También se puede decir que este algoritmo es un algoritmo confiable, sin embargo, presenta algunos inconvenientes sobre todo si se piensa en la aplicación de este algoritmo para el desacoplamiento de un sistema real. Estos inconvenientes tienen que ver, como se explicó en el Capítulo 4, con la estabilidad del sistema retroalimentado y con la robustez de las condiciones de desacoplamiento. Se presentan también , por lo tanto, muchas opciones de

trabajo futuro tanto a nivel práctico, en la programación de algoritmos, como a nivel teórico. La solución completa del problema de desacoplamiento no regular estático, obteniendo las condiciones necesarias y suficientes para saber si un sistema es desacoplable, asi como la determinación de la ecuación polinomial que pudiera determinar las matrices F y G de la retroalimentación, es un ejemplo del trabajo futuro que se debe realizar. El desarrollo de un algoritmo que obtenga un sistema desacoplado y estable con alguna ubicación deseada de polos también es un trabajo importante que puede tener mucha aplicación, a nivel práctico, en la implementación de control a sistemas físicos.

El tercer y último algoritmo desarrollado fue el algoritmo *structinf*. Este algoritmo confiable, desarrollado también con base en el algoritmo *interactor* y en las funciones de MATLAB y del Polynomial Toolbox, obtiene la estructura al infinito de cualquier matriz racional. La idea básica del algoritmo implica, como vimos en el Capítulo 5, que en algunos casos sea necesario obtener el interactor en varias ocasiones lo cual puede representar un inconveniente para este algoritmo. Sin embargo, la capacidad del algoritmo de trabajar indistintamente cualquier matriz, sin importar que forma tengan sus elementos, es decir, sin importar si son polinomios, funciones racionales propias, o si estan combinados ambos tipos dentro de la misma matriz, es una característica interesante que puede ser una ventaja con respecto a otro métodos.

Por lo que respecta a este último algoritmo, existen ya algoritmos confiables que obtienen, por ejemplo, la estructura al infinito de matrices polinomiales [10,32]. Las técnicas empleadas en estos algoritmos son, sin embargo, fundamentalmente diferentes. La comparación cuantitativa de nuestro algoritmo con estos otros algoritmos existentes, quedó fuera de nuestros objetivos. Es importante mencionar que esta comparación, la cual implica necesariamente una forma de cuantificar la complejidad de los algoritmos, no solo es un trabajo práctico en el que se mide el tiempo de ejecución y el número de operaciones utilizadas, sino que requiere también de un cálculo analítico.

Unido a este cálculo analítico de la complejidad de un algoritmo, se debe considerar también la prueba formal de la estabilidad numérica del algoritmo. Todo esto se ha desarrollado poco y casi no se ha aplicado a los algoritmos que se han venido obteniendo. Por lo tanto, puede ser tema también de investigaciones y trabajos futuros.

Anexo 1. El funcionamiento de los programas.

A continuación se presentan algunos ejemplos (hechos en MATLAB) que muestran el funcionamiento de los programas desarrollados.

Para el programa interactor.

>>help interactor

INTERACTOR Interactor matrix of a Linear System

The command
interactor(num,den)
returns the polynomial matrix Interactor of the linear system
represented by the rational transfer matrix T(s) given by the
arrays NUM and DEN of the same dimensions, containing the
numerators and the denominators of T(s).
The Transfer matrix T(s) should have full rank.

The command
interactor(A,B,C)
returns the polynomial matrix Interactor of the linear system
represented in state space by (A,B,C).
The equivalent Transfer matrix T(s) should have full rank.

If the matrix T(s) has more rows than columns, this command also
returns a polynomial called dd, such that, the interactor of the
system is: E=ans*1/dd

» num=[1 1; 1 1]
» den=[s+1 s+2; s+3 s+4]
» interactor(num,den)

ans =

$$\begin{bmatrix} s & 0 \\ 2s^2 - s^3 & s^3 \end{bmatrix}$$

» A=[1 2 3; 0 1 2; 1 1 4]
» B=[1 2; 2 3; 0 1]
» C=[0 0 1; 1 1 0; 1 1 4]
» interactor(A,B,C)

Constant polynomial matrix: 1-by-1
dd =
1
ans =

$$\begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ -1 & -4 & 1 \end{bmatrix}$$

Para el programa desacop.

>>help desacop

DESACOP Feedback Decoupling of Linear Systems

The command
[F,G]=desacop(A,B,C)
returns the matrix F and G such that the feedback u=Fx+Gv decouples the square system represented by the matrix A, B and C. If the system is not decouplable, the command returns F=0, G=0

The command
[F,G]=desacop(num,den)
returns the matrix F and G such that the feedback u=Fx+Gv decouples the square system represented by the transfer matrix T(s) given by the arrays NUM and DEN of the same dimensions, containing the numerators and the denominators of T(s).
It also returns the state space representation (A,B,C) used in the feedback calculations. If the system is not decouplable, the command returns F=0, G=0

The closed loop system (A+BF,BG,C) has as a transfer matrix, the following one: diag{1/s^Ni} where {Ni} are the orders of the zeros at infinity.

If the system is decouplable but not square, this commands only returns an indicative message.

» A=[2 8 0 6 12; 1 0 0 0 0; -25 -50 -10 -53 -74; 0 0 1 0 0; 0 0 0 1 0];
» B=[0 1; 0 0; 1 -5; 0 0; 0 0];
» C=[0 6 0 1 8; 1 -9 0 0 -12];
[F,G]=desacop(A,B,C)

Constant polynomial matrix: 2-by-5
F =

$$\begin{bmatrix} 6 & 10 & 2 & 11 & 14 \\ 7 & -8 & 0 & 6 & -12 \end{bmatrix}$$

Constant polynomial matrix: 2-by-2
G =

$$\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

» num=[1 1; 1 1];
» den=[s+1 s+2; s+3 s+4];
[F,G]=desacop(num,den)
F =
0
G =
0

Para el programa structinf.

>>help structinf

STRUCTINF Infinite structure of a rational matrix

The command
structinf(num,den)
returns a vector with the powers of s of the diagonal of the Smith Mcmillan form at infinity of a rational matrix A(s) given by the arrays NUM and DEN of the same dimensions, containing the numerators and the denominators of A(s).
The matrix A(s) should have full rank

The command
structinf(num,1)
returns a vector with the powers of s of the diagonal of the Smith Mcmillan form at infinity of a polynomial matrix A(s)=num(s)
The matrix A(s) should have full rank

» num=[1 s+1 2; 1 2 3; 1 1 1];
» den=[s^1 1 s+1; s^2 s^3 s^4; s+1 s+1 1];
» structinf(num,den)
ans =
1
0
-2

» num=[s 1; 1 s+1; 0 s^4]
» structinf(num,1)
ans =
4
1

Bibliografía

[1] B. D. O. Anderson, "From Youla-Kučera to Identification, Adaptive and Nonlinear Control" *Proceedings of the IFAC World Congress*, pp. 39-59, San Francisco, California, 1996.

[2] C. T. Chen, *Linear System Theory and Design*, Holt, Rinehart and Winston, 1984.

[3] W. D. Cooper and A. D. Helfrick, *Instrumentación electrónica moderna y técnicas de medición*, Prentice Hall, 1991.

[4] J. M. Dion and C. Commault, "The Minimal Delay Decoupling Problem: Feedback Implementation with Stability", *SIAM J. Control and Optimization*, vol. 26, no. 1, pp. 66-82, 1988.

[5] G. H. Golub and C. F. Van Loan, *Matrix Computations*, The John Hopkins University Press, 1996.

[6] J. C. González, *Herramientas Computacionales para Análisis de Sistemas en Control Automático*, Tesis de Maestría, CINVESTAV-IPN, Unidad Guadalajara, 1998.

[7] S. Grossman, *Algebra Lineal*, McGraw Hill, 1996.

[8] D. Henrion, *Reliable Algorithms for Polynomial Matrices*, PhD. Thesis, Institute of Information Theory and Automation, Prague, Czech Republic, 1998.

[9] D. Henrion and M. Sebek, "Improved Polynomial Matrix Determinant Computations", *IEEE Transactions on Circuits and Systems*, vol. 46, no. 10, pp 1307-1308, 1999.

[10] D. Henrion, J. Ruiz León and M. Sebek, "Extraction of Infinite Zeros of Polynomial Matrices" *39th IEEE Conference on Decision and Control*, pp. 4221-4226, Sydney, Australia, December 2000.

[11] A. N. Herrera, "Structural Results About the Interactor", *Proceedings of the 29th Conference on Desicion and Control*, pp.1048-1049, Honolulu, Hawai, 1990.

[12] I. N. Herstein, *Algebra Moderna*, Editorial Trillas, 1990.

[13] M. Hromčík, *Numerical Algorithms for Polynomial Matrices*, PhD. Thesis, Czech Technical University, Prague, Czech Republic, 1999.

[14] K. J. Hunt and V. Kučera, "The Standard H_2-Optimal Control Problem: A Polynomial Solution", *International Journal of Control*, vol. 56, no. 1, pp. 245-251, 1992.

[15] T. Kailath, *Linear Systems*, Prentice Hall, 1980.

[16] F. Kraffer, *State-Space Algorithms for Polynomial Matrix Operations*, Publicaciones Técnicas del Departamento de Ingeniería Eléctrica, Serie Amarilla no. 35, CINVESTAV-IPN, 1998.

[17] F. Kraus and V. Kučera, "Control Assignment of Polynomial Matrix Polytopes", *Proceedings of the European Control Conference*, Brussels, Belgium, July 1997.

[18] V. Kučera, "Diophantine Equations in Control - a Survey", *Automatica*, vol. 29, no. 6, pp. 1361-1375, 1993.

[19] V. Kučera, *Discrete Linear Control, The Polynomial Equation Approach,* Wiley Chichester, 1979.

[20] V. Kučera, "Stationary LQG Control of Singular Systems", *IEEE Transactions on Automatic Control*, vol. 31, no. 1, pp 31-39, 1986.

[21] F. Lizaola and J. Ruiz León, "Cálculo de los Ceros al Infinito de Matrices Racionales Propias" *Quinta Conferencia de Ingeniería Eléctrica CIE-99,* pp. 15-17, México, D.F., 1999.

[22] Polyx Ltd, *The Polynomial Toolbox for MATLAB*, version 2.0, 1999. See www.polyx.cz

[23] Waterloo Maple Corporate, *MAPLE,* version 6.0. See www.mapleapps.com

[24] J. C. Martinez and M. Malabre, "The Row by Row Decoupling Problem with Stability: a Structural Approach", *IEEE Transactions on Automatic Control*, vol. 39, no. 12, pp 2457-2460, 1994.

[25] The MathWorks, *MATLAB,* version 6.0. See www.mathworks.com/products/matlab

[26] A. S. Morse, "System Invariants Under Feedback and Cascade Control" *Proceedings International Symposium*, pp. 61-74, Udine, Italy, 1975.

[27] P. Hr. Petkov, N. D. Christov, and M. M. Konstantinov, *Computational Methods for Linear Control Systems,* Prentice Hall, 1991.

[28] M. W. Rogozinski, A. P. Paplinski and M. J. Gibbard, "An Algorithm for the Calculation of a Nilpotent Interactor Matrix for Linear Multivariable Systems", *IEEE Transactions on Automatic Control*, vol. AC-32, no. 3, pp 234-237, 1987.

[29] J. Ruiz León, P. Zagalak and V. Eldem, "On the Problem of Decoupling", *Proceedings 3rd IFAC Conference on System Structure and Control*, pp. 611-616, Nantes, France, 1995.

[30] J. Ruiz León, J. C. Zuñiga and D. Henrion, "Computation of the Interactor of a Linear Multivariable System" accepted to *IASTED International Conference on Control and Applications 2001*, Banff, Canada, June 2001.

[31] G. W. Stewart, *Introduction to Matrix Computations,* Academic Press, 1973.

[32] P. Van Dooren, "The Computation of Kronecker's Canonical Form of a Singular Pencil" *Linear Algebra and its Applications,* vol. 27, pp. 103-141, 1979.

[33] M. Vidyasagar, *Control Systems Synthesis: a Factorization Approach,* MIT Press, 1985.

[34] W. A. Wolovich and P. L. Falb, "Invariants and Canonical Forms under Dynamic Compensation", *SIAM J. Control and Optimization*, vol. 14, no. 6, pp. 996-1008, 1976.

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
UNIDAD GUADALAJARA

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, aprobó la Tesis: **"Desarrollo de Algoritmos Numéricos para Teoría de Control"** del Sr. **Juan Carlos Zuñiga Anaya**, el día **20** de **Julio** de **2001**

EL JURADO

Dr. Bernardino Castillo Toledo
Investigador Cinvestav 3A
CINVESTAV DEL IPN
Guadalajara

Dra. Ofelia Begovich Mendoza
Investigador Cinvestav 3A
CINVESTAV DEL IPN
Guadalajara

Dr. Antonio Ramírez Treviño
Investigador Cinvestav 2A
CINVESTAV DEL IPN
Guadalajara

Dr. José Javier Ruiz León
Investigador Cinvestav 2A
CINVESTAV DEL IPN
Guadalajara