UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO

INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA

LICENCIATURA EN INGENIERÍA EN ELECTRÓNICA

CÓMPUTO DE LA MATRIZ DE LYAPUNOV TIPO RETARDADA EN UNA TARJETA DE DESARROLLO DE BAJO COSTO

T E S I S

QUE PARA OBTENER EL TÍTULO DE LICENCIADO EN INGENIERÍA EN ELECTRÓNICA

PRESENTA:

ALEJANDRO GUTIÉRREZ LUNA

DIRECTOR DE TESIS:

DR. CARLOS CUVAS CASTILLO

PACHUCA HGO., MÉXICO, NOVIEMBRE DE 2025

Universidad Autónoma del Estado de Hidalgo
Instituto de Ciencias Básicas e Ingeniería
School of Engineering and Basic Sciences

Mineral de la Reforma, Hgo., a 27 de octubre de 2025

Número de control: ICBI-D/1861/2025
Asunto: Autorización de impresión.

MTRA. OJUKY DEL ROCÍO ISLAS MALDONADO
DIRECTORA DE ADMINISTRACIÓN ESCOLAR DE LA UAEH

Con Título Quinto, Capítulo II, Capítulo V, Artículo 51 Fracción IX del Estatuto General de nuestra Institución, por este medio, le comunico que el Jurado asignado a la egresada de la Licenciatura en Ingeniería en Electrónica **Alejandro Gutiérrez Luna,** quien presenta el trabajo de titulación **"CÓMPUTO DE LA MATRIZ DE LYAPUNOV TIPO RETARDADA EN UNA TARJETA DE DESARROLLO DE BAJO COSTO",** ha decidido, después de revisar fundamento en lo dispuesto en el Título Tercero, Capítulo I, Artículo 18 Fracción IV; dicho trabajo en la reunión de sinodales, **autorizar la impresión del mismo**, una vez realizadas las correcciones acordadas.

A continuación, firman de conformidad los integrantes del Jurado:

Presidente: Ing. Benigno Martínez Gutiérrez

Secretario: Dr. Francisco Germán Mejía Hernández

Vocal: Dr. Carlos Cuvas Castillo

Suplente: Dr. Alejandro Castaño Hernández

Sin otro particular por el momento, reciba un cordial saludo.

Atentamente
"Amor, Orden y Progreso"

Mtro. Gabriel Vergara Rodríguez
Director del ICBI

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO ICBI DIRECCIÓN

GVR/YCC

Ciudad del Conocimiento, Carretera Pachuca-Tulancingo Km. 4.5 Colonia Carboneras, Mineral de la Reforma, Hidalgo, México. C.P. 42184
Teléfono: 771 71 720 00 Ext. 40001
direccion_icbi@uaeh.edu.mx, vergarar@uaeh.edu.mx

Dedicatoria

A mis padres, porque este no es solo un logro académico mío, es un reflejo de todo el tiempo, esfuerzo y dinero que han invertido en mi educación. Cada sacrificio realizado por ustedes forjó el camino que hoy permitió este éxito. Esta tesis es un reflejo de ello y me llena de orgullo poder corresponderles y honrarlos de esta manera. Gracias por ser los pilares de mi vida y enseñarme el valor del trabajo duro; sin más que decir. Los amo.

Agradecimientos

A mis padres, Alejandro Gutiérrez Lobato y Marisol Luna González, gracias por apoyarme, escucharme y motivarme día con día, y corregirme cuando es necesario, no hay palabras para agradecer todos los cansancios, desvelos y dolores que han soportado y, si hoy soy lo que soy, es gracias a ustedes. No puedo estar más orgulloso de ser su hijo, los amo profundamente.

A mi hermana, Abigail Gutiérrez Luna, tú eres la persona con la que más he convivido en toda mi vida, eres mi mejor amiga, confidente, mi cómplice y algunas veces enemiga, pero te agradezco que me escuchas quejarme, me das consejos, apoyo y compañía. Cuentas conmigo siempre. Te amo, bis-bis.

A mi novia, Danna Elizabeth Cabrera Garcia, gracias por tenerme paciencia, y aunque a veces pueda perderme en mi mundo, no dejo de pensarte. Gracias por darme ánimos en los momentos en que me veía superado, y escucharme aunque a veces no tengas idea de qué estoy hablando, incluso cuando mis explicaciones parecían más una exposición que una conversación. Tú has sido mi pilar emocional en muchos momentos y lo seguirás siendo en los que vengan, te amo.

A mis amigos, Israel Naranjo Huautla, José Gerardo Hernández Gómez, David Gutiérrez Ramírez y José Alfredo Alvarado Cortez, gracias por todas las experiencias que vivimos. Mi estancia en la universidad no habría sido tan divertida sin ustedes, todas las anécdotas y experiencias las llevo conmigo. Gracias por ser mis amigos y estar conmigo en cada trabajo, proyecto, materia y desvelo. Les deseo mucho éxito en el camino que tenemos por delante.

Al Dr. Carlos Cuvas Castillo, por permitirme trabajar con usted y por toda la paciencia que ha tenido conmigo, por compartir su conocimiento y guiarme cuando me perdía, todos sus consejos y motivaciones han sido muy valiosos para mí.

A mis sinodales, el Dr. Francisco Germán Mejía Hernández, el Ing. Benigno Martínez Gutiérrez y al Dr. Alejandro Castaño Hernández, por sus comentarios, explicaciones y guía para dar forma a la presente tesis.

A Aracely Gutiérrez Luna, que en paz descanse, por haber sido el más claro ejemplo de no dejarse vencer y rendirse ante las dificultades, por tener la mejor actitud incluso en los momentos más complicados. Gracias, maestra Chely, por enseñarme a no rendirme nunca.

A Beily, Eevee, Tobby y Loona, porque más que mascotas, son mi familia, sin palabras, solo con su compañía, me calman incluso cuando me veo superado por el estrés, y este trabajo es prueba del apoyo que representan.

Resumen

El presente trabajo de investigación surge de la observación de la falta de herramientas accesibles en áreas de la ingeniería, específicamente en el área de control, con los sistemas lineales con un único retardo puntual, en los que el análisis de estabilidad es importante, pues, de no evaluarla, se pueden omitir problemas de control que deriven en inestabilidad, pérdidas de desempeño y otros problemas indeseados.

Una de las maneras utilizadas para evaluar la estabilidad en dichos sistemas es un criterio basado en la matriz de Lyapunov tipo retardada; sin embargo, dicha herramienta está disponible en sistemas de alto desempeño y hardware especializado, que suele ser caro. Por esta razón, se desarrolló su cómputo, en la tarjeta de desarrollo FRDM-KL46Z.

Para dicho procedimiento, considerando las limitaciones que presenta la tarjeta, se requirió el desarrollo de cada una de las operaciones especificadas del punto 2.6.1 al 2.6.14.1 de la presente tesis, que son requeridas para el cálculo de la matriz de Lyapunov tipo retardada, utilizando para ello el lenguaje de programación C mediante el entorno de desarrollo MCUXpresso IDE.

Además, utilizando los softwares CoolTerm y MATLAB, se graficaron los resultados, lo que permitió verificar que los cálculos realizados fueran correctos.

Abstract

The present research work arises from the observation of the lack of accessible tools in areas of engineering, specifically in the field of control, for linear systems with a single point delay, in which stability analysis is important since, if not evaluated, problems that lead to instability, performance losses, and other undesirable issues may be overlooked.

One of the methods used to evaluate stability in such systems is a criterion based on the delayed-type Lyapunov matrix; however, this tool is available only in high-performance systems and specialized hardware, which tend to be expensive. For this reason, a method was developed that allows its computation by calculating it on the FRDM-KL46Z development board.

For this procedure, considering the limitations of the board, it was necessary to develop each of the operations required for the computation of the delayed-type Lyapunov matrix, using the C programming language through the MCUXpresso IDE development environment.

Additionally, using the CoolTerm and MATLAB software, the results were plotted, which made it possible to verify that the calculations performed were correct.

Índice general

DEDICATORIA I

AGRADECIMIENTOS II

RESUMEN III

ABSTRACT IV

ÍNDICE GENERAL V

ÍNDICE DE FIGURAS VIII

ÍNDICE DE TABLAS X

NOTACIÓN XI

1. CAPÍTULO 1 INTRODUCCIÓN 1

1.1. ESTADO DEL ARTE 1

1.1.1. Aplicaciones comparables en entornos embebidos *1*

1.1.1.1. Algoritmo QPMPC en un microcontrolador embebido ARM Cortex M4 2

1.1.1.2. Un control subóptimo basado en la matriz de Lyapunov para sistemas neutros con retardo 2

1.1.1.3. Control óptimo no lineal adaptativo para un proceso de deshidratación de plátano 3

1.1.1.4. Estabilización anidada para control de crucero conectado mediante la matriz de Lyapunov tipo retardada 3

1.1.1.5. Diseño de controladores con retroalimentación retardada mediante la matriz de Lyapunov 3

1.1.2. Soluciones para la matriz de Lyapunov en MATLAB *4*

1.1.2.1. Función *lyap()* de MATLAB 4

1.1.2.2. Toolbox LYAPACK en MATLAB 4

1.2. ANTECEDENTES 5

1.3. PLANTEAMIENTO DEL PROBLEMA 5

1.4. SOLUCIÓN PROPUESTA 6

1.5. JUSTIFICACIÓN 6

1.6. OBJETIVOS 6

1.6.1. Objetivo general *6*

1.6.2. Objetivos específicos *7*

1.7. ALCANCES 7

1.8. LIMITACIONES 8

1.9. HERRAMIENTAS DE SOFTWARE EMPLEADAS 8

1.9.1. MCUXpresso IDE *8*

1.9.2. CoolTerm *9*

1.9.3. MATLAB *10*

2. CAPÍTULO 2 MARCO TEÓRICO 12

2.1. SISTEMAS DINÁMICOS 12

2.1.1. Sistemas con retardo *12*

2.1.1.1. Estabilidad en el sentido de Lyapunov 13

2.1.1.2. Estabilidad asintótica 14

2.1.1.3. Estabilidad exponencial 14

2.2. Teorema de estabilidad de Lyapunov 14
2.3. Matriz fundamental 16
2.4. Matriz de Lyapunov tipo retardada 16
2.5. Cómputo de la matriz de Lyapunov tipo retardada 17
2.6. Operaciones requeridas por la matriz de Lyapunov tipo retardada 18
2.6.1. Ceil 18
2.6.2. Factorial 19
2.6.3. Transpuesta de una matriz 19
2.6.4. Multiplicación de una matriz por un escalar 19
2.6.5. Multiplicación de una matriz por un vector 19
2.6.6. Producto de Kronecker 20
2.6.7. Vectorización de Matrices 20
2.6.8. Norma 1 de una matriz 20
2.6.9. Suma de matrices 21
2.6.10. Multiplicación de matrices 21
2.6.11. Matriz identidad 23
2.6.12. Matriz inversa 23
2.6.13. Potencias de una matriz 24
2.6.14. Matriz exponencial 24
2.6.14.1. Método de aproximación de Padé 25

3. CAPÍTULO 3 DESARROLLO 27

3.1. Microcontroladores y tarjetas de desarrollo de bajo costo 27
3.1.1. FRDM-KL46Z 28
3.1.2. Arduino UNO R3 29
3.1.3. PIC16F887 30
3.1.3.1. Comparación de tarjetas de desarrollo y microcontroladores considerados 31
3.2. Desarrollo de los códigos para el cómputo de la matriz de Lyapunov tipo retardada 32
3.2.1. Archivo funciones.h 32
3.2.1.1. Función productoKronecker 32
3.2.1.1.1. Ejemplo de uso 34
3.2.1.2. Función norma1 34
3.2.1.2.1. Ejemplo de uso 36
3.2.1.3. Función summat2 36
3.2.1.3.1. Ejemplo de uso 37
3.2.1.4. Función mulmat2 38
3.2.1.4.1. Ejemplo de uso 40
3.2.1.5. Función inversa 40
3.2.1.5.1. Ejemplo de uso 43
3.2.1.6. Función inversa2 44
3.2.1.7. Función MatrizPotencia 47
3.2.1.7.1. Ejemplo de uso 50
3.2.1.8. Función MatExpPad 50
3.2.1.8.1. Cálculo de *NP* 54
3.2.1.8.2. Cálculo de *DP* 55
3.2.1.8.3. Cálculo de la fracción racional de matrices 56
3.2.2. Codigo_Principal.c 57
3.2.2.1. Armado de *L* 58

3.2.2.2. Armado de *M* 60
3.2.2.3. Armado de *N* 61
3.2.2.4. Armado de $\boldsymbol{eLh}$ 62
3.2.2.5. Cálculo de $\boldsymbol{CI}$ 62
3.2.2.6. Solución vectorizada 63
3.2.2.7. Preparación y tratamiento de los resultados 64
3.3. Recepción y guardado de datos 66
3.4. Graficación 67

4. CAPÍTULO 4 RESULTADOS 72

4.1. Matrices de entrada 72
4.2. Resultados en la tarjeta FRDM-KL46Z 72
4.3. Resultados en CoolTerm 73
4.4. Graficación de resultados 74
4.5. Comparación de los resultados 75
4.6. Cálculo del error 76

5. CONCLUSIONES Y TRABAJO FUTURO 77

5.1. Trabajos futuros 78

REFERENCIAS 79

Apéndice A Funciones auxiliares para el cálculo de la matriz de Lyapunov tipo retardada 82
A.1. Función transpuesta 82
A.2. Función ceros1 83
A.3. Función ceros2 84
A.4. Función identidad 85
A.5. Función Identidad1 86
A.6. Función Identidad2 87
A.7. Función ceil 88
A.8. Función Mat_esc 89
A.9. Función fact 90
A.10. Función asigna 91
A.11. Función summat 92
A.12. Función mulmat 93
A.13. Función mulmatvec2 95
Apéndice B Código funciones.h completo 96
Apéndice C Código_Principal.c completo 103
Apéndice D Código Graficador.m completo 107
Apéndice E Guía 108
E.1. SDK de la FRDM-KL46Z 108
E.2. Creación del Proyecto 108
E.3. Importación de los códigos 109
E.4. Construcción y ejecución del Codigo_Principal.c 110
E.5. Captura de datos con CoolTerm 111
E.6. Graficación 112

Índice de figuras

Figura 1.1: Logo oficial de MCUXpresso IDE. Imagen tomada de [16], basado en un logotipo de NXP Semiconductors. __9
Figura 1.2: Logotipo oficial de CoolTerm. Imagen tomada de [18]. _____10
Figura 1.3: Logo oficial de MATLAB. Imagen tomada de [20], basado en un logotipo de MathWorks, Inc. _____10
Figura 2.1: Representación gráfica de la estabilidad en el sentido de Lyapunov. Imagen tomada de [26]. _____13
Figura 2.2: Representación gráfica de la estabilidad asintótica. Imagen tomada de [26]. _____14
Figura 2.3; Representación gráfica de la estabilidad exponencial. Imagen tomada de [26]._____14
Figura 3.1: Tarjeta de desarrollo FRDM-KL46Z. Imagen tomada de [36]. _____28
Figura 3.2: Tarjeta de desarrollo Arduino Uno R3. Imagen tomada de [37]. _____29
Figura 3.3: Microcontrolador PIC16F887. Imagen tomada de [39]._____30
Figura 3.4: Diagrama de flujo de la función productoKronecker._____33
Figura 3.5: Diagrama de flujo de la función norma1. _____35
Figura 3.6: Diagrama de flujo de la función summat2. _____37
Figura 3.7: Diagrama de flujo de la función mulmat2. _____39
Figura 3.8: Diagrama de flujo de la función inversa. _____41
Figura 3.9: Diagrama de flujo de la función inversa2. _____45
Figura 3.10: Diagrama de flujo de la función MatrizPotencia. _____48
Figura 3.11: Diagrama de flujo de la función MatExpPad. _____51
Figura 3.12: Diagrama de Flujo del Codigo_Principal.c. _____57
Figura 3.13: Resultados en la interfaz de CoolTerm._____66
Figura 3.14: Resultados guardados en "datos_prueba_1.csv". _____67
Figura 3.15: Diagrama de flujo del código Graficador.m en MATLAB. _____68
Figura 4.1: Bandera colocada al final de “Codigo_Principal.c” en MCUXpresso IDE. _____72
Figura 4.2: Ventana "Expressions View" mostrando el contenido de Y después de una iteración. _____73
Figura 4.3: Resultados en la interfaz de CoolTerm._____73
Figura 4.4: Resultados guardados en "datos_prueba_1.csv". _____74
Figura 4.5: Ventana de comandos de MATLAB. _____74
Figura 4.6: Gráfica en MATLAB de los resultados de la tarjeta FRDM-KL46Z. _____75
Figura 4.7: Gráfica de referencia. Imagen tomada de [14]. _____75
Figura 4.8: Gráfica de los resultados del cálculo de la matriz de Lyapunov tipo retardada en MATLAB._____76
Figura 4.9: Error de los resultados. _____76
Figura 4.10: Gráfica de la integral del error cuadrático (ISE). _____77

Figura A.1: Diagrama de flujo de la función transpuesta. 82
Figura A.2: Diagrama de flujo de la función ceros1. 83
Figura A.3: Diagrama de flujo de la función ceros2. 84
Figura A.4: Diagrama de flujo de la función identidad. 85
Figura A.5: Diagrama de flujo de la función Identidad1. 86
Figura A.6: Diagrama de flujo de la función Identidad2. 87
Figura A.7: Diagrama de flujo de la función ceil. 88
Figura A.8: Diagrama de flujo de la función Mat_esc. 89
Figura A.9: Diagrama de flujo de la función fact. 90
Figura A.10: Diagrama de flujo de la función asigna. 91
Figura A.11: Diagrama de flujo de la función summat. 92

FIGURA A.12: DIAGRAMA DE FLUJO DE LA FUNCIÓN MULMAT. 93
FIGURA A.13: DIAGRAMA DE FLUJO DE LA FUNCIÓN MULMATVEC2. 95

Índice de Tablas

TABLA 3.1: TABLA DE COMPARACIÓN DE LOS MICROCONTROLADORES Y TARJETAS DE DESARROLLO. _____31
TABLA 3.2: ITERACIONES DEL EJEMPLO DE USO DE LA FUNCIÓN INVERSA. _____44

Notación

$\mathbb{R}$	Campo de los números reales.
$\mathbb{N}$	Campo de los números naturales.
$\mathbb{C}$	Campo de los números complejos.
$\mathbb{K}$	Cuerpo $\mathbb{R}$ o $\mathbb{C}$.
$\mathbb{Z}^+$	Campo de los números enteros positivos.
$\mathbb{R}^n$	Espacio n-dimensional sobre números reales definido por $\mathbb{R}^n = \{(x_1, \cdots, x_n) \mid x_i \in \mathbb{R}, 1 \leq i \leq n\}$.
$\mathbb{R}^{n\times n}$	Matrices de dimensión $n \times n$ con coeficientes reales.
$\mathbb{R}^{2n^2\times 2n^2}$	Matrices de dimensión $2n^2 \times 2n^2$ $con\ coeficientes\ reales$.
$\mathbb{K}^{n\times m}$	$\mathbb{K}$- espacio vectorial de matrices de n filas y m columnas con entradas en $\mathbb{K}$.
$\|\| \cdot \|\|$	Norma euclidiana de una matriz o vector.
$\|\| \cdot \|\|_1$	Norma 1 de una matriz o vector.
$Q > 0$	Matriz Q simétrica definida positiva.
$[0, h]$	Intervalo de 0 a H.
$\lceil x \rceil$	Techo (ceil) de $x, donde\ \lceil x \rceil = n \Leftrightarrow n - 1 < x \leq n$.
$n!$	Factorial de n.
A	Matriz de dimensión $n \times m$.
A^T	Transpuesta de A, de dimensión $m \times n$.
A^{-1}	Inversa de la matriz A.
$A \otimes B$	Producto de Kronecker de las matrices A y B.
$A + B$	Suma de las matrices A y B.
AB	Multiplicación de las matrices A y B.
$vec(A)$	Vector en $\mathbb{K}^{n\times m}$, resultado de apilar las filas de $A \in \mathbb{K}^{n\times m}$.
$vec_F(A)$	Vector en $\mathbb{K}^{n\times m}$, resultado de apilar las columnas de $A \in \mathbb{K}^{n\times m}$.
I_n	Matriz identidad de dimensión $n \times n$.
A^k	Matriz A a la potencia k, donde $k \in \mathbb{Z}^+$.
$\sum$	Suma de una sucesión de términos.

Capítulo 1
Introducción

“El retardo es natural en muchos sistemas de control y, con frecuencia, es una fuente de inestabilidad; no obstante, en muchos casos estos mismos retardos pueden tener un efecto estabilizador. Por lo tanto, la estabilidad y el control de los sistemas con retardo son de importancia teórica y práctica, ya que la presencia de un retardo en un sistema degrada el rendimiento” [1].

Para evaluar la estabilidad de los sistemas dinámicos con un retardo puntual, se considera un criterio basado en la matriz de Lyapunov tipo retarda. Más aún, la matriz de Lyapunov tipo retardada ha sido utilizada para desarrollar leyes de control para procesos con retardo inherente. Sin embargo, estos métodos se aplican normalmente en entornos con software y equipos especializados, lo que genera una brecha en los casos donde no se tiene acceso a estas herramientas.

En la presente tesis se plantea el cálculo de la matriz de Lyapunov tipo retardada en una tarjeta de desarrollo de bajo costo, donde se contempla la programación desde cero de las operaciones matemáticas que se necesitan (productos de Kronecker, exponenciación de matrices, suma de matrices, entre otros) en forma de funciones, el desarrollo de un código principal que lleva a cabo su cómputo utilizando las funciones de forma ordenada y la graficación en MATLAB de los resultados obtenidos.

1.1. Estado del arte

La adaptación y optimización de operaciones complejas y que en tarjetas de desarrollo y microcontroladores resultan computacionalmente exigentes ha sido un área de investigación activa, buscando soluciones con diferentes enfoques. Esto toma crucial importancia en un contexto donde se demanda el uso de dichas operaciones y se necesita reducir costos para hacerlos más accesibles. Reportes de esto son los proyectos presentados a continuación, donde se ejemplifican diferentes enfoques para resolver esta problemática.

1.1.1. Aplicaciones comparables en entornos embebidos

Aquí se presentan proyectos que implementaron operaciones matemáticas complejas en hardware más económico en comparación con los usados tradicionalmente para su respectivo desarrollo. A través de distintos enfoques consiguieron adaptar u optimizar los cálculos para que pudieran ser ejecutados en las respectivas plataformas seleccionadas. Además, se reportan trabajos en los que se utiliza la matriz de Lyapunov tipo retardada en leyes de control para procesos reales.

Con estos antecedentes se pretende exponer la importancia de la propuesta planteada en esta tesis, en la que se lleva a cabo el cálculo de la matriz de Lyapunov tipo retardada en una tarjeta de desarrollo de bajo costo, específicamente, la tarjeta FRDM-KL46Z.

1.1.1.1. Algoritmo QPMPC en un microcontrolador embebido ARM Cortex M4

Este proyecto aborda la problemática de reducir los costos asociados al control activo de vibraciones (AVC), ya que las vibraciones afectan a muchas estructuras, por lo que los (AVC) se utilizan en cada vez más ámbitos como: "control de vibraciones de puentes, sistemas de suspensión de vehículos inteligentes, control activo de vibraciones de rotores de helicópteros, control de vibraciones en estructuras espaciales" [2].

Los autores utilizaron el control predictivo cuadrático (QPMPC) para el cálculo de dichas vibraciones, en un microcontrolador ARM Cortex-M4, utilizando para su desarrollo la librería "pqOASES" y utilizando MATLAB para resolver la incompatibilidad, debido a que "pqOASES" está disponible solo en C++, mientras que el código generado estaba en C [2].

De esta manera, Batista y Takács demostraron que un controlador avanzado, que tradicionalmente está reservado para hardware más robusto, puede ser implementado de manera eficaz en una plataforma embebida, cumpliendo con los requerimientos del sistema en tiempo real gracias a una estructura correcta del algoritmo.

Dicho proyecto guarda relación con el planteamiento de esta tesis, donde, si bien el costo del microcontrolador ARM Cortex-M4 es más elevado que el de la tarjeta FRDM-KL46Z, el propósito de adaptar operaciones complejas, tradicionalmente reservadas para hardware robusto, en microcontroladores, es viable gracias a un uso eficiente de los recursos.

Otros trabajos que incluyen el desarrollo de algoritmos complejos en microcontroladores de bajo costo son: "Algoritmos genéticos distribuidos para dispositivos de bajo consumo, bajo costo y memoria limitada" [3], "Una implementación de controlador difuso en microcontrolador de bajo costo para convertidores de energía renovable" [4].

1.1.1.2. Un control subóptimo basado en la matriz de Lyapunov para sistemas neutros con retardo

En este trabajo se propuso una solución con enfoque subóptimo para el control de sistemas lineales con retardos de tipo neutral [5].

La solución propuesta considera el estado del sistema con retardo y evita el enfoque de Inecuaciones Matriciales Lineales (LMI). La ley de control presentada depende no solo del estado del sistema con retardo, sino también de la matriz de Lyapunov tipo retardada asociada al sistema. De tal forma, la matriz de Lyapunov define el control subóptimo desarrollado.

Como prototipo experimental se utilizó un deshidratador atmosférico. El control se programó en una PC, adquiriendo los datos con la tarjeta NI USB-6008 OEM. En la PC se utilizó el software LabVIEW, además se empleó un PID industrial Honeywell DC1040 para obtener en lazo cerrado un sistema con retardo neutral [5]. En este proyecto se destaca la utilización de la matriz de Lyapunov en la síntesis del controlador, cuyo cómputo se llevó a cabo en la PC.

1.1.1.3. Control óptimo no lineal adaptativo para un proceso de deshidratación de plátano

Se presenta un control óptimo no lineal sintetizado vía el enfoque de optimalidad inversa para regular la temperatura de un proceso de deshidratación de plátano [6].

La plataforma de pruebas consiste en una cámara de secado y una cámara de calentamiento con una tubería que recicla el aire caliente dentro del sistema, dicha tubería induce un retardo en el estado. Se utilizó la matriz de Lyapunov tipo retardada para construir el funcional de Lyapunov-Krasovskii tipo completo, desarrollando así la ley de control adaptativo, por lo que su cómputo se hace indispensable para el desarrollo de este trabajo.

1.1.1.4. Estabilización anidada para control de crucero conectado mediante la matriz de Lyapunov tipo retardada

En este artículo, se estudia la estabilidad de un modelo de seguimiento de Control de Crucero Conectado (CCC) [7].

El análisis se realiza en el dominio del tiempo, mediante la matriz de Lyapunov tipo retardada, y propone una metodología basada en la construcción de diagramas de estabilidad desde el vehículo líder hasta el vehículo de la cola. Los resultados reportados muestran que dicho diseño puede utilizarse como herramienta de ajuste en la búsqueda de parámetros de estabilización, destacando que la matriz de Lyapunov tipo retardada permitió seleccionar retardos de estabilización al permitir la construcción de gráficas en el espacio de parámetros, incluidos los retardos [7].

1.1.1.5. Diseño de controladores con retroalimentación retardada mediante la matriz de Lyapunov

Se propone el diseño de controladores con retroalimentación de salida retardada que optimizan una función cuadrática [8].

El criterio de desempeño se expresa en términos de la matriz de Lyapunov tipo retardada, cuya derivada respecto a las ganancias del controlador se utiliza para obtener el gradiente del índice de desempeño [8]. Este trabajo utiliza la matriz de Lyapunov tipo retardada para el desarrollo de controladores y no se limita a usarla como herramienta de análisis de estabilidad resolviendo un problema de optimización cuadrática en el dominio del tiempo.

1.1.2. Soluciones para la matriz de Lyapunov en MATLAB

La matriz de Lyapunov es fundamental para el diseño y análisis de controles, por lo que se han desarrollado muchas maneras de automatizar su algoritmo, como en MATLAB, donde existe la función *lyap()* y el *toolbox* LYAPACK, de los que se hablará a continuación. Cabe recalcar que ninguna de las 2 trabaja la matriz de Lyapunov tipo retardada directamente.

1.1.2.1. Función *lyap()* de MATLAB

Según la documentación oficial en el sitio web de MATLAB [9], la función *lyap()* sirve para calcular a las formas generales y especiales de la matriz de Lyapunov del tipo:

Ecuación de Lyapunov generalizada

$$AXE^T + EXA^T + Q = 0, \tag{1.1}$$

donde:

- A, es una matriz cuadrada de $n \times n$, que representa la dinámica del sistema.
- X, es la matriz incógnita.
- E, es una matriz cuadrada de $n \times n$, generaliza el caso estándar cuando $E = I$.
- Q, es una matriz cuadrada de $n \times n$, definida positiva.
- A^T, es la transpuesta de A.
- E^T, es la transpuesta de E.

1.1.2.2. Toolbox LYAPACK en MATLAB

Otra herramienta utilizada en MATLAB es el toolbox “LYAPACK”, desarrollado para ofrecer soporte en el cálculo de matrices de Lyapunov y ecuaciones de Riccati, utiliza algoritmos implementados en el entorno de MATLAB [10]. LYAPACK permite trabajar con modelos matemáticos de control y análisis de estabilidad en sistemas dinámicos [10].

LYAPACK no está integrado directamente en la suite estándar de MATLAB, sin embargo, es posible descargarse e integrarse al entorno de MATLAB [10]. Sin embargo, LYAPACK está pensado para ejecutarse en estaciones de trabajo de propósito general, aunado al hecho de que está fuertemente integrado en el ecosistema de MATLAB, complica su implementación en sistemas

embebidos, en especial de bajo costo, debido a que no poseen los requerimientos de hardware necesarios [10].

1.2. Antecedentes

"Los sistemas de control suelen operar con retardos, principalmente debido al tiempo que se tarda en adquirir la información necesaria para la toma de decisiones de control y ejecutarlas" [11]. Debido a esto, los retardos están implicados en áreas de la información y comunicación: estabilidad de los sistemas de control en red (NCS) o redes de comunicación de alta velocidad [11].

Los primeros métodos para el análisis de estabilidad en sistemas dinámicos se basaron en las funciones de Lyapunov, si bien, como lo explica en su trabajo Lyapunov, "The general problem of the stability of motion"[12], estas fórmulas estaban inicialmente para sistemas lineales y no lineales sin retardo. Entonces las funciones de Lyapunov permitían establecer la estabilidad de un sistema sin tener que resolver sus ecuaciones diferenciales, convirtiéndolas en una de las herramientas más utilizadas en teoría de control.

"En un sistema con retardo de control determinado, un aspecto relevante a analizar es la estabilidad" [13]. Para ello se puede utilizar la matriz de Lyapunov tipo retardada, pues "es crucial para la aplicación exitosa de las funcionales cuadráticas para el análisis de estabilidad de los sistemas con retardo en el tiempo" [13].

1.3. Planteamiento del problema

La investigación de la estabilidad en sistemas dinámicos con retardo ha cobrado gran relevancia, pues estos sistemas se encuentran presentes en muchos ámbitos de la ingeniería, desde el control de procesos industriales, robótica, telecomunicaciones, etc. Los sistemas dinámicos con retardo dependen de estados anteriores; esto dificulta en gran medida su estudio, análisis y determinación de estabilidad.

La matriz de Lyapunov tipo retardada es una de las herramientas empleadas para tratar a dichos sistemas. Sin embargo, supone la ejecución de operaciones que resultan computacionalmente exigentes, como productos de Kronecker, exponenciación de matrices y aproximaciones basadas en el método de Padé. Esto deja la resolución de la matriz de Lyapunov tipo retardada, reservada para software especializado.

Bajo este contexto, el desarrollo de estos cálculos en dispositivos de bajo costo, como microcontroladores o tarjetas de desarrollo comunes, se vuelve sumamente importante.

Entonces, considerando que el principal problema es la falta de implementaciones de la matriz de Lyapunov tipo retardada en dispositivos de bajo costo, surge la cuestión: **¿es posible la**

implementación del cómputo de la matriz de Lyapunov tipo retardada en una tarjeta de desarrollo o microcontrolador de bajo costo?

1.4. Solución propuesta

La presente tesis propone como solución el desarrollo del cálculo de la matriz de Lyapunov tipo retardada, en un dispositivo de bajo costo, como lo es la tarjeta de desarrollo FRDM-KL46Z de NXP. Con ese fin se ideó crear códigos en lenguaje de programación C, dentro del entorno de programación MCUXpresso IDE, en donde se incorporaron las operaciones requeridas para el cálculo de la matriz en forma de funciones, dichos códigos se implementaron en la tarjeta FRDM-KL46Z, la cual los ejecuta en tiempo real.

Con el propósito de registrar y analizar los resultados obtenidos de la tarjeta FRDM-KL46Z, se empleó el software CoolTerm, que facilitó que los datos obtenidos se guardaran automáticamente en un archivo de hoja de cálculo (.csv).

Estos archivos se procesaron después en MATLAB, donde se generaron gráficos, lo que permitió tener una referencia visual de los resultados y permitió validar con los valores de referencia provenientes del artículo “Lyapunov–Krasovskii approach to the robust stability analysis of time-delay systems” [14].

1.5. Justificación

El análisis de estabilidad y control de sistemas dinámicos con retardo representa un reto importante, la matriz de Lyapunov tipo retardada juega un papel clave en dichos ámbitos, sin embargo, debido a la complejidad que implican sus operaciones, suele estar disponible en software especializado.

Se justifica el cálculo de la matriz de Lyapunov tipo retardada en la tarjeta FRDM-KL46Z, considerada de bajo costo, para mejorar la accesibilidad de las aplicaciones de estabilidad y control, y, del mismo modo, acercando estas herramientas a aplicaciones prácticas en ingeniería.

1.6. Objetivos

Se presentan los objetivos propuestos, tanto general como específicos, para el desarrollo de la tesis.

1.6.1. Objetivo general

Calcular la matriz de Lyapunov tipo retardada en la tarjeta de desarrollo FRDM-KL46Z de NXP, a través del desarrollo de cada operación y proceso requerido por la matriz de Lyapunov tipo

retardada, para ofrecer una alternativa accesible que funcione para el análisis de estabilidad y control de sistemas dinámicos con retardo puntual.

1.6.2. Objetivos específicos

- Programar cada operación que requiere la matriz de Lyapunov tipo retardada en forma de funciones en lenguaje C, para mandarlas a llamar en el código principal.

- Programar el código principal utilizando las funciones de manera ordenada para calcular la matriz de Lyapunov tipo retardada.

- Ejecutar los códigos en la tarjeta de desarrollo FRDM-KL46Z utilizando el software MCUXpresso IDE para ejecutarlos en tiempo real.

- Visualizar los resultados de cada iteración utilizando el software CoolTerm para guardarlos en un archivo de hoja de cálculo (.csv).

- Graficar los resultados utilizando el software MATLAB para permitir validar los resultados obtenidos.

1.7. Alcances

Como se mencionó anteriormente, el objetivo principal de este proyecto es desarrollar las funciones y los códigos que se requieren para calcular la matriz de Lyapunov retardada, utilizando el lenguaje de programación C en el entorno MCUXpresso IDE, con el propósito de ejecutarlos en la tarjeta FRDM-KL46Z. A continuación, se presentan los alcances principales del proyecto:

- **Funciona con matrices cuadradas de $n \times n$:** los códigos de las funciones y programa principal, así como el código de MATLAB, están desarrollados para trabajar con matrices cuadradas de dimensiones $n \times n$ y no están limitados a un tamaño predefinido.

- **Registro de resultados**: los resultados producidos en las iteraciones se almacenan en archivos de hojas de cálculo (.csv) mediante CoolTerm, esto permite la creación de un historial organizado de los datos de manera rápida y sencilla.

- **Graficación**: los resultados se procesan en MATLAB y se genera una gráfica, lo que ofrece una representación de los resultados obtenidos y facilita su valoración.

- **Accesibilidad**: prueba de que el análisis de estabilidad y control en sistemas dinámicos con retardo se puede llevar a cabo en contextos económicos y con recursos de hardware

limitados, siempre que exista una adecuada organización y se desarrollen los códigos específicamente para tener una mejor utilización de los recursos hardware disponibles.

1.8. Limitaciones

- **Automatización parcial:** si bien se logró implementar el cómputo de la matriz de Lyapunov tipo retardada en la tarjeta de desarrollo FRDM-KL46Z, el proceso no es completamente automático, pues necesita la intervención del usuario en distintas fases, como la configuración de los entornos de programación, definición del tamaño de las matrices de entrada tanto en MCUXpresso IDE y MATLAB, así como durante la configuración del nombre del archivo (.csv) y su importación en MATLAB, entre otros.

- **Recursos de hardware:** la tarjeta FRDM-KL46Z no cuenta con unidad de punto flotante (FPU).

- **Tipo de sistema:** este proyecto se enfoca solo en sistemas lineales con un retardo puntual, no toma en cuenta sistemas no lineales o con múltiples retardos.

1.9. Herramientas de software empleadas

Para realizar el cómputo de la matriz de Lyapunov tipo retardada con los objetivos planteados, fueron necesarios los softwares:

- MCUXpresso IDE fue utilizado para el desarrollo, compilación y ejecución de los códigos en la tarjeta FRDM-KL46Z; usa el lenguaje de programación C. Fue seleccionado gracias a que es el IDE oficial de NXP.

- CoolTerm se empleó para establecer comunicación con la tarjeta FRDM-KL46Z, permitiendo tanto visualizar como guardar los resultados obtenidos en tiempo real.

- MATLAB sirvió como plataforma para procesar y analizar los resultados obtenidos y generar gráficas para facilitar su interpretación.

En los siguientes puntos se dará a conocer más acerca de estos softwares.

1.9.1. MCUXpresso IDE

MCUXpresso IDE es el entorno de programación oficial desarrollado por NXP para microcontroladores basados en ARM Cortex-M; en la Figura 1.1 se muestra el logo oficial. Aunque este IDE no estaba específicamente dirigido a la serie Kinetis, a la cual pertenece la tarjeta FRDM-KL46Z, pues para esta serie estaba el software Kinetis Design Studio, sin embargo, NXP en 2017

anunció la descontinuación de Kinetis Design Studio, por lo que MCUXpresso IDE fue la opción recomendada por NXP para migrar [15].

Figura 1.1: Logo oficial de MCUXpresso IDE. Imagen tomada de [16], basado en un logotipo de NXP Semiconductors.

Según NXP Semiconductors [15], MCUXpresso IDE está desarrollado sobre Eclipse CDT y utiliza la cadena de herramientas GNU ARM Embedded Toolchain, esto permite ofrecer una plataforma de estándares abiertos, lo que lo hace compatible para las familias de microcontroladores LPC, i.MX RT y Kinetis.

De acuerdo con NXP Semiconductors [15], entre algunas de sus características principales destacan:

- Compatibilidad con las tarjetas Freedoom (FRDM).
- Uso de MCUXpresso Config Tools para la configuración de pines.
- SDK para las tarjetas de desarrollo.
- Herramientas de depuración avanzada mediante Serial Wire Debug como:
    - Breakpoints.
    - Inspección de memoria.
    - Visualización de registros.

1.9.2. CoolTerm

CoolTerm es una aplicación gratuita de terminal de puerto serie, funciona sin emulación de terminal, pues se centra en el manejo de datos, permite intercambiar datos con hardware conectado a puertos serie [17]. En la, Figura 1.2 se muestra su logotipo oficial.

Figura 1.2: Logotipo oficial de CoolTerm. Imagen tomada de [18].

Está disponible para Windows y macOS, Linux no cuenta con soporte oficial [18] y entre sus principales características están [18]:

- Visualización de datos recibidos en formato de texto.
- Compatibilidad con múltiples conexiones simultáneas.
- Control de flujo por hardware (CTS, DTR).
- Control de flujo por software (XON/XOFF)
- Registro de datos recibidos.

1.9.3. MATLAB

MATLAB es una aplicación especializada en el cómputo numérico y análisis de datos; debido a sus prestaciones y exactitud, es ampliamente utilizada en ingeniería y ciencia en general [19], en la Figura 1.3, se muestra su logotipo oficial.

Figura 1.3: Logo oficial de MATLAB. Imagen tomada de [20], basado en un logotipo de MathWorks, Inc.

MATLAB combina un entorno de escritorio perfecto para el análisis iterativo y procesos de diseño [21]. Utiliza un lenguaje de programación propio conocido como “lenguaje M” o “lenguaje MATLAB”, lo que facilita la expresión de matemáticas de matrices y *Arrays* directamente [19].

Posee diversas características, entre las que destacan:

- Funciones integradas para la visualización, creación de gráficas e interfaces visuales y aplicaciones específicas [21].
- Extensibilidad mediante *toolboxes* [22].
- Capacidad para interactuar con hardware y otros lenguajes de programación [23].

Facilidad para manejar datos complejos gracias a su lenguaje de programación [23].

Capítulo 2
Marco teórico

En esta sección se presenta un conjunto de fundamentos teóricos y conceptos que sustentan el desarrollo de la presente tesis. En sus apartados se abordan principios necesarios para comprender tanto la problemática como la solución propuesta; del mismo modo, dan a conocer las herramientas y conocimientos involucrados.

Los temas aquí desarrollados abarcan desde conceptos esenciales para la matriz de Lyapunov tipo retardada, así como también se presentan las características y función de los IDE y herramientas utilizadas en el contexto del trabajo realizado.

2.1. Sistemas dinámicos

"Los sistemas dinámicos son sistemas cuyo estado está especificado de forma única por un conjunto de variables y cuyo comportamiento se describe mediante reglas predefinidas" [24], los cuales pueden describirse en tiempo continuo.

Sistema dinámico de tiempo continuo, este tipo de modelo es llamado una ecuación diferencial [24]:

$$\frac{dx}{dt} = F(x, t), \tag{2.1}$$

donde:

- x, es el estado del sistema en tiempo continuo.
- t, es el tiempo.
- F, es la función de evolución en tiempo continuo.

2.1.1. Sistemas con retardo

Los sistemas con retardo son comunes en sistemas que tienen tiempos de procesamiento considerables por distintos factores como procesamiento del control, tratamiento de señales, retardos en las mediciones, etc. [25].

Mientras que para los sistemas lineales libres de retardo se tiene el siguiente sistema general [25]:

$$\dot{x}(t) = Ax(t), \qquad x_0 = x(t_0), \tag{2.2}$$

donde:

- $x(t)$, es el vector de estado en el tiempo t.
- x_0, es la condición inicial del estado.
- $x(t_0)$, es el vector de estado en el instante inicial.

Para los sistemas lineales con un retardo puntual se tiene el siguiente sistema general [13]:

$$\dot{x}(t) = A_0x(t) + A_1x(t-h), \qquad x(\theta) = \varphi(\theta), \qquad \theta \in [-h, 0], \qquad t \geq 0, \tag{2.3}$$

donde:

- A_0, es la matriz del sistema sin retardo.
- A_1, es la matriz del sistema asociada al retardo.
- $h > 0$, es el retardo puntual.
- Sea φ: $[-h, 0] \longrightarrow \mathbb{R}^n$ una función inicial.

Y x_t denota la restricción de la solución del sistema en el segmento $[t-h, t]$ [13]:

$$x_t(\theta) = x(t-\theta), \ \theta \in [-h, 0]. \tag{2.4}$$

2.1.1.1. Estabilidad en el sentido de Lyapunov

Como se observa en la Figura 2.1, la estabilidad en el sentido de Lyapunov es cuando el conjunto de condiciones iniciales $\varphi(t)$ y la respuesta del sistema $x(t)$ comienzan en $\varphi(0)$ y $x(0)$, lo que genera un entorno inicial δ lo suficientemente cerca del punto de equilibrio x_p que mantiene a ϵ unidades del punto de equilibrio, pero nunca lo alcanza [26].

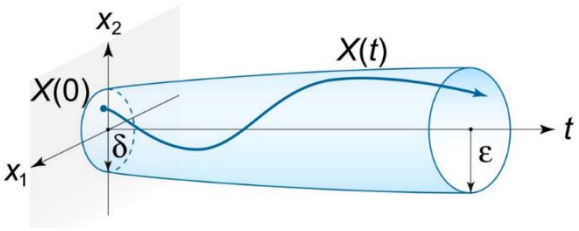


Figura 2.1: Representación gráfica de la estabilidad en el sentido de Lyapunov. Imagen tomada de [26].

Teniendo el siguiente comportamiento [26]:

$$|x(0) - \varphi(0)| < \delta \Rightarrow |x(t) - \varphi(t)| < \in. \tag{2.5}$$

2.1.1.2. Estabilidad asintótica

Como se observa en la Figura 2.2, la estabilidad asintótica es cuando el conjunto de condiciones iniciales $\varphi(t)$ y la respuesta del sistema $x(t)$, se encuentran lo suficientemente cerca del punto de equilibrio y eventualmente lo alcanza, cuando el tiempo tiende a infinito [26].

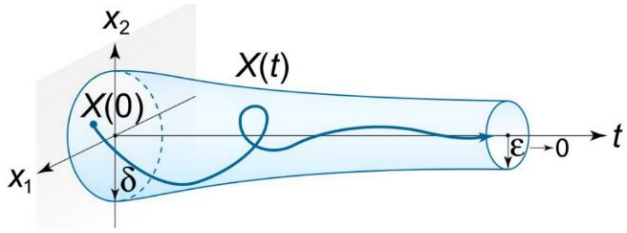


Figura 2.2: Representación gráfica de la estabilidad asintótica. Imagen tomada de [26].

Teniendo el siguiente comportamiento [26]:

$$|x(0) - \varphi(0)| < \delta \Rightarrow \lim_{t \to +\infty} |x(t) - \varphi(t)| = 0. \tag{2.6}$$

2.1.1.3. Estabilidad exponencial

Como se muestra en la Figura 2.3, la estabilidad exponencial ocurre cuando el conjunto de condiciones iniciales $\varphi(t)$ y la respuesta del sistema $x(t)$, se encuentran cerca del punto de equilibrio y convergen a una tasa de cambio exponencial [26].

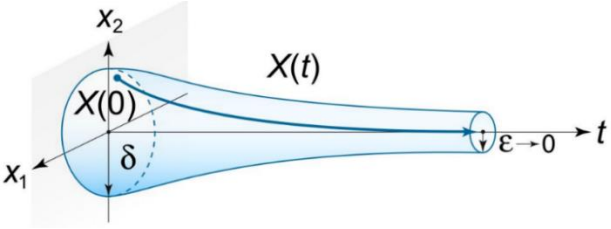


Figura 2.3; Representación gráfica de la estabilidad exponencial. Imagen tomada de [26].

Teniendo el siguiente comportamiento [26]:

$$|x(0) - \varphi(0)| < \delta \Rightarrow |x(t) - \varphi(t)| \leq \alpha |x(0) - \varphi(0)| e^{-\beta t}. \tag{2.7}$$

2.2. Teorema de estabilidad de Lyapunov

"La prueba de estabilidad de Lyapunov se limita a verificar la estabilidad asintótica y exponencial, que resulta bastante útil para el diseño de controladores" [27]. La teoría de estabilidad de Lyapunov nos proporciona una condición alternativa para comprobar si los sistemas invariantes en el tiempo (LTI) son homogéneos [27].

Se introduce de manera breve la definición de matriz definida positiva para comprender los enunciados posteriores.

Una matriz Q simétrica de $n \times n$ es definida positiva si [27]:

$$x^T Q x > 0, \ \forall x \in \mathbb{R}^n \backslash \{0\}, \tag{2.8}$$

donde:

- x, es un vector de $n \times 1$.
- x^T, es la transpuesta de x.

"Cuando el signo $>$, se cambia por $<$, se obtiene la definición de una matriz definida negativa. Las matrices definidas positivas son siempre no singulares, y sus inversas también son definidas positivas." [27].

Teorema 1 *[27] Sea el sistema homogéneo en tiempo continúo descrito en la ecuación (2.2).*

Las siguientes condiciones son equivalentes:

1. *El sistema es asintóticamente estable.*
2. *El sistema es exponencialmente estable.*
3. *Todos los valores de la matriz A tienen parte real estrictamente negativa (Hurwitz).*
4. *Para toda matriz simétrica definida positiva Q, existe una solución P positiva definida a la ecuación de Lyapunov:*

$$A^T P + PA = -Q, \tag{2.9}$$

donde:

- *A, es la matriz dinámica del sistema.*
- *A^T, es la transpuesta de A.*

- *P, es una matriz simétrica.*
- *Q, es una matriz dada, simétrica y definida positiva.*

2.3. Matriz fundamental

Se introduce de manera breve la definición de matriz fundamental que cumple un rol similar para el sistema (2.3).

"[28] Se dice que la matriz $K(t)$ de dimension $n \times n$ es la matriz fundamental del sistema (2.3) si se satisface la ecuación matricial (2.10)".

$$\frac{d}{dt}K(t) = A_0K(t) + A_1K(t-h),\ t \geq 0, \tag{2.10}$$

y $K(t) = 0_{n\times n}$ para todo $t < 0, k(0) = I$, donde I es la matriz identidad de $n \times n$.

La matriz fundamental también satisface la ecuación matricial (2.11), sin embargo, no significa que la matriz $K(t)$ conmute individualmente con los coeficientes matriciales $A_k,\ K = 0, 1$ [28].

$$\frac{d}{dt}K(t) = K(t)A_0 + K(t-h)A_1,\ t \geq 0. \tag{2.11}$$

2.4. Matriz de Lyapunov tipo retardada

La matriz de funciones $U(\tau)$ es conocida como la matriz de Lyapunov tipo retardada (2.12) asociada a W, donde K es la matriz fundamental del sistema lineal con retardo puntual (2.3) [13]:

$$U(\tau) = \int_0^\infty K^T(t)WK\,(t+\tau)dt. \tag{2.12}$$

A continuación se analizan las propiedades básicas de $U(\tau)$ [13].

1. Propiedad dinámica:

$$\frac{d}{dt}U(\tau) = U(\tau)A_0 + U(\tau-h)A_1,\ \tau \geq 0. \tag{2.13}$$

2. Propiedad simétrica:

$$U(-\tau) = U^T(\tau), \qquad \tau \geq 0, \tag{2.14}$$

donde U^T es la transpuesta de U.

3. Propiedad algebraica:

$$U(0)A_0 + U(-h)A_1 + A_0^T U(0) + A_1^T U(h) = -W. \quad (2.15)$$

Las ecuaciones (2.13), (2.14) y (2.15) admiten una única solución cuando el sistema satisface la condición de Lyapunov (2.16) [13].

Se dice que el sistema (2.3) satisface la condición de Lyapunov si el espectro del sistema [13]:

$$\Lambda = \{s: det(sI - A_0 - e^{-sh}A_1) = 0\}. \quad (2.16)$$

2.5. Cómputo de la matriz de Lyapunov tipo retardada

El cálculo de la matriz de Lyapunov tipo retardada puede reducirse a la construcción de un problema especial de valores de frontera para un sistema lineal libre de retardo mediante el método semianalítico [13]. Este método consiste en obtener las dinámicas de la componente de la matriz de Lyapunov tipo retardada [13]. Se tiene a la siguiente expresión:

$$Y(\tau) = e^{L\tau}CI \text{ para } \tau \in [0, h]. \quad (2.17)$$

En el que el vector $Y(\tau)$ contiene las componentes de la matriz de Lyapunov en las primeras n-posiciones.

Donde: $L \in \mathbb{R}^{2n^2 \times 2n^2}$, y es una matriz estructurada construida a partir de las matrices de entrada A_0, A_1, I y productos de Kronecker de la siguiente manera [13]:

$$L = \begin{bmatrix} A_0^T \otimes I & A_1^T \otimes I \\ -I \otimes A_1^T & -I \otimes A_0^T \end{bmatrix} \quad (2.18)$$

Siendo:

- A_0^T, es la transpuesta de A_0, que es la matriz de las variables de estado sin retardo.
- A_1^T, es la transpuesta de A_1, que es la matriz de las variables de estado con retardo.
- I, es una matriz identidad.
- $\otimes$, representa el producto de Kronecker.

CI (Condiciones iniciales):

$$CI = (M + Ne^{Lh})^{-1}W_1, \quad (2.19)$$

donde M se construye como se muestra en la ecuación (2.20), N se construye como se muestra en la ecuación (2.21) y W_1 se construye como se muestra en la ecuación (2.22) [13].

$$M = \begin{bmatrix} I & 0 \\ A_0^T \otimes I & A_1^T \otimes I \end{bmatrix} \tag{2.20}$$

En donde 0, es una matriz de $n^2 \times n^2$ llena de ceros.

$$N = \begin{bmatrix} 0 & -I \\ I \otimes A_1^T & I \otimes A_0^T \end{bmatrix} \tag{2.21}$$

$$W_1 = \begin{bmatrix} vec(0_{n\times n}) \\ -vec(W) \end{bmatrix} \tag{2.22}$$

Donde:

- $vec(0_{n\times n})$, es el vector resultante de apilar las columnas de la matriz $0_{n\times n}$.
- $-vec(W)$, es el vector resultante de apilar las columnas de la matriz $(-W)$.

2.6. Operaciones requeridas por la matriz de Lyapunov tipo retardada

En esta sección se presentan de manera breve las operaciones matemáticas que requiere el cálculo de la matriz de Lyapunov tipo retardada, las cuales se tomaron como base para programarlas en "funciones.h".

2.6.1. Ceil

"Dado cualquier número real x, el techo (ceil) de x, que se denota por $\lceil x \rceil$, se define de la siguiente manera" [29]:

$$\lceil x \rceil = n, \text{ es el único entero tal que } n - 1 < x \leq n. \tag{2.23}$$

Simbólicamente, "si x es un número real y n es un entero, entonces" [29]:

$$\lceil x \rceil = n \ \Leftrightarrow \ n - 1 < x \leq n. \tag{2.24}$$

2.6.2. Factorial

Para cada entero positivo n, la cantidad n factorial que se denota por $n!$, se define como el producto de todos los enteros de 1 a n, como se muestra en la ecuación (2.25) [29].

$$n! = n \cdot (n-1) \cdots 3 \cdot 2 \cdot 1. \tag{2.25}$$

Mientras que 0!, se define como 1 [29].

2.6.3. Transpuesta de una matriz

Dada una matriz A de $m \times n$ (2.26), la transpuesta de A es la matriz $n \times m$ (2.27), denotada mediante A^T, cuyas columnas se forman a partir de las filas correspondientes de A [30].

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}, \tag{2.26}$$

$$A^T = \begin{bmatrix} a_{00} & a_{10} \\ a_{01} & a_{11} \end{bmatrix}. \tag{2.27}$$

2.6.4. Multiplicación de una matriz por un escalar

"Si A es una matriz cualquiera (2.28) y c, es cualquier escalar (2.29), entonces el producto cA (2.30), es la matriz que se obtiene al multiplicar cada elemento de A por c" [31], es decir:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \tag{2.28}$$

$$c = z, \tag{2.29}$$

entonces cA es:

$$cA = \begin{bmatrix} za & zb \\ zc & zd \end{bmatrix}. \tag{2.30}$$

2.6.5. Multiplicación de una matriz por un vector

Si A es una matriz de $m \times n$, con columnas $a_1, a_2, \cdots\ a_n$, y si x está en $\mathbb{R}^n$, entonces el producto de A y x, denotado como Ax, es la combinación lineal de las columnas de A, utilizando las correspondientes entradas en x [30], es decir:

$$Ax = [a_1, a_2, \cdots, a_n]\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = x_1a_1 + x_2a_2 + \cdots + x_na_n. \quad (2.31)$$

2.6.6. Producto de Kronecker

Las operaciones de álgebra matricial, por lo general, tienen restricciones en el tamaño de las matrices involucradas; por ejemplo $A + B$ se define si A y B tienen el mismo orden, o el producto de AB se define si la cantidad de columnas de A es igual a la cantidad de filas de B.

Sin embargo, el producto de Kronecker es una operación que, como menciona María Gabriela en su tesis doctoral [32], es una operación binaria definida en todo el conjunto de matrices sin restricciones de tamaños.

Sea $A \in \mathbb{R}^{m\times n}$ y $B \in \mathbb{R}^{p\times q}$, el producto de Kronecker entre A y B, denotado como $A \otimes B$ se define como una matriz bloque de tamaño $(mp) \times (pq)$, construida al multiplicar cada elemento de A por toda la matriz B [32], como se muestra en la ecuación (2.32).

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}. \quad (2.32)$$

"Es importante considerar que el producto de Kronecker no es conmutativo, es decir $A \otimes B \neq B \otimes A$" [32].

2.6.7. Vectorización de Matrices

Al manipular matrices en algunas ocasiones resulta útil vectorizarlas, es decir, transformarlas en un vector apilando sus filas o columnas, a este proceso se le llama vectorización una matriz [32].

"Dada una matriz cualquiera $A \in \mathbb{K}^{m\times n}$, $vec(A)$ (2.33) y $vec_F(A)$ (2.34) son los vectores columna en $\mathbb{K}^{m\times n}$ que resultan de apilar las columnas y las filas de A, respectivamente" [32].

$$vec(A) = [a_{00}, \cdots, a_{(m-1)0}, a_{01} \cdots, a_{(m-1)1}, \cdots, a_{0(n-1)}, \cdots, a_{(m-1)(n-1)}]^T, \quad (2.33)$$

$$vec_F(A) = [a_{00}, \cdots, a_{0(n-1)}, a_{10} \cdots, a_{1(n-1)}, \cdots, a_{(m-1)0}, \cdots, a_{(m-1)(n-1)}]^T. \quad (2.34)$$

2.6.8. Norma 1 de una matriz

La norma 1, mostrada en la ecuación (2.35), calcula la suma absoluta máxima de los elementos por columna en una matriz [33].

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{n} |a_{ij}|, \tag{2.35}$$

donde:

- $\|A\|_1$, es la Norma 1 de la matriz A.
- $\sum_{1=1}^{n} |a_{ij}|$, es la suma de todos los elementos de la columna j.
- a_{ij}, es el elemento de la fila *i*, columna *j* de la matriz A.
- $\max_{1 \le j \le n}$, indica que se toma el valor máximo de todas las sumas de los valores de las columnas.

2.6.9. Suma de matrices

"La suma de matrices se define únicamente cuando las matrices son del mismo tamaño, es decir cuando sean $A = (a_{ij})$ y $b = (a_{ij})$" [34]. La suma de matrices se obtiene al sumar cada componente correspondiente de A y B [34] como se observa en la ecuación (2.36).

$$A + B = (a_{ij} + b_{ij}) = \begin{bmatrix} a_{00} + b_{00} & a_{01} + b_{01} & \cdots & a_{0n} + b_{0n} \\ a_{10} + b_{10} & a_{11} + b_{11} & \cdots & a_{1n} + b_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m0} + b_{m0} & a_{m1} + b_{m1} & \cdots & a_{mn} + b_{mn} \end{bmatrix}. \tag{2.36}$$

2.6.10. Multiplicación de matrices

Dos matrices se pueden multiplicar únicamente si el número de columnas de la primera matriz es igual al número de filas de la segunda. "De otro modo, los vectores que forman el renglón *i* en *A* (2.37) y la columna *j* de *B* (2.38) no tendrán el mismo número de componentes y el producto punto en la ecuación no estará definido" [34].

$$A = \begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0n} \\ a_{10} & a_{11} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m0} & a_{m1} & \cdots & a_{mn} \end{bmatrix}, \tag{2.37}$$

$$B = \begin{bmatrix} b_{00} & b_{01} & \cdots & b_{0p} \\ b_{10} & b_{11} & \cdots & b_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n0} & b_{n1} & \cdots & b_{np} \end{bmatrix}. \tag{2.38}$$

Para llevar a cabo la multiplicación de matrices, se deben multiplicar las filas de A (2.39) por las columnas de B (2.40), lo que da como resultado a AB, sin embargo, no es conmutativo, es decir que $AB \neq BA$ [34], por ejemplo:

$$A = \begin{vmatrix} 1 & -2 \\ 3 & 4 \end{vmatrix}, \tag{2.39}$$

$$B = \begin{vmatrix} -1 & 5 \\ 2 & -3 \end{vmatrix}, \tag{2.40}$$

calculando a $C = AB$:

$$C_{00} = (1)(-1) + (-2)(2) = -1 - 4 = -5, \tag{2.41}$$

$$C_{01} = (1)(5) + (-2)(-3) = 5 + 6 = 11, \tag{2.42}$$

$$C_{10} = (3)(-1) + (4)(2) = -3 + 8 = 5, \tag{2.43}$$

$$C_{11} = (3)(5) + (4)(-3) = 15 - 12 = 3, \tag{2.44}$$

calculando a $D = BA$:

$$C_{00} = (-1)(1) + (5)(3) = -1 - 15 = 14, \tag{2.45}$$

$$C_{01} = (-1)(-2) + (5)(4) = 2 + 20 = 22, \tag{2.46}$$

$$C_{10} = (2)(1) + (-3)(3) = 2 - 9 = -7, \tag{2.47}$$

$$C_{11} = (2)(-2) + (-3)(4) = -4 - 12 = -16, \tag{2.48}$$

dando como resultados a:

$$C = \begin{vmatrix} -5 & 11 \\ 5 & 3 \end{vmatrix}, \tag{2.49}$$

$$D = \begin{vmatrix} 14 & 22 \\ -7 & -16 \end{vmatrix}. \tag{2.50}$$

2.6.11. Matriz identidad

"La matriz identidad I_n de una matriz $n \times n$, es una matriz de $n \times n$ cuyos elementos de la diagonal principal son iguales a 1 y todos los demás son 0" [34], es decir;

$$I_n = b_{ij} \text{ donde } \begin{cases} 1 \; si \; i = j \\ O \; si \; si \neq j \end{cases}, \tag{2.51}$$

teniendo la forma:

$$\begin{pmatrix} 1_{00} & \cdots & 0_{0n} \\ \vdots & \ddots & \vdots \\ 0_{m0} & \cdots & 1_{mn} \end{pmatrix} \tag{2.52}$$

2.6.12. Matriz inversa

Una matriz A de $n \times n$ es invertible si existe otra matriz C de $n \times n$ tal que:

$$CA = I, \tag{2.53}$$

$$AC = I, \tag{2.54}$$

donde $I = I_n$, la matriz identidad $n \times n$. En este caso, C es un inverso de A [30].

Para obtener la inversa de una matriz existen varios métodos, pero para este proyecto se utilizó el método de Gauss-Jordan, del que se da un ejemplo a continuación.

Si se tiene una matriz *A:*

$$A = \begin{bmatrix} 2 & 1 \\ 5 & 3 \end{bmatrix}, \tag{2.55}$$

se forma junto a su identidad:

$$\left[\begin{array}{cc|cc} 2 & 1 & 1 & 0 \\ 5 & 3 & 0 & 1 \end{array}\right], \tag{2.56}$$

al hacer 1 el pivote de la fila 1 ($R_1 \leftarrow \frac{1}{2}R_1$):

$$\left[\begin{array}{cc|cc} 1 & 1/2 & 1/2 & 0 \\ 5 & 3 & 0 & 1 \end{array}\right], \tag{2.57}$$

se anula el elemento debajo del pivote en la columna 1 ($R_2 \leftarrow R_2 - 5R_1$):

$$\begin{bmatrix} 1 & 1/2 & | & 1/2 & 0 \\ 0 & 1/2 & | & -5/2 & 1 \end{bmatrix}, \tag{2.58}$$

al hacer 1 el pivote de la fila 2 ($R_2 \leftarrow 2R_2$):

$$\begin{bmatrix} 1 & 1/2 & | & 1/2 & 0 \\ 0 & 1 & | & -5 & 2 \end{bmatrix}, \tag{2.59}$$

se anula el elemento arriba del pivote de la columna 2 ($R_1 \leftarrow R_1 - \frac{1}{2}R_2$):

$$\begin{bmatrix} 1 & 0 & | & 3 & -1 \\ 0 & 1 & | & -5 & 2 \end{bmatrix}, \tag{2.60}$$

como el lado izquierdo ahora contiene a *I*, el lado derecho ahora contiene a A^{-1}:

$$A^{-1} = \begin{bmatrix} 3 & -1 \\ -5 & 2 \end{bmatrix}. \tag{2.61}$$

2.6.13. Potencias de una matriz

"Si A es una matriz de $n \times n$, y k es un entero positivo, entonces A^k denota el producto de k copias de A" [30]:

$$A^k = A \cdot A \cdot \ldots \cdot A \ (k \text{ veces}). \tag{2.62}$$

En el caso de que k sea 0, el resultado es la misma matriz A.

2.6.14. Matriz exponencial

Una de las operaciones matriciales más calculadas es la exponencial denotada por la ecuación (2.63) [35].

$$e^{At} = \sum_{k=0}^{\infty} \frac{(At)^k}{k!}, \tag{2.63}$$

donde:

- e^{At}, es la exponencial de la matriz A.
- $(At)^k$, es el producto matricial repetido.
- $\Sigma_{k=0}^{\infty}$, indica la suma infinita.

- $k!$, es el factorial de k.

Una manera para calcularlo es el método de aproximación de Padé, que se explicará a continuación.

2.6.14.1. Método de aproximación de Padé

Una clase muy útil para calcular e^{At} son las funciones de Padé [35] definidas por:

$$R_{pq}(z) = D_{pq}(z)^{-1} N_{pq}(z), \tag{2.64}$$

que se puede reescribir como:

$$e^{z} = D_{pq}(z)^{-1} N_{pq}(z) = \frac{N_{p,q}(z)}{D_{p,q}(z)}, \tag{2.65}$$

donde:

$$N_{pq}(z) = \sum_{k=0}^{\rho} \frac{(p+q-k)!p!}{(p+q)!k!(p-k)!} z^{k}, \tag{2.66}$$

$$D_{pq}(z) = \sum_{k=0}^{q} \frac{(p+q-k)!q!}{(p+q)!k!(q-k)!} (-z)^{k}, \tag{2.67}$$

en los cuales:

- p, q, son los grados de los polinomios en el numerador (p) y denominador (q).
- !, indica el factorial.
- k, es el índice de sumatoria.

Para elaborar su código, se reescribió a *NP* (2.68) Y *DP* (2.70) en términos de *j*:

$$NP = \sum_{j=0}^{\rho} c_j \cdot L^{j}, \tag{2.68}$$

donde:

$$c_j = \frac{(p+q-j)!p!}{(p+q)!j!(p-j)!}, \tag{2.69}$$

y

$$DP = \sum_{j=0}^{q} d_j \cdot L^j, \tag{2.70}$$

donde:

$$d_j = \frac{(p+q-j)!q!}{(p+q)!j!(q-j)!}. \tag{2.71}$$

Capítulo 3
Desarrollo

En este capítulo se mostrará el desarrollo del cómputo de la matriz de Lyapunov tipo retardada, desde la elección de la tarjeta FRDM-KL46Z y otras tarjetas de desarrollo y microcontroladores considerados, el desarrollo de las principales operaciones que se necesitan para el cálculo de la matriz de Lyapunov tipo retardada en forma de funciones, así como el desarrollo del código principal y el proceso de graficación de los resultados obtenidos, utilizando de manera auxiliar a los softwares CoolTerm y MATLAB.

3.1. Microcontroladores y tarjetas de desarrollo de bajo costo

Los microcontroladores, así como las tarjetas de desarrollo consideradas de bajo costo, son ampliamente utilizados en sistemas embebidos, en gran medida gracias a su accesibilidad, que permite el ahorro de recursos económicos en aplicaciones específicas.

Por lo general, están diseñados para ejecutar tareas sencillas en tiempo real, como el control de sensores, actuadores e incluso adquisición de datos; entre sus características principales, se encuentran:

- Arquitecturas simples (ARM, AVR, Cortex-M, etc.).
- Frecuencias de reloj de entre 16 MHz y 80 MHz.
- Memorias RAM de entre 2 y 64 KB.
- Memoria FLASH de entre 16 KB y 512 KB.
- En su mayoría no cuentan con unidad de punto flotante (FPU).

Si bien su punto fuerte es que son accesibles, también los limita en gran medida, pues a la hora de ejecutar operaciones complejas como lo es la matriz de Lyapunov tipo retardada, son descartados casi instantáneamente debido a sus limitaciones de hardware.

En los siguientes puntos se presentan las 3 opciones consideradas para ejecutar la matriz de Lyapunov tipo retardada, así como sus características principales y, finalmente, una comparación general que permitió elegir la mejor opción.

3.1.1. FRDM-KL46Z

La tarjeta de desarrollo FRDM-KL46Z, como se observa en la Figura 3.1, tiene un diseño sencillo que es ideal para el prototipado rápido de aplicaciones basadas en microcontroladores; contiene un microcontrolador de la serie Kinetis L Series, basado en el núcleo ARM Cortex-M0+ [36].

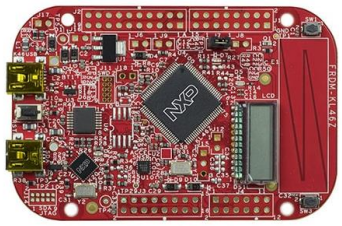


Figura 3.1: Tarjeta de desarrollo FRDM-KL46Z. Imagen tomada de [36].

"La tarjeta FRDM-KL46Z incorpora el adaptador de serie y depuración integrado OpenSDA, estándar abierto de Freescale, lo que ofrece comunicaciones serie, programación de memoria flash y depuración con control de ejecución" [36].

Especificaciones NXP Semiconductors [36]:

- Microcontrolador MKL46Z256VLL4.
- Memoria flash de 256 KB.
- Memoria SRAM de 32 KB.
- Arquitectura: ARM Cortex-M0+ de 32 bits.
- Frecuencia de reloj de 48 MHz.
- Dos ADC de 16 bits.
- Unidad de punto flotante (FPU): No disponible.
- Interfaz USB OTG, comunicación UART, SPI, I^2C.
- Depuración y programación: A través de la interfaz OpenSDA.

3.1.2. Arduino UNO R3

La placa de desarrollo Arduino UNO R3 mostrada en la Figura 3.2 es la más conocida e insignia de Arduino, pues es perfecta para familiarizarse con la programación y la electrónica. Está equipada con el microcontrolador ATmega328P y el procesador ATMega16U2 [37].

Figura 3.2: Tarjeta de desarrollo Arduino Uno R3. Imagen tomada de [37].

Entre sus especificaciones se encuentran [37]:

- Microcontrolador ATmega328P.
- 6 pines de E/S digitales PWM.
- 6 canales de conversión A/D de 10 bits.
- Memoria SRAM de 2 KB.
- Memoria flash de 32 KB.
- Frecuencia de reloj de 16 MHz.
- Unidad de punto flotante (FPU): No disponible.

3.1.3. PIC16F887

El PIC16F887 fue desarrollado por Microchip Technology, y debido a su muy bajo costo, es muy accesible, lo que lo facilita en entornos académicos y de prototipado [38]. Posee un encapsulado de tipo PDIP de 40 pines, como se muestra en la Figura 3.3.

Figura 3.3: Microcontrolador PIC16F887. Imagen tomada de [39].

Entre sus especificaciones principales se encuentran [38]:

- Arquitectura: RISC de 8 bits con conjunto de 35 instrucciones.
- Frecuencia de operación de hasta 20 MHz.
- Memoria flash de 14 KB.
- Memoria RAM de 368 bytes y EEPROM de 256 bytes.
- 35 pines de propósito general.
- 14 canales conversores A/D de 10 bits.
- Dos temporizadores de 8 bits (TMR0 y TMR2) y uno de 16 bits (TMR1).
- Dos módulos CCP de Captura/Comparación/PWM.
- Interfaces de comunicación USART, SPI e I^2C.
- Voltaje de operación de entre 2.0 V a 5.5 V.
- Unidad de punto flotante (FPU): No disponible.

3.1.3.1. Comparación de tarjetas de desarrollo y microcontroladores considerados

	FRDM-KL46Z	**Arduino UNO R3**	**PIC16F887A**
Arquitectura	ARM Cortex-M0+ (32 bits)	AVR (8 bits)	RISC (8 bits)
Frecuencia de reloj	48 MHz	16 MHz	20 MHz
Memoria RAM	32 KB	2 KB	368 bytes
Memoria Flash	256 KB	32 KB	14 KB
Unidad de Punto Flotante	No	No	No
Conversores A/D	2 canales de 16 bits	6 canales de 10 bits	14 canales de 10 bits
Interfaces de comunicación	UART, SPI, I²C, USB OTG	UART, SPI, I²C	USART, SPI, I²C
Aplicaciones principales.	Educativo y sistemas embebidos reales	Educación y prototipado básico	Educación básica y control simple

Tabla 3.1: Tabla de comparación de los microcontroladores y tarjetas de desarrollo.

Se seleccionó a la tarjeta FRDM-KL46 para ejecutar el cálculo de la matriz de Lyapunov tipo retardada debido a que es la que posee mejores prestaciones de las opciones consideradas.

3.2. Desarrollo de los códigos para el cómputo de la matriz de Lyapunov tipo retardada

Para el desarrollo del proyecto, se consideró dividir las tareas en 2 fases principales, siendo la primera el construir los códigos para calcular la matriz de Lyapunov tipo retardada, en lenguaje C, dentro del entorno de desarrollo MCUXpresso IDE, considerando que la tarjeta de desarrollo seleccionada fue la FRDM-KL46Z de NXP.

La segunda tarea consistió en graficar los resultados obtenidos de ejecutar los códigos de la primera fase en la FRDM-KL46Z; para mayor información, consulte el **Apéndice E**.

Dentro de la primera fase, se consideró que hacer todo dentro de un único código hace más complicado su desarrollo, por lo que se dividió en 2 códigos. Al primero se le llamó "funciones.h" y consistió en desarrollar todas las operaciones que necesita la matriz de Lyapunov tipo retardada. Al segundo código se le llamó "Codigo_Principal.c" y está diseñado para calcular la matriz de Lyapunov tipo retardada, mandando a llamar a las funciones desarrolladas en "funciones.h" y, posteriormente, trata los datos para enviarlos a CoolTerm.

3.2.1. Archivo funciones.h

Como se explicó anteriormente, con el fin de conservar un mejor orden que a su vez permita tener mayor optimización para mejorar su integración en la tarjeta FRDM-KL46Z, las funciones usadas en el código principal fueron programadas en un archivo *.h* (header) donde se almacenaron y, gracias a que dicho archivo se incluye en el código principal *(#include <funciones.h>*), estas pueden ser mandadas a llamar fácilmente, mejorando tanto la estética como, a su vez, el orden en los códigos. Para visualizar el código completo (funciones.h), revise el **Apéndice B**.

En esta sección se presentan los códigos con las funciones más importantes para el cálculo de la matriz de Lyapunov tipo retardada, para consultar el resto de funciones, revise el **Apéndice A**.

3.2.1.1. Función productoKronecker

El cálculo del producto de Kronecker es necesario para el cálculo de la matriz de Lyapunov tipo retardada, pues representa la ecuación en su forma vectorizada, el producto de Kronecker permite construir matrices de dimensión ampliada conservando la estructura original del sistema. En la Figura 3.4 se presenta la lógica principal utilizada para el desarrollo de la función producto de Kronecker basado en la ecuación (2.32). La expresión i/n representa una división entera, mientras que $i\%n$ representa la operación módulo.

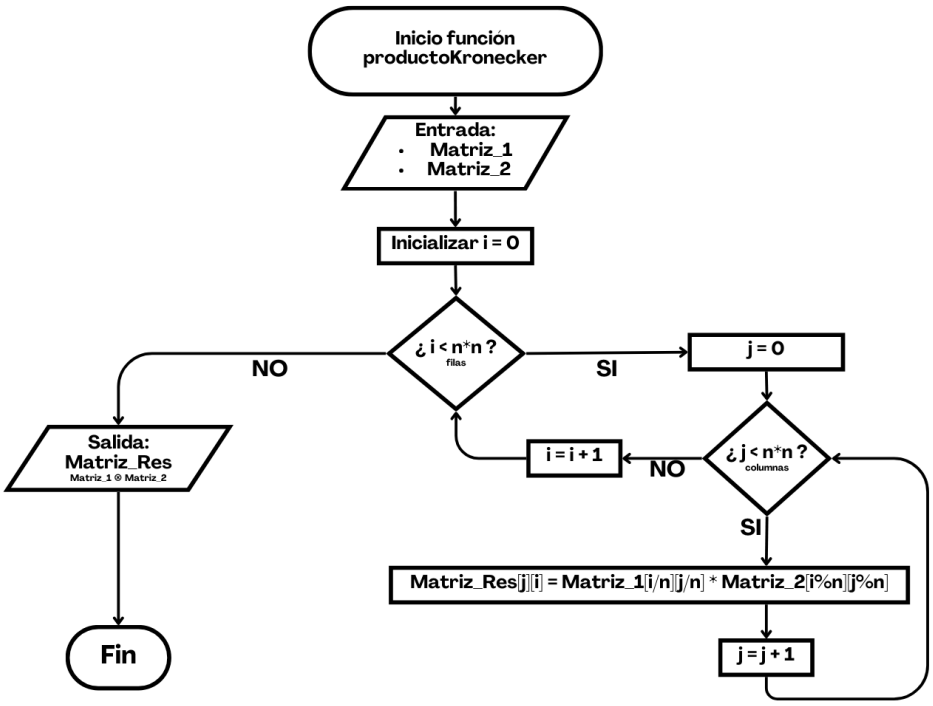


Figura 3.4: Diagrama de flujo de la función productoKronecker.

Esta función recibe como entradas dos matrices cuadradas de dimensiones $n \times n$ y devuelve en su salida una matriz de dimensión $n^2 \times n^2$ que representa el producto de Kronecker de ambas.

Código de la función productoKronecker.

```
void productoKronecker(float Matriz_1[n][n], float
Matriz_2[n][n], float Matriz_Res[n*n][n*n]) {
    for (int i = 0; i < n*n; i++) {
        for (int j = 0; j < n*n; j++) {
            Matriz_Res[i][j] = Matriz_1[i / n][j / n] *
Matriz_2[i % n][j % n];
        }
    }
}
```

Donde `Matriz_1[n][n]` y `Matriz_2[n][n]` (de entrada) son matrices de $n \times n$, y `Matriz_Res[i][j]` (de salida) es una matriz de $n^2 \times n^2$, es la matriz resultado del producto de Kronecker Primero se inicia un ciclo `for i`, que recorrerá las filas de la matriz de salida, después se inicia el ciclo `for j`, que recorrerá las columnas de la matriz de salida.

En cada iteración, `Matriz_1[i / n][j / n]` y `Matriz_2[i % n][j % n]` acceden a las posiciones correspondientes de las matrices de entrada mediante divisiones y módulos, es decir, `[i / n][j / n]` indican la fila y columna de la `Matriz_1` que se usará, por ejemplo, si $i =$

2 y $n = 2$ entonces $i / n = 1$, lo que quiere decir que se está usando la segunda fila de `Matriz_1`.

El operador módulo (%) devuelve el residuo de una división entera, esto permite generar un patrón cíclico de 0 a $n - 1$ entonces `[i % n][j % n]` indican la fila y columna de la `Matriz_2`, por ejemplo, si $i = 3$ y $n = 2$, entonces $i \,\%\, n = 1$, lo que indica que se está accediendo a la segunda fila de `Matriz_2`, mientras que si $j = 2$, entonces $j \,\%\, n = 0$, lo que indica que se está accediendo a la primera columna de `Matriz_2`, la multiplicación de estos 2 valores se guarda en la `Matriz_Res[i][j]`.

3.2.1.1.1. Ejemplo de uso

Supóngase que se requiere calcular el producto de Kronecker de la matriz A^T que se muestra en la ecuación (3.1) con la matriz identidad I que se muestra en la ecuación (3.2):

$$A^T = \begin{bmatrix} 0 & -1 \\ 1 & -2 \end{bmatrix}, \tag{3.1}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{3.2}$$

La función ejecuta un doble barrido en la matriz resultado con los ciclos `for i` y `for j`, también en cada iteración se calculan las posiciones relativas en A y I usando los operadores de división y módulo, de esta manera se va construyendo la matriz de salida.

Quedando la matriz de salida como se muestra en la ecuación (3.3).

$$A \otimes I \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & -2 & 0 \\ 0 & 1 & 0 & -2 \end{bmatrix}. \tag{3.3}$$

3.2.1.2. Función norma1

La norma 1 (2.35) de una matriz se utiliza como un criterio de escalamiento, garantizando la estabilidad numérica en la aproximación de Padé a la matriz L que es el sistema vectorizado para el cálculo de la matriz de Lyapunov tipo retardada. En la Figura 3.5, se muestra la lógica principal utilizada para el desarrollo de la función norma1.

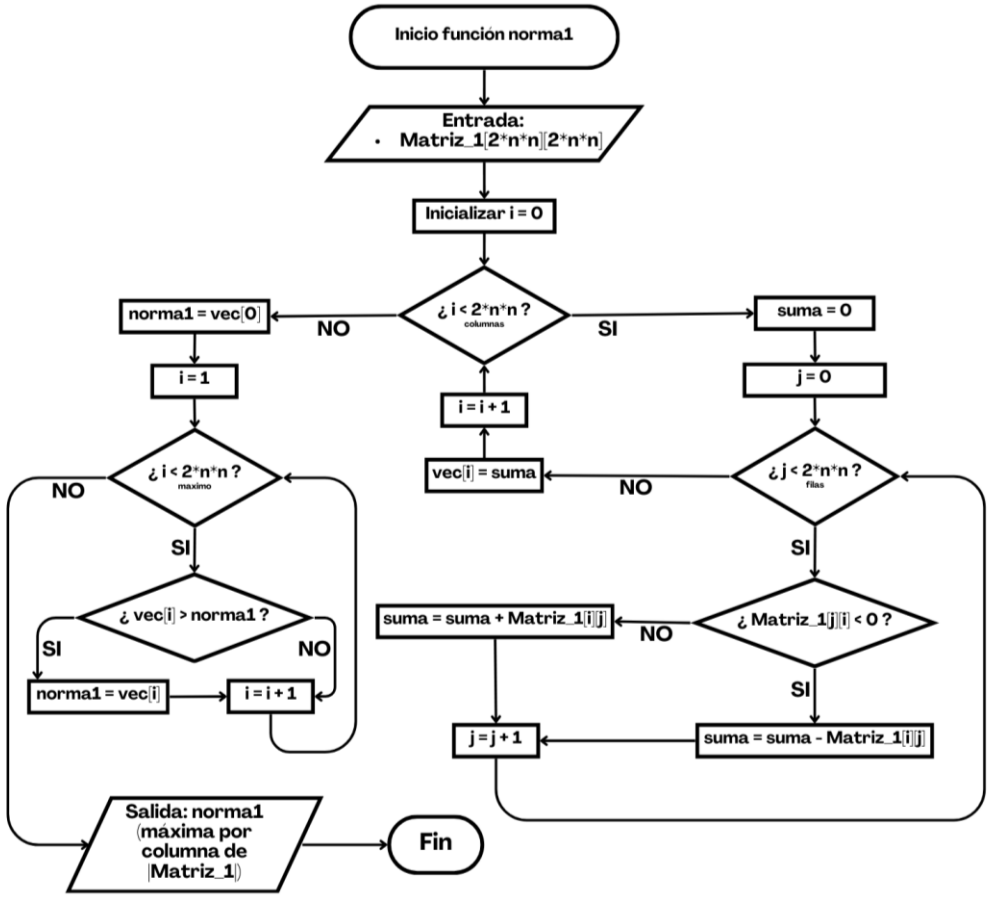


Figura 3.5: Diagrama de flujo de la función norma1.

Código de la función norma1:

```
float norma1(float Matriz_1[2*n*n][2*n*n]) {
      float suma, norma1;
      float vec[2*n*n];
      for (int i = 0; i < 2*n*n; i++){
            suma=0;
         for (int j = 0; j < 2*n*n; j++)
                           {
                           if (Matriz_1[j][i]<0)
                                 suma=suma-Matriz_1[j][i];
                           else
                                 suma=suma+Matriz_1[j][i];
                           }
                           vec[i]=suma;
                     }
      norma1=vec[0];
      for (int i = 1; i < 2*n*n; i++){
            if (vec[i]>norma1)
                  norma1=vec[i];
      }
      return norma1;
```

```
}
```

Donde `Matriz_1[2*n*n][2*n*n]` es la matriz de entrada de dimensión ampliada, `vec[2*n*n]` es un arreglo auxiliar en donde se almacenan las sumas por columna y `norma1` es el valor máximo de las sumas. Inicia con un ciclo `for i` que recorre las columnas de la matriz y, dentro de él, un ciclo `for j` que recorre las filas para sumar los valores absolutos por columna. Para identificar esto, la línea `suma=suma-Matriz_1[j][i];` lo convierte a positivo si es negativo, restándolo de `suma`, forzando una multiplicación de signos para generar la suma, y si es positivo, la línea `suma=suma+Matriz_1[j][i];` lo sumará directamente.

Después, se inicializa la variable norma1 con el primer valor de `vec[0]` como base para comparar, y el ciclo `for i` recorrerá los demás valores de `vec[]` y en cada iteración las líneas `if (vec[i]>norma1)` y `norma1=vec[i];` compara si el valor actual de `vec[i]` es mayor que el guardado en `norma1` y siempre que sea así se actualizará el valor de `norma1` y al final nos la retornará con el valor máximo encontrado.

3.2.1.2.1. Ejemplo de uso

Supóngase que se necesita calcular la Norma 1 de la matriz L que nos servirá para calcular el criterio de escalabilidad numérica de Padé, para la construcción de la matriz $L \in \mathbb{R}^{8\times 8}$:

$$L = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 \\ 1 & 0 & -2 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & -2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & -1 & 0 & 0 & 1 & 2 \end{bmatrix}. \tag{3.4}$$

Donde `vec` tiene los valores: 2, 2, 4, 4, 1, 3, 3, 5, por lo tanto, la Norma 1 de L (3.4) es 5.

3.2.1.3. Función summat2

En el caso particular de matrices de tamaño $2n^2 \times 2n^2$, que es el tamaño requerido para representar las matrices que intervienen en la solución de la matriz de Lyapunov tipo retardada, fue necesario desarrollar a la función summat2 además de la función summat, pues solo trabaja con matrices de tamaño $n \times n$. En la Figura 3.6 se muestra la lógica principal utilizada para el desarrollo de la función `summat2`.

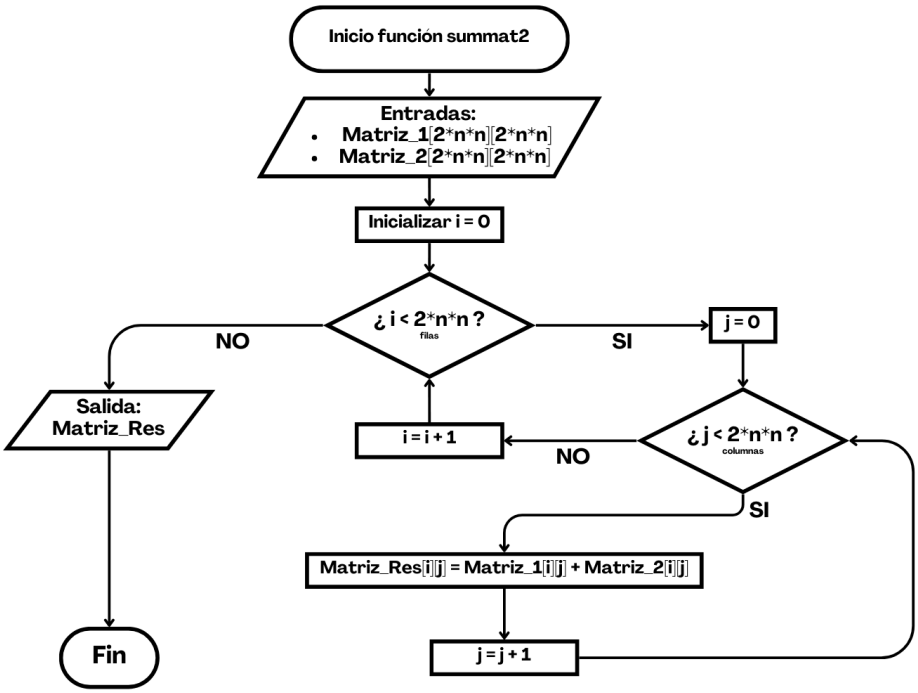


Figura 3.6: Diagrama de flujo de la función summat2.

Código de la función summat2:

```
void summat2(float Matriz_1[2*n*n][2*n*n], float
Matriz_2[2*n*n][2*n*n], float Matriz_Res[2*n*n][2*n*n]) {
    for (int i = 0; i < 2*n*n; i++) {
        for (int j = 0; j < 2*n*n; j++) {
            Matriz_Res[i][j] = Matriz_1[i][j] + Matriz_2[i][j];
        }
    }
}
```

Donde `Matriz_1[2*n*n][2*n*n]` y `Matriz_2[2*n*n][2*n*n]` son matrices de entrada, es decir, las matrices que se van a sumar, y `Matriz_Res[i][j]` es la matriz de salida, donde se almacenan los valores resultantes de la suma. Primero inicia un ciclo `for i`, que recorre las filas de ambas matrices y dentro de él, se ejecuta un ciclo `for j` para recorrer las columnas de ambas matrices y en cada iteración, la línea de código `Matriz_Res[i][j] = Matriz_1[i][j] + Matriz_2[i][j];` va sumando los elementos en cada iteración.

3.2.1.3.1. Ejemplo de uso

Supóngase que se necesita sumar dos matrices simuladas de tamaño $2n^2$ x $2n^2$ para realizar una iteración del método basado en la aproximación de Padé, donde la matriz NP (3.5) contiene la

suma acumulada de las iteraciones anteriores y la matriz $Aux2$ (3.6) contiene el término actual ponderado.

$$Aux2 = \begin{bmatrix} 0.00003 & 0 & -0.00003 & -0.00001 & 0 & 0 & 0.00006 & 0.00004 \\ -0.00001 & 0.00004 & -0.00001 & -0.00003 & 0 & 0 & -0.00006 & -0.00001 \\ 0.00004 & 0 & -0.00007 & 0.00001 & 0 & 0 & 0.00006 & 0.00001 \\ 0.00001 & 0.00003 & -0.00004 & -0.00001 & 0 & 0 & 0 & 0.00001 \\ -0.00003 & 0.00003 & 0.00001 & -0.00001 & -0.00003 & -0.00004 & 0.00001 & -0.00001 \\ 0.00003 & -0.00003 & 0.00001 & -0.00001 & 0.00004 & 0.00006 & -0.00003 & 0.00008 \\ -0.00001 & 0.00001 & 0 & 0 & 0 & 0 & -0.00004 & -0.00003 \\ 0 & 0 & 0.00001 & -0.00001 & 0 & 0 & 0.00001 & 0.00003 \end{bmatrix}, \quad (3.5)$$

$$NP = \begin{bmatrix} 0.99889 & 0 & -0.04778 & 0 & 0 & 0 & -0.05111 & 0.00111 \\ 0 & 0.99889 & -0.00111 & -0.04667 & 0 & 0 & -0.00111 & -0.05333 \\ 0.04778 & 0 & 0.90333 & 0 & 0 & 0 & 0.04667 & -0.00111 \\ 0 & 0.04778 & 0..00111 & 0.90222 & 0 & 0 & 0.00111 & 0.04889 \\ -0.00111 & 0.00111 & 0 & 0 & 0.99889 & -0.05222 & 0 & 0 \\ 0.05222 & -0.05222 & -0.00111 & 0.00111 & 0.05222 & 1.10333 & -0.00111 & 0.00111 \\ 0 & 0 & -0.00111 & 0.0111 & 0 & 0 & 0.99889 & -0.05222 \\ 0.00111 & -0.00111 & 0.05 & -0.05 & 0 & 0 & 0.05333 & 1.10222 \end{bmatrix}, \quad (3.6)$$

da como resultado:

$$NP\ acumulada \coloneqq \begin{bmatrix} 0.99892 & 0 & -0.04781 & -0.00001 & 0 & 0 & -0.05105 & 0.00115 \\ -0.00001 & 0.99893 & -0.00112 & -0.04670 & 0 & 0 & -0.00117 & -0.05334 \\ 0.04782 & 0 & 0.90326 & 0.00001 & 0 & 0 & 0.04673 & -0.00110 \\ 0.00001 & 0.04781 & -0.00003 & 0.90221 & 0 & 0 & 0.00111 & 0.04890 \\ -0.00114 & 0.00114 & 0.00001 & -0.00001 & 0.99886 & -0.05226 & 0.00001 & -0.00001 \\ 0.05225 & -0.05225 & -0.00110 & 0.00110 & 0.05226 & 1.10339 & -0.00114 & 0.00119 \\ -0.00001 & 0.00001 & -0.00111 & 0.01110 & 0 & 0 & 0.99885 & -0.05225 \\ 0.00111 & -0.00111 & 0.05001 & -0.05001 & 0 & 0 & 0.05334 & 1.10225 \end{bmatrix}. \quad (3.7)$$

3.2.1.4. Función mulmat2

La multiplicación de matrices es otra operación fundamental para el cálculo de la matriz de Lyapunov tipo retardada, se hace específicamente con matrices de $2n^2 \times 2n^2$ en donde se utiliza en el método de aproximación basado en Padé, donde es requerida para calcular las potencias sucesivas de la matriz escalada L_m, es decir: $L_m^j = L_m^{j-1} \cdot L_m$. La Figura 3.7 muestra la lógica principal utilizada para desarrollar la función mulmat2.

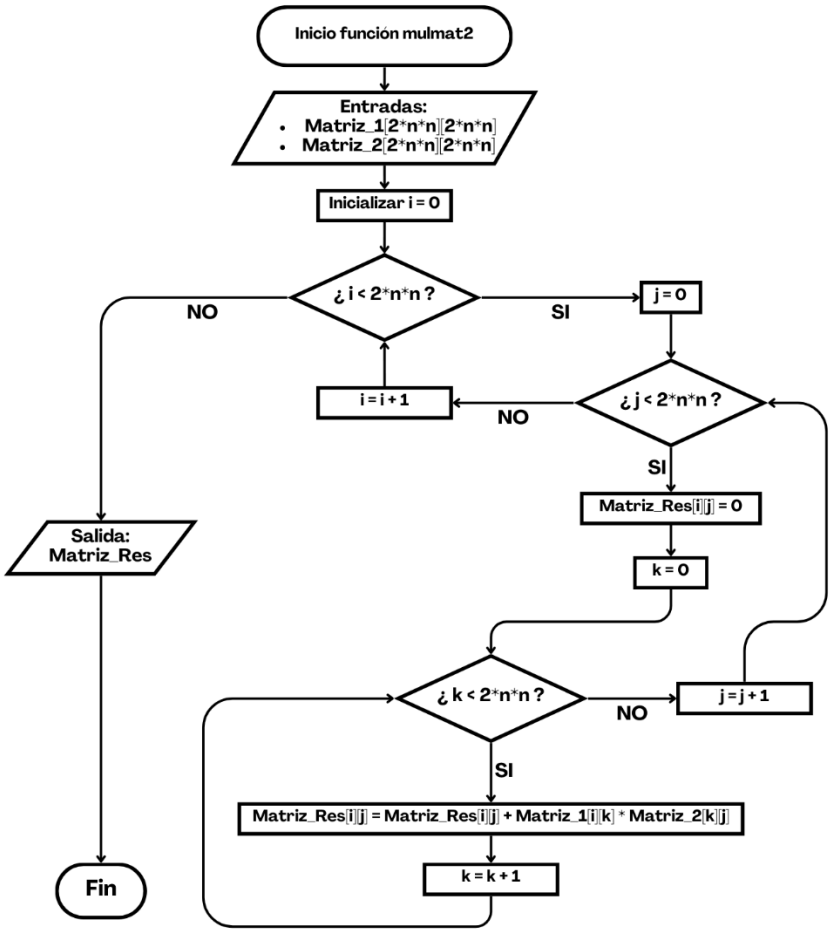


Figura 3.7: Diagrama de flujo de la función mulmat2.

Código de la función mulmat2:

```
void mulmat2(float Matriz_1[2*n*n][2*n*n], float
Matriz_2[2*n*n][2*n*n], float Matriz_Res[2*n*n][2*n*n]) {
    for (int i = 0; i < 2*n*n; i++) {
        for (int j = 0; j < 2*n*n; j++) {
            Matriz_Res[i][j] = 0;
            for (int k = 0; k < 2*n*n; k++)
                Matriz_Res[i][j] += Matriz_1[i][k] *
Matriz_2[k][j];
        }
    }
}
```

Donde `Matriz_1[2*n*n][2*n*n]` y `Matriz_2[2*n*n][2*n*n]` son las matrices de entrada de tamaño $2n^2 \times 2n^2$ y `Matriz_Res[2*n*n][2*n*n]` es la matriz de salida del mismo tamaño, después se inician 3 ciclos for anidados, primero el ciclo `for i` para recorrer las filas de la matriz resultado, y después el ciclo `for j` que recorre las columnas. La línea de código

`Matriz_Res[i][j] = 0;` hace que, antes de calcular el valor de la posición `[i][j]` se inicie en 0.

Y el tercer ciclo, `for k`, realiza la suma de productos entre la fila i de `Matriz_1` y la columna j de `Matriz_2` y va almacenando los resultados en `Matriz_Res`.

3.2.1.4.1. Ejemplo de uso

Supóngase que se está ejecutando la iteración 1 de la aproximación basada en Padé, por lo que se necesita calcular L_m^2, teniendo como base a L_m como:

$$L_m = \begin{bmatrix} 0.1 & 0 & 0.01 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0.01 & 0 & 0 & 0 & 0 \\ -0.01 & 0 & -0.02 & 0 & 0 & 0 & 0 & 0.01 \\ 0 & -0.01 & 0 & -0.02 & 0 & 0 & 0 & -0.01 \\ 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.01 & 0.02 & 0 & 0 \\ -0.01 & 0 & 0.01 & 0 & 0 & 0 & 0.1 & 0 \\ 0 & -0.01 & 0 & 0.01 & 0 & 0 & 0 & 0.02 \end{bmatrix}. \quad (3.8)$$

Al multiplicar L_m por L_m se eleva, obteniendo a L_m^2.

$$L_m^2 = \begin{bmatrix} 0.00990 & 0 & 0.00080 & 0 & 0 & 0 & 0 & 0.00010 \\ 0 & 0.00990 & 0 & 0.00080 & 0 & 0 & 0 & -0.00010 \\ -0.00080 & -0.00010 & 0.00030 & 0.00010 & 0 & 0 & 0 & 0 \\ 0 & -0.00070 & 0 & 0.00020 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.00200 & 0.00040 & 0 & 0 \\ -0.00100 & 0 & 0.00120 & 0 & 0 & 0 & 0.01000 & 0 \\ 0 & -0.00100 & 0 & 0.00120 & 0 & 0 & 0 & 0.00060 \end{bmatrix}. \quad (3.9)$$

3.2.1.5. Función inversa

La función inversa calcula la matriz inversa de una matriz de tamaño $n \times n$, utilizando el método clásico de Gauss-Jordan. En la Figura 3.8 se muestra la lógica principal utilizada para el desarrollo de la función inversa.

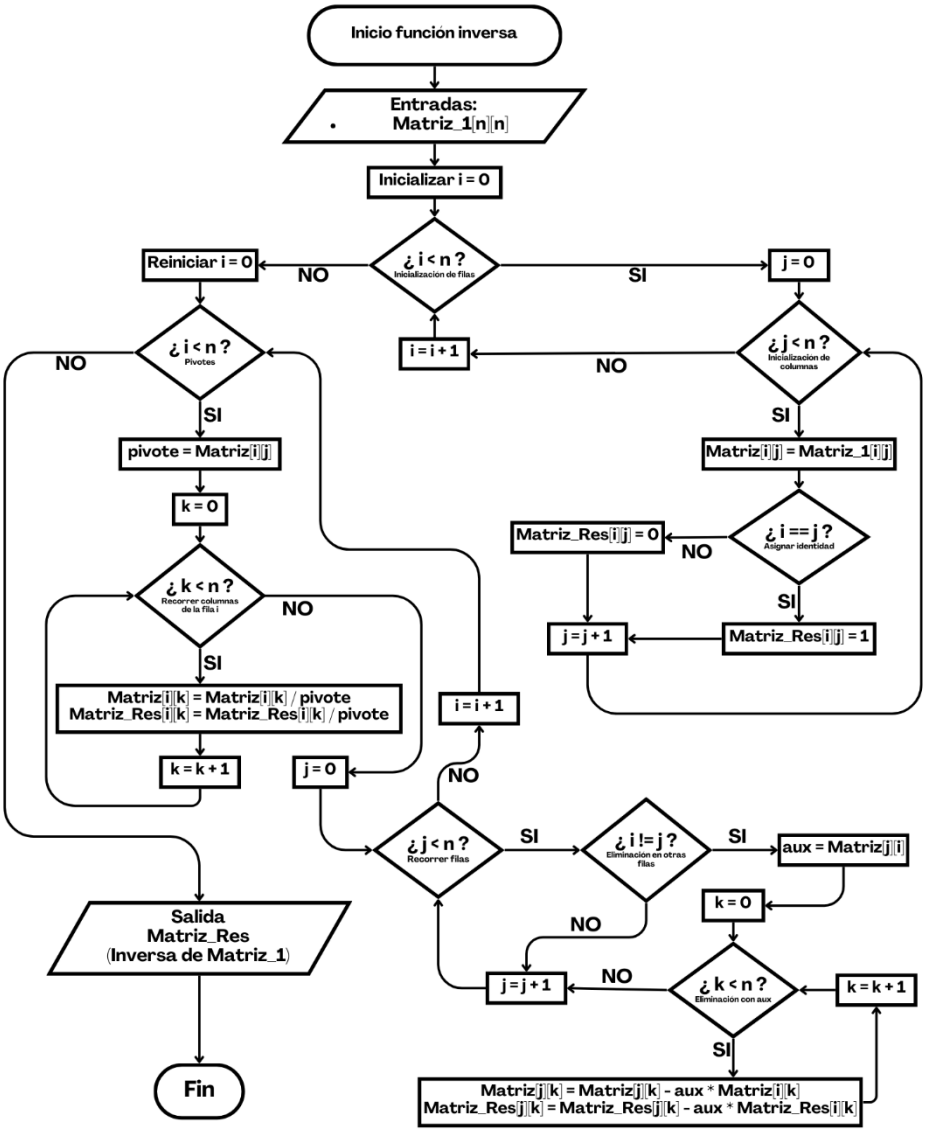


Figura 3.8: Diagrama de flujo de la función inversa.

Código de la función inversa:

```
void inversa(float Matriz_1[n][n], float Matriz_Res[n][n]) {
     float Matriz[n][n];
     float pivote;
     float aux;
     for (int i=0;i< n;i++)
               for (int j=0;j< n;j++){
                 Matriz[i][j]=Matriz_1[i][j];
                  if (i!=j)
                       Matriz_Res[i][j]=0;
                  else
                       Matriz_Res[i][j]=1;
               }
```

```
        for (int i=0;i< n;i++){
                pivote=Matriz[i][i];
                for(int k=0;k<n;k++){
                    Matriz[i][k]=Matriz[i][k]/pivote;
                    Matriz_Res[i][k]=Matriz_Res[i][k]/pivote;
                }
                for(int j=0; j<n; j++){
                    if (i!=j)
                    {
                        aux=Matriz[j][i];
                        for (int k=0; k<n; k++){
                            Matriz[j][k]=Matriz[j][k]-
aux*Matriz[i][k];
                            Matriz_Res[j][k]=Matriz_Res[j][k]-
aux*Matriz_Res[i][k];
                        }
                    }
                }
            }
}
```

El código declara las variables `Matriz[n][n]`, que será una copia de la matriz de entrada, `pivote`, para guardar el valor del elemento diagonal que se va a normalizar, y `aux`, que se usará para calcular factores multiplicativos en la eliminación de otros elementos de la columna actual.

```
for (int i=0;i< n;i++)
                for (int j=0;j< n;j++){
                  Matriz[i][j]=Matriz_1[i][j];
```

Con estos ciclos anidados, de `for i` recorrerá las filas y `for j` recorrerá las columnas y copiará cada elemento en cada iteración de la matriz `Matriz_1`, en la variable `Matriz`, esto porque la matriz original se modificará durante el proceso de inversión, por lo que es mejor trabajar con una copia de la matriz de entrada.

```
if (i!=j)
                        Matriz_Res[i][j]=0;
                   else
                        Matriz_Res[i][j]=1;
                }
```

Este condicional `if`, permite simultáneamente iniciar a `Matriz_Res`, como una matriz identidad, ya que asignará valor de 1 cuando $i = j$, y 0 cuando $i \neq j$, creando una matriz con 1 en la diagonal.

```
for (int i=0;i< n;i++){
                pivote=Matriz[i][i];
                for(int k=0;k<n;k++){
                    Matriz[i][k]=Matriz[i][k]/pivote;
Matriz_Res[i][k]=Matriz_Res[i][k]/pivote;
}
```

Con el ciclo `for i`, se recorrerán todas las filas de `Matriz`, el índice `i` indica la fila pivote actual y se guarda el valor del elemento diagonal (pivote) de la fila actual.

Después, en el ciclo `for k`, se normaliza la fila `i`, dividiendo cada elemento de `Matriz`, por el valor del `pivote`, convirtiendo el elemento diagonal en 1.

```
for(int j=0; j<n; j++){
                    if (i!=j)
                    {
                        aux=Matriz[j][i];
                        for (int k=0; k<n; k++){
                            Matriz[j][k]=Matriz[j][k]-
aux*Matriz[i][k];
                            Matriz_Res[j][k]=Matriz_Res[j][k]-
aux*Matriz_Res[i][k];
                        }
```

Se inicia un ciclo `for j,` que recorre todas las demás filas donde $j \neq i$ para hacer 0 todos los elementos de la columna `i`, excepto el de la fila `i` que ya fue normalizado.

Después, dentro de la variable `aux,` se guarda el valor de `Matriz` que se quiere eliminar `[j][i]` y, con el ciclo `for k`, se realiza la operación de eliminación fila por fila, a la fila `j`, se le resta la fila `i` multiplicada por `aux`, tanto en `Matriz_Res` como en `Matriz`, esta operación convierte el elemento que se está iterando en 0 y transforma las demás posiciones de la fila de forma que `Matriz_Res` se mantiene equivalente.

Al finalizar, `Matriz` queda convertida en una matriz identidad y `Matriz_Res` contiene la matriz inversa de `Matriz_1`.

3.2.1.5.1. Ejemplo de uso

Supóngase que se requiere obtener la matriz inversa de la matriz: $M = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$

Se inicia Matriz_Res como: $Matriz\_Res = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ y la función itera como se muestra en la Tabla 3.2.

Iteración	Acción	Matriz	Matriz_Res
0	Inicialización	$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
1	Normalización de fila 0 dividiendo por el pivote 1	$\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
2	Elimina el "2" de la fila 1 usando (2*F1 – F2)	$\begin{bmatrix} 1 & 3 \\ 0 & -2 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}$
3	Normalización de la fila 1 dividiendo entre el pivote (-2)	$\begin{bmatrix} 1 & 3 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 1 & -0.5 \end{bmatrix}$
4	Se elimina el "3" de la fila 0 usando (3*F2-F1)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -2 & 1.5 \\ 1 & -0.5 \end{bmatrix}$

Tabla 3.2: Iteraciones del ejemplo de uso de la función inversa.

Quedando la matriz inversa como:

$$M^{-1} = \begin{bmatrix} -2 & 1.5 \\ 1 & -0.5 \end{bmatrix}. \tag{3.10}$$

3.2.1.6. Función inversa2

La función inversa2 está hecha para calcular la matriz inversa de una matriz de tamaño $2n^2 \times 2n^2$, al igual que la función inversa, utiliza el método clásico de Gauss-Jordan. En la Figura 3.9 se muestra la lógica principal utilizada para el desarrollo de la función.

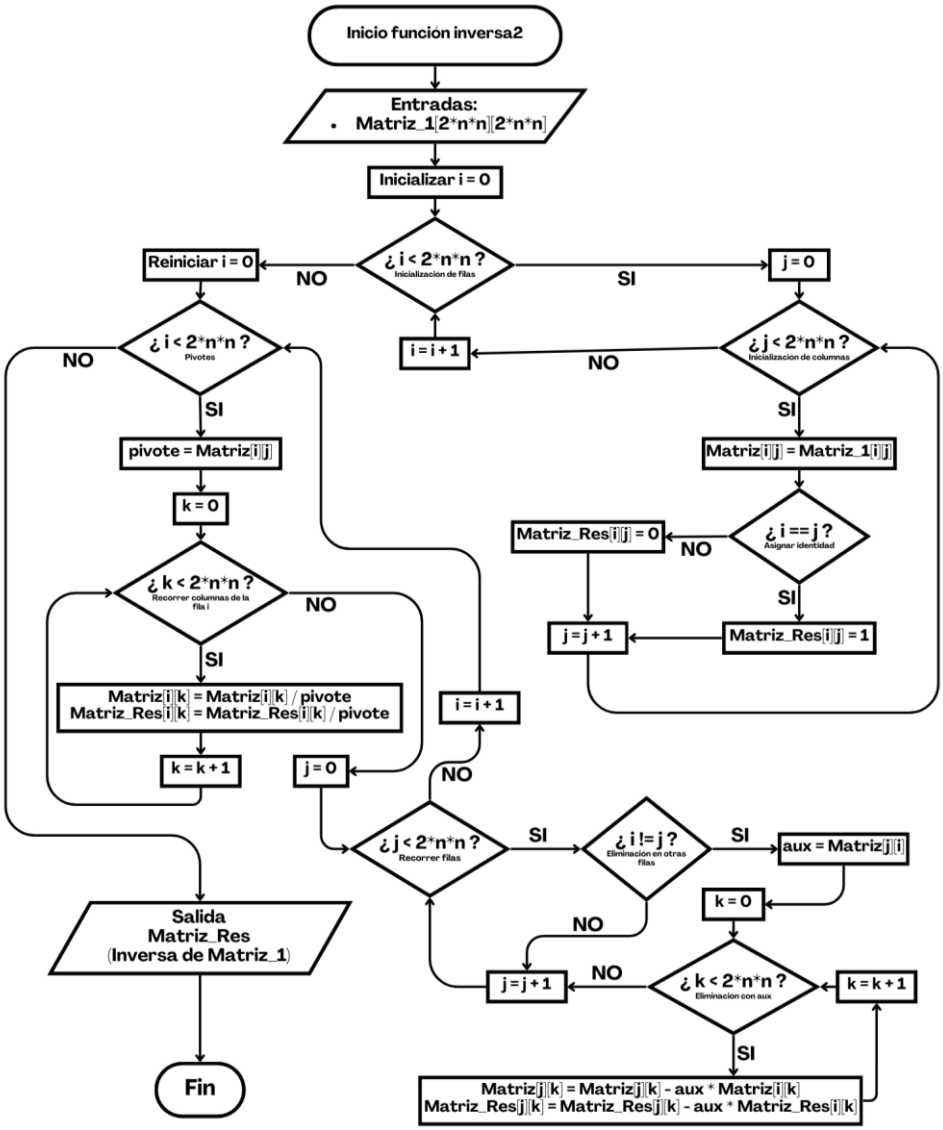


Figura 3.9: Diagrama de flujo de la función inversa2.

Código de la función inversa2:

```
void inversa2(float Matriz_1[2*n*n][2*n*n], float
Matriz_Res[2*n*n][2*n*n]) {
     float Matriz[2*n*n][2*n*n];
     float pivote;
     float aux;
     for (int i=0;i<2*n*n;i++)
               for (int j=0;j< 2*n*n;j++){
                 Matriz[i][j]=Matriz_1[i][j];
                  if (i!=j)
                      Matriz_Res[i][j]=0;
                  else
                      Matriz_Res[i][j]=1;
               }
```

```
        for (int i=0;i< 2*n*n;i++){
                pivote=Matriz[i][i];
                for(int k=0;k<2*n*n;k++){
                    Matriz[i][k]=Matriz[i][k]/pivote;
                    Matriz_Res[i][k]=Matriz_Res[i][k]/pivote;
                }
                for(int j=0; j<2*n*n; j++){
                    if (i!=j)
                    {
                        aux=Matriz[j][i];
                        for (int k=0; k<2*n*n; k++){
                            Matriz[j][k]=Matriz[j][k]-
aux*Matriz[i][k];
                            Matriz_Res[j][k]=Matriz_Res[j][k]-
aux*Matriz_Res[i][k];
                        }
                    }
                }
            }
}
```

Donde `Matriz_1` es la matriz de entrada de dimensión $2n^2 \times 2n^2$, y `Matriz_Res` es la matriz de salida, donde se guardará la inversa, y `Matriz`, que se utilizará como copia de `Matriz_1`. Posteriormente, se declaran las variables `pivote`, que se utiliza para almacenar el valor del elemento diagonal de la fila actual, y `aux`, que se utiliza para guardar el valor del elemento que se requiere eliminar en una fila diferente a la actual.

```
float Matriz[2*n*n][2*n*n];
     float pivote;
     float aux;
```

El primer ciclo for anidado copia y pega los elementos de `Matriz_1` en `Matriz` con el ciclo `for i` para las filas y el ciclo `for j` para las columnas. Simultáneamente, en cada iteración, a través de un condicional `if`, se construye una matriz identidad en `Matriz_Res`; evaluando si `i = j` asigna un 1 (si es diagonal) y 0 si no se cumple la condición.

```
for (int i=0;i<2*n*n;i++)
                for (int j=0;j< 2*n*n;j++){
                  Matriz[i][j]=Matriz_1[i][j];
                   if (i!=j)
                       Matriz_Res[i][j]=0;
                   else
                       Matriz_Res[i][j]=1;
                }
```

Con esto preparado, se inicia el proceso de inversión de Gauss-Jordan. Se inicia el proceso fila por fila, donde con el ciclo `for i` obtiene el pivote que debe convertirse en 1 y después normaliza la fila dividiendo todos los elementos entre el pivote, lo que convierte al pivote en 1 y ajusta los demás elementos para mantener la equivalencia tanto en `Matriz` como en `Matriz_Res`.

```
for (int i=0;i< 2*n*n;i++){
            pivote=Matriz[i][i];
            for(int k=0;k<2*n*n;k++){
                Matriz[i][k]=Matriz[i][k]/pivote;
```

Con la fila i normalizada, el ciclo for `j` elimina los demás elementos de la columna `i` (donde $j \neq i$) y `aux=Matriz[i][j]` toma el valor de esa posición (fuera de la diagonal) que se convertirá en 0.

```
            for(int j=0; j<2*n*n; j++){
                      if (i!=j)
                      {
                          aux=Matriz[j][i];
```

Para la eliminación fila a fila, se crea un ciclo `for k`, que actualizará toda la fila `j`, restándole la fila `i` multiplicada por `aux`, resultando en que el valor de la columna `i` y fila `j` se vuelve 0, esto se aplica tanto en `Matriz` como en `Matriz_Res`.

```
for (int k=0; k<2*n*n; k++){
                                Matriz[j][k]=Matriz[j][k]-
aux*Matriz[i][k];
                                Matriz_Res[j][k]=Matriz_Res[j][k]-
aux*Matriz_Res[i][k];
                           }
                      }
                }
             }}
```

3.2.1.7. Función MatrizPotencia

La función MatrizPotencia calcula las potencias de una matriz (2.62), es útil para la matriz de Lyapunov tipo retardada, pues durante la aproximación de Padé requiere elevar matrices intermedias a una potencia. En la Figura 3.10 se muestra la lógica principal utilizada para el desarrollo de la función MatrizPotencia.

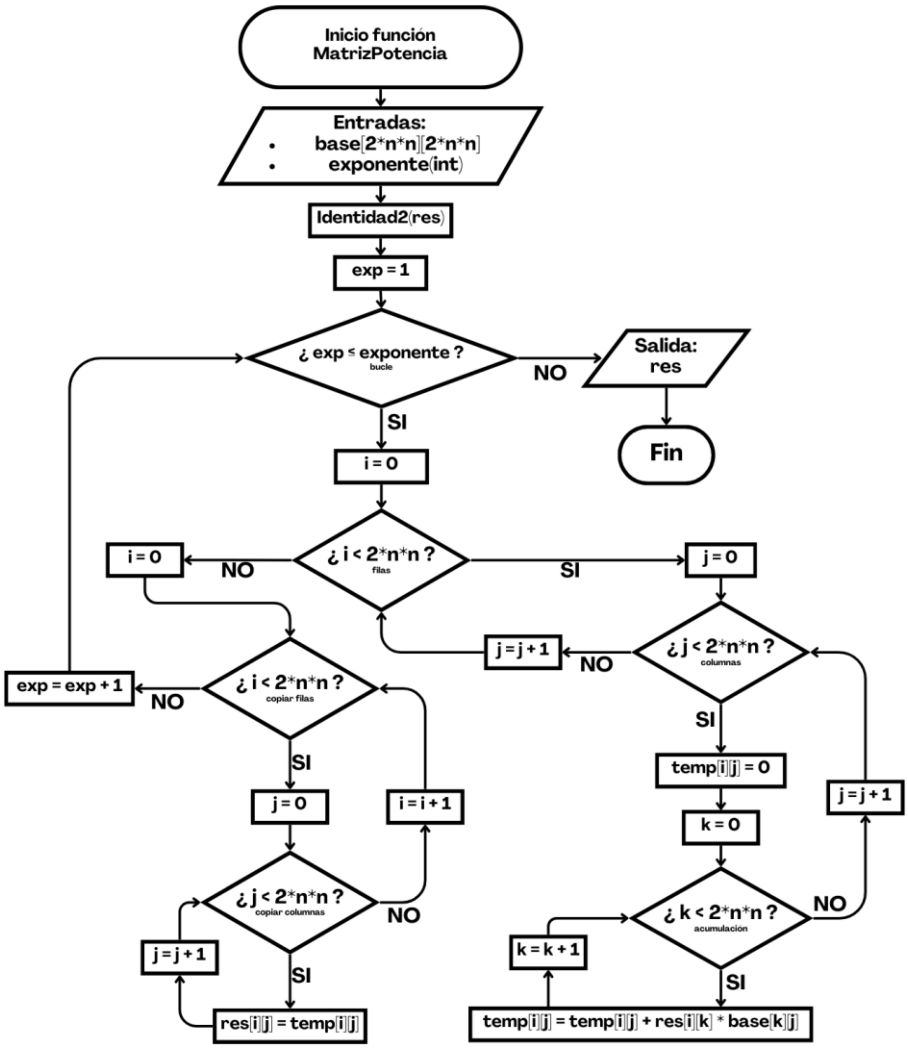


Figura 3.10: Diagrama de flujo de la función MatrizPotencia.

Código de la función MatrizPotencia:

```
void MatrizPotencia(float res[2*n*n][2*n*n], float
base[2*n*n][2*n*n], int exponente) {
    float temp[2*n*n][2*n*n];
    Identidad2(res);
    for (int exp = 1; exp <= exponente; exp++) {
        for (int i = 0; i < 2*n*n; i++) {
            for (int j = 0; j < 2*n*n; j++) {
                temp[i][j] = 0;
                for (int k = 0; k < 2*n*n; k++) {
                    temp[i][j] = temp[i][j] + res[i][k] *
base[k][j];
                }
            }
```

```
            }
            for (int i = 0; i < 2*n*n; i++) {
                for (int j = 0; j < 2*n*n; j++) {
                    res[i][j] = temp[i][j];
                }
            }
        }
    }
```

Donde `base` es una matriz de $2n^2 \times 2n^2$ que se va a elevar a una potencia (de entrada), `exponente` es un número entero que indica la potencia a la que se desea elevar la matriz, `res` es la matriz resultado de la operación (de salida) y `temp` es una matriz auxiliar temporal.

Primero, con la función `identidad2`, se inicia a `res` como la matriz identidad de $2n^2$ x $2n^2$, debido a que cualquier matriz elevada a la 0 es la matriz identidad.

```
    void MatrizPotencia(float res[2*n*n][2*n*n], float
    base[2*n*n][2*n*n], int exponente) {
        float temp[2*n*n][2*n*n];
        Identidad2(res);
```

Después inicia 4 ciclos for anidados, donde el ciclo `for exp` se ejecuta desde 1 hasta el valor alojado en exponente para elevar la matriz a la potencia requerida, los ciclos `for i` y `for j` se ejecutan para conocer la posición en `temp` y `for k` ejecuta la operación de la línea `temp[i][j] = temp[i][j] + res[i][k] * base[k][j];`.

```
    for (int exp = 1; exp <= exponente; exp++) {
            for (int i = 0; i < 2*n*n; i++) {
                for (int j = 0; j < 2*n*n; j++) {
                    temp[i][j] = 0;
                    for (int k = 0; k < 2*n*n; k++) {
                        temp[i][j] = temp[i][j] + res[i][k] *
    base[k][j];
                    }
                }
            }
```

Por último, el resultado intermedio de la multiplicación guardado en `temp` se copia a res.

```
            for (int i = 0; i < 2*n*n; i++) {
                for (int j = 0; j < 2*n*n; j++) {
                    res[i][j] = temp[i][j];
                }
            }
        }
```

```
}
```

3.2.1.7.1. Ejemplo de uso

Se desea calcular A^3 siendo A:

$$A = \begin{bmatrix} 1 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.11}$$

Definiendo a `int exponente = 3;` el código ejecutará la operación: $R = A \cdot A \cdot A = A^3$ quedando R como:

$$R = \begin{bmatrix} 1 & 0.3 & 0.03 & 0.001 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0.3 & 0.03 & 0.001 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.3 & 0.03 & 0.001 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0.3 & 0.03 & 0.001 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0.3 & 0.03 & 0.001 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0.3 & 0.03 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.12}$$

3.2.1.8. Función MatExpPad

La función MatExpPad está basada en la ecuación reescrita de las funciones de Padé (2.65). En la Figura 3.11 se presenta la lógica principal utilizada para desarrollar la función MatExpPad.

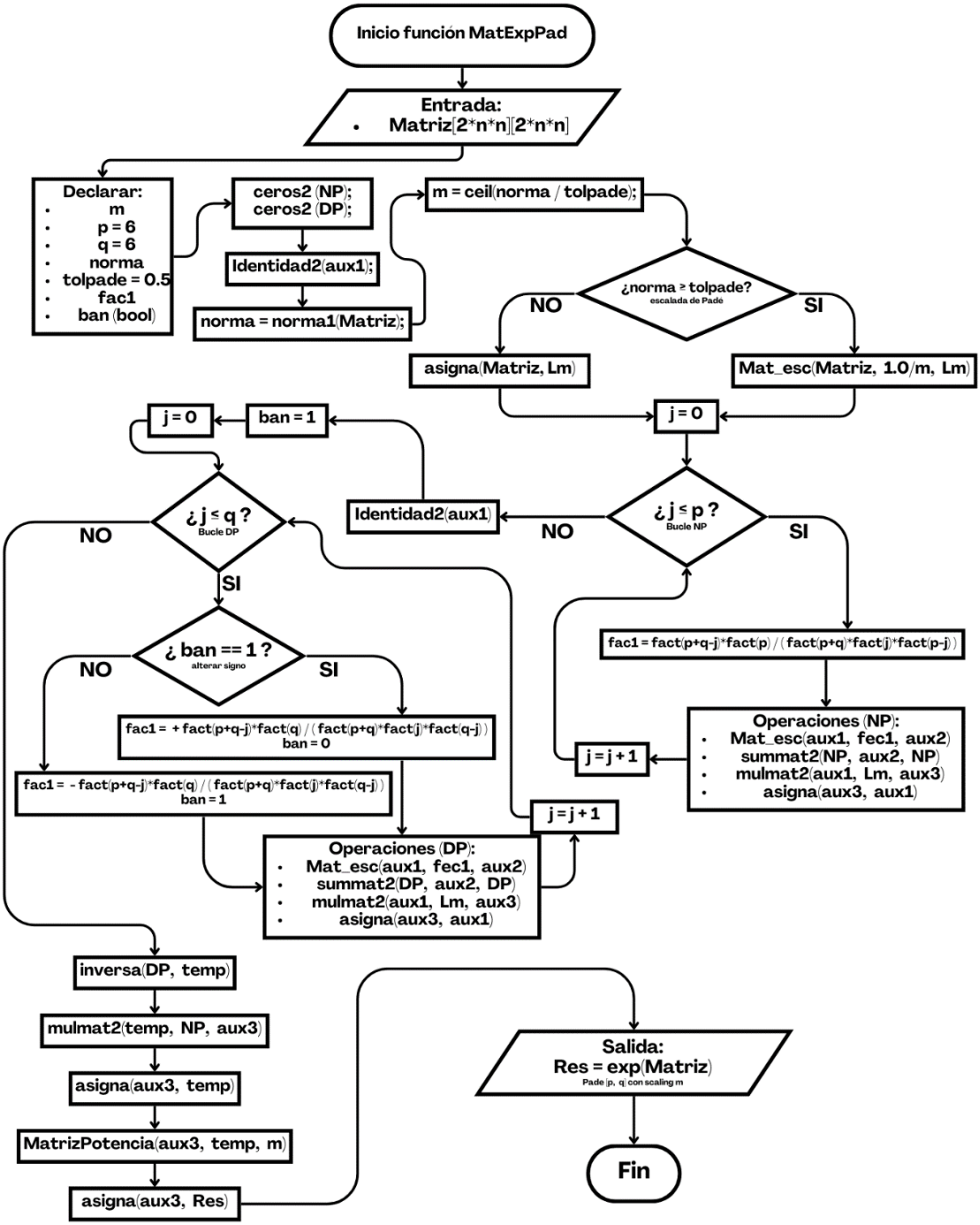


Figura 3.11: Diagrama de flujo de la función MatExpPad.

Código de la función MatExpPad:

```
void MatExpPad(float Matriz[2*n*n][2*n*n], float
Res[2*n*n][2*n*n]){

            int m;
```

```
          float norma, tolpade = 0.5;
          float fac1;
          int p = 6, q = 6;
          float Lm[2*n*n][2*n*n];
          float NP[2*n*n][2*n*n];
          float DP[2*n*n][2*n*n];
          float aux1[2*n*n][2*n*n];
          float aux2[2*n*n][2*n*n];
          float aux3[2*n*n][2*n*n];
          float temp[2*n*n][2*n*n];

          ceros2(NP);
          ceros2(DP);

          Identidad2(aux1);
          norma=norma1(Matriz);
          m = ceil(norma/tolpade);

          if (norma>=tolpade)
           {

                  Mat_esc(Matriz,1.0/m,Lm);
           }

          else
          {
            asigna(Matriz,Lm);
          }

          for (int j=0;j<=p;j++){
                 fac1 =  1.0*fact(p+q-
j)*fact(p)/(1.0*fact(p+q)*fact(j)*fact(p-j));
                 Mat_esc(aux1,fac1,aux2);
                 summat2(NP,aux2,NP);
                 mulmat2(aux1,Lm,aux3);
                 asigna(aux3,aux1);

          }

          Identidad2(aux1);
          bool ban=1;
          for (int j=0;j<=q;j++){
           if (ban){
                    fac1 = 1.0*fact(p+q-
j)*fact(q)/(1.0*fact(p+q)*fact(j)*fact(q-j));
                   ban=0;
           }
           else
           {
                  fac1 = -1.0*fact(p+q-
j)*fact(q)/(1.0*fact(p+q)*fact(j)*fact(q-j));
                 ban=1;
```

```
            }

                    Mat_esc(aux1,fac1,aux2);
                    summat2(DP,aux2,DP);
                    mulmat2(aux1,Lm,aux3);
                    asigna(aux3,aux1);
              }

          inversa2(DP,temp);
          mulmat2(temp,NP,aux3);
          asigna(aux3,temp);
          MatrizPotencia(aux3,temp,m);
          asigna(aux3,Res);
}
```

En esta función se utilizan varias funciones ya explicadas, por lo que se mencionarán las operaciones de las funciones que se mandaron a llamar.

Teniendo como matriz de entrada a `Matriz`, de dimensiones $2n^2 \times 2n^2$ y como salida esta `Res`, que es la matriz resultado e^A aproximada mediante Padé.

Después se declaran variables como:

- `m`: factor de escalado.
- `norma`: usada para calcular cuánto se debe escalar.
- `Tolpade`: umbral para decidir si se necesita escalar definido con 0.5.
- `Fac1`: almacena coeficientes temporales.
- `p y q`: orden de la aproximante de Padé definidos con 6.

Matrices temporales:

- `Lm`: matriz escalada.
- `NP`: Numerador de Padé.
- `DP`: denominador de Padé.
- `aux1, aux2, aux3, temp`: matrices temporales para cálculos intermedios.

Posteriormente, se inicializan las matrices `NP` Y `DP` en ceros con la función `ceros2`.

```
ceros2(NP);
ceros2(DP);
```

Después se inicializa a `aux1` como matriz identidad con la función `identidad2` para acumular potencias de `Lm`, después calcula la norma1 de `Matriz` con la función `norma1` para decidir cuántas veces debe escalar `m`.

```
Identidad2(aux1);
norma=norma1(Matriz);
m = ceil(norma/tolpade);
```

A través de un condicional `if`, se determina si se escala la matriz o si se usará sin escalar. Si es mayor al umbral, se escala la matriz con la función `Mat_esc` y, si no, se usará sin escalar usando la función `asigna` para copiar `Matriz` en `Lm`.

```
if (norma >= tolpade) {
    Mat_esc(Matriz, 1.0 / m, Lm);
} else {
    asigna(Matriz, Lm);
}
```

3.2.1.8.1. Cálculo de *NP*

Primero se calcula a c_j (2.69):

```
fac1 = 1.0 * fact(p + q - j) * fact(p) / (1.0 * fact(p + q) *
fact(j) * fact(q - j));
```

Entonces, el coeficiente de Padé para el numerador se calcula mediante combinaciones de factoriales, siguiendo la fórmula general del numerador de Padé de orden p, q. En el ejemplo $p = 6$ y $q = 6$, los coeficientes están equilibrados para mejor precisión.

- L^j es la potencia j-ésima de la matriz escalada L.

En cada iteración, escala L^j con su coeficiente `fac1` y suma el resultado a `NP` y multiplica `aux1` por `Lm` para actualizar la potencia para la próxima iteración y por.

```
    for (int j = 0; j <= p; j++) {
        fac1 = 1.0 * fact(p + q - j) * fact(p) / (1.0 * fact(p +
q) * fact(j) * fact(p - j));
```

```
        Mat_esc(aux1, fac1, aux2);
        summat2(NP, aux2, NP);
        mulmat2(aux1, Lm, aux3);
        asigna(aux3, aux1);
    }
```

3.2.1.8.2. Cálculo de *DP*

En la construcción de `DP` primero se inicializa la matriz `DP` en ceros con la función `zeros2` que será el acumulador que construirá la suma $\Sigma d_j \cdot L^j$.

Inicializa `aux1` como matriz identidad con la función `identidad2`, de tamaño $2n^2 \times 2n^2$ para iniciar con $L^0 = I$ para calcular a L^j y declara una bandera lógica `ban` para alternar el signo de cada término.

Para comenzar a construir a d_j (2.71), un ciclo `for j` inicia un bucle desde $= 1$ hasta $j = q$, construyendo cada término del denominador.

```
Identidad2(aux1);
         bool ban=1;
         for (int j=0;j<=q;j++)
```

Y que a través de un condicional `if` determinará si `j` es par o impar y usará positivo cuando `j` sea par y negativo cuando `j` sea impar.

```
if (ban){
                    fac1 = 1.0*fact(p+q-
j)*fact(q)/(1.0*fact(p+q)*fact(j)*fact(q-j));
                    ban=0;
           }
           else
           {
                   fac1 = -1.0*fact(p+q-
j)*fact(q)/(1.0*fact(p+q)*fact(j)*fact(q-j));
                   ban=1;
           }
```

Después, con la función `Mat_esc`, escala a `aux1` (que representa a Lm^j) por el coeficiente escalar (`fac1`) y guarda el resultado en `aux2`.

Suma el resultado de la iteración a `DP` con la función `summat2`, permitiendo acumular los términos del polinomio racional del denominador.

Con la función `Mulmat2,` multiplica la matriz actual en `aux1` (Lm^j) por Lm para obtener a Lm^{j+1} y lo guarda en `aux3`.

Y con la función asigna, guarda el resultado en `aux1` para que esté listo y se utilice en la siguiente iteración.

```
Mat_esc(aux1,fac1,aux2);
summat2(DP,aux2,DP);
mulmat2(aux1,Lm,aux3);
asigna(aux3,aux1);
```

3.2.1.8.3. Cálculo de la fracción racional de matrices

Para calcular (2.65), la función `inversa` calcula la inversa del denominador de Padé (`DP`) y el resultado lo guarda en `temp`.

```
inversa2(DP,temp);
```

La función `mulmat2` multiplica el resultado anterior de `temp` (DP^{-1})por `NP` y el resultado lo guarda en `aux3`, con la función `asigna`, copia nuevamente el resultado a `temp`, pues `temp` será el acumulador para la potencia posterior.

```
mulmat2(temp,NP,aux3);
asigna(aux3,temp);
```

La función MatrizPotencia eleva la matriz $R(L)$ a la potencia *m*, es decir: $(DP^{-1} \cdot NP)^m$, recordando que *m* es el *ceil(norma1/tolpade)* es decir, si la norma 1 de Matriz es mayor que el umbral, se escala la matriz y luego se deshace el escalamiento elevando la aproximación a la potencia *m*.

```
inversa2(DP,temp);
mulmat2(temp,NP,aux3);
asigna(aux3,temp);
MatrizPotencia(aux3,temp,m);
```

Y finalmente se copia el resultado en aux3 con la función `asigna` que ahora es el arreglo de salida `Res`.

```
asigna(aux3,Res)
```

3.2.2. Codigo_Principal.c

Este código, como se mencionó anteriormente, se desarrolló para calcular a la matriz de Lyapunov tipo retardada; para esto, manda a llamar a las funciones del código "funciones.h" conforme las necesite. De esta manera, el código se mantuvo ordenado y limpio, lo que también permitió disminuir los errores durante su desarrollo.

Después, también, se encarga de preparar y dar formato a los resultados obtenidos, para enviarlos vía UART a CoolTerm, para su graficación. En el **Apéndice C**, se puede consultar el código completo de "Codigo_Principal.c".

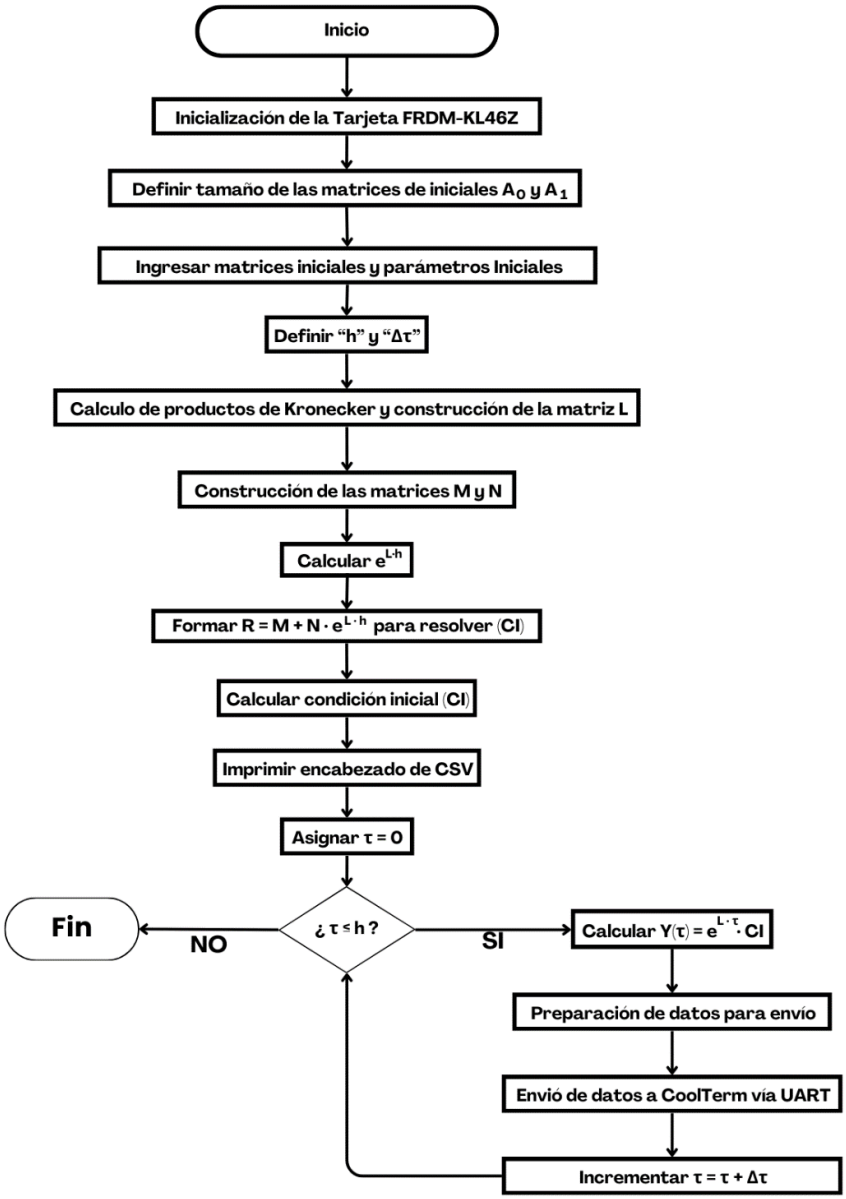


Figura 3.12: Diagrama de Flujo del Codigo_Principal.c.

Se define a $n = 2$, pues esta constante se usa para determinar las dimensiones de las matrices y vectores.

```
#define n 2
```

Se declara una variable local persistente, que se utiliza como contador de prueba en el bucle infinito del programa; no afecta el desarrollo de los cálculos.

```
volatile static int s = 0 ;
```

Después se inicia la declaración de matrices iniciales del sistema, donde:

- $A_0,\ A_1$: son las matrices del sistema dinámico (de entrada).
- h: corresponde al valor máximo en el intervalo $[0, h]$.
- *Wvec*: es el vector de condiciones iniciales extendidas.

```
float A0[n][n] = {{0, 1}, {-1, -2}};
float A1[n][n] = {{0, 0}, {-1, 1}};
float h=1;
float Wvec[2*n*n] = {0, 0, 0, 0, -3, 0, 0, -3};
```

Y se inicia una matriz identidad de tamaño $n \times n$:

```
Identidad(Iden);
```

Se comienza a organizar el cálculo los cálculos de cada parte de la ecuación (2.19).

3.2.2.1. Armado de *L*

Recordando que *L* es una matriz de tamaño $2n^2 \times 2n^2$ (2.18), se tratará como una matriz a bloques, como se muestra en la ecuación (3.13), donde $(l_{00}, l_{01}, l_{10}, l_{11})$ serán submatrices de tamaño $n^2 \times n^2$.

$$L = \begin{bmatrix} l_{00} & l_{01} \\ l_{10} & l_{11} \end{bmatrix}. \tag{3.13}$$

Primero se debe obtener a A_0^T y A_1^T a partir de A_0 y A_1, para ello, se usa la función `transpuesta` y posteriormente almacenar a A_0^T y A_1^T en `result0` y `result1` respectivamente.

```
transpuesta(A0, resul0);
transpuesta(A1, resul1);
```

Y recordando que ya se tiene inicializada la variable `iden` como una matriz identidad de $n \times n$ se comienza a realizar los cálculos de los cuatro productos de Kronecker para construir cada submatriz de la matriz expandida L, cada uno se almacenará en una matriz auxiliar de $n^2 \times n^2$ para después ensamblar la matriz L de tamaño $2n^2 \times 2n^2$.

Para la submatriz l_{00}:

```
productoKronecker(resul0, Iden, aux11);
for (int i = 0; i < n*n; i++){
       for (int j = 0; j < n*n; j++){
          L[i][j] = aux11[i][j];
       }
   }
```

Para la submatriz l_{01}:

```
productoKronecker(resul1, Iden, aux12);
   for (int i = 0; i < n*n; i++){
          for (int j = 0; j < n*n; j++){
             L[i][j+n*n] = aux12[i][j];
          }
      }
```

Para la submatriz l_{10}:

```
productoKronecker(Iden, resul1, aux21);
         for (int i = 0; i < n*n; i++){
                for (int j = 0; j < n*n; j++){
                   L[i+n*n][j] = -aux21[i][j];
                }
            }
```

Para la submatriz l_{11}:

```
productoKronecker(Iden, resul0, aux22);
          for (int i = 0; i < n*n; i++){
                 for (int j = 0; j < n*n; j++){
                   L[i+n*n][j+n*n] = -aux22[i][j];
                 }
            }
```

Donde, para las 4 operaciones, la línea `productoKronecer();` calcula el producto de Kronecker y lo almacena en su cuadrante correspondiente, en su respectiva matriz `aux` y con los ciclos anidados `for i` y `for j` se va determinando el lugar en el que se asignarán las matrices que contienen los productos de Kronecker (aux) de tal manera que al final quede la matriz L ensamblada dentro de `L[2*n*n][2*n*n]`.

3.2.2.2. Armado de M

Recordando que M es una matriz de tamaño $2n^2 \times 2n^2$ (2.20), se tratará como una matriz a bloques como se muestra en la ecuación (3.14), donde $(m_{00}, m_{01}, m_{10}, m_{11})$ serán submatrices de tamaño $n^2 \times n^2$.

$$M = \begin{bmatrix} m_{00} & m_{01} \\ m_{10} & m_{11} \end{bmatrix}. \tag{3.14}$$

La submatriz m_{00}, se trata de una matriz identidad que se genera usando la función `Identidad1` y se almacena en la matriz auxiliar `aux1` y a través de los ciclos `for i` y `for j` anidados, se asigna el cuadrante superior izquierdo dentro de la matriz M.

```
Identidad1(aux1);
                  for (int i = 0; i < n*n; i++){
                         for (int j = 0; j < n*n; j++){
                            M[i][j] = aux1[i][j];
                         }
                     }
```

La submatriz m_{01}, se trata de una matriz de tamaño $n^2 \times n^2$llena de ceros, que se genera con la función `ceros1` y se almacena en la matriz auxiliar `aux1` y de la misma manera, con los ciclos `for i` y `for j` anidados, se asigna el cuadrante superior derecho dentro de la matriz M.

```
                ceros1(aux1);
                       for (int i = 0; i < n*n; i++){
                              for (int j = 0; j < n*n; j++){
                                 M[i][j+n*n] = aux1[i][j];
                              }
                       }
```

Para la submatriz m_{10}, recordando que la matriz auxiliar `aux11`, ya contiene a $A_0^T \otimes I$, simplemente se hace la asignación en el cuadrante inferior izquierdo con los ciclos `for i` y `for j` anidados y acomodándolo en la matriz M.

```
                  for (int i = 0; i < n*n; i++){
                         for (int j = 0; j < n*n; j++){
```

```
            M[i+n*n][j] = aux11[i][j];
          }
      }
```

Y para la submatriz m_{11}, recordando que la matriz auxiliar `aux12`, ya contiene a $A_1^T \otimes I$, al igual que con el cuadrante inferior izquierdo, se hace la asignación en el cuadrante inferior derecho con los ciclos `for i` y `for j` anidados, acomodándolo en la matriz M.

```
for (int i = 0; i < n*n; i++){
       for (int j = 0; j < n*n; j++){
         M[i+n*n][j+n*n] = aux12[i][j];
        }
 }
```

3.2.2.3. Armado de *N*

Recordando que N es una matriz de tamaño $2n^2 \times 2n^2$ (2.21), se tratará como una matriz a bloques como se muestra en la ecuación (3.15), donde $(n_{00}, n_{01}, n_{10}, n_{11})$ son submatrices de tamaño $n^2 \times n^2$.

$$N = \begin{bmatrix} n_{00} & n_{01} \\ n_{10} & n_{11} \end{bmatrix}. \tag{3.15}$$

La submatriz n_{00}, es una matriz de $n^2 \times n^2$ llena de ceros, se genera con la función `ceros1` en la variable `aux1` y posterior a eso, a través de los ciclos anidados `for i` y `for j` se acomodan dentro de la matriz N en el cuadrante superior izquierdo.

```
ceros1(aux1);
    for (int i = 0; i < n*n; i++){
         for (int j = 0; j < n*n; j++){
             N[i][j] = aux1[i][j];
         }
    }
```

La submatriz n_{01}, al tratarse de una matriz identidad de $n^2 \times n^2$, se genera con la función `Identidad1` y se almacena en `aux1`, que, aunque se usó para generar la submatriz n_{00} llena de 0, al utilizar la función `Identidad1` sobre `aux1`, la regenera como una matriz identidad de $n^2 \times n^2$. Posteriormente, con los ciclos anidados `for i` y `for j` se acomodan dentro de la matriz N en el cuadrante superior derecho.

```
Identidad1(aux1);
    for (int i = 0; i < n*n; i++){
         for (int j = 0; j < n*n; j++){
```

```
                N[i][j+n*n] = -aux1[i][j];
            }
        }
```

Para la submatriz n_{10}, recordando que la variable `aux21` ya contiene a: $I \otimes A_1^T$ de cuando se calculó l_{10} de la matriz L, solo restaría acomodar a través de los ciclos anidados `for i` y `for j` dentro de la matriz expandida N en el cuadrante inferior izquierdo.

```
    for (int i = 0; i < n*n; i++){
          for (int j = 0; j < n*n; j++){
             N[i+n*n][j] = aux21[i][j];
          }
    }
```

Y para la submatriz n_{11}, recordando que la variable `aux22` ya contiene a $I \otimes A_0^T$ de cuando se calculó a l_{11} de la matriz L, solo restaría acomodar a través de los ciclos anidados `for i` y `for j` dentro de la matriz expandida N en el cuadrante inferior derecho.

```
    for (int i = 0; i < n*n; i++){
         for (int j = 0; j < n*n; j++){
             N[i+n*n][j+n*n] = aux22[i][j];
         }
    }
```

3.2.2.4. Armado de e^{Lh}

Con la matriz expandida L construida, se puede empezar a construir a e^{Lh} de la ecuación (2.19), donde se multiplica a L por el escalar h, para ello se utiliza la función `Mat_esc`, para obtener a Lh y almacenar el resultado en la matriz auxiliar `aux2`.

```
Mat_esc(L,h,aux2);
```

Y con Lh construido y almacenado en `aux2`, se calcula la exponencial de la matriz Lh usando Padé mediante la función `MatExpPad`, obteniendo a e^{Lh} y almacenándolo en la variable `R`.

```
MatExpPad(aux2,R);
```

3.2.2.5. Cálculo de CI

Con e^{Lh} y N ya construidas y almacenadas en `R` y `N` respectivamente, se calcula a $N \cdot e^{Lh}$ que, recordando que ambas son matrices de dimensiones $2n^2 \times 2n^2$, por lo que, para hacer la

multiplicación de dichas matrices, se utiliza la función `mulmat2` y el resultado se almacena en la matriz auxiliar `aux2`.

```
mulmat2(N,R,aux2);
```

Y con $N \cdot e^{Lh}$ calculada y guardada en `aux2`, recordando que la matriz M está calculada y almacenada en `M`, se suman las matrices, recordando que las matrices son de $2n^2 \times 2n^2$, se utiliza a la función `summat2` y el resultado se almacena en la matriz auxiliar R.

```
summat2(M,aux2,R);
```

Con toda la expresión $M + N \cdot e^{Lh}$ calculada y almacenada en la variable `R`, el siguiente paso es calcular su inversa, que se calcula con la función `inversa2` que de igual manera está desarrollada para trabajar con matrices de $2n^2 \times 2n^2$ y el resultado se almacena en la matriz auxiliar `aux2`.

```
inversa2(R,aux2);
```

Para este punto, la matriz auxiliar `aux2` ya contiene a $(M + Ne^{Lh})^{-1}$ y lo último es multiplicarlo por el vector W almacenado en la variable `Wvec`, para esto se desarrolló la matriz identidad mulmatvec2 que multiplica matrices de $2n^2 \times 2n^2$ y vectores de $2n^2$, y guarda el resultado en `CI`.

```
mulmatvec2(aux2,Wvec,CI);
```

De esta manera, la operación $(M + Ne^{Lh})^{-1}W_1$ queda almacenada en la variable `CI`.

3.2.2.6. Solución vectorizada

Teniendo a la condición inicial vectorizada CI, en la variable `CI` se evalúa la solución mediante la expresión (2.17).

Para evaluar la expresión $Y(\tau)$ se define un incremento temporal, que para el código se asignó en la variable `float incretau = 0.1`, con el cual se recorre el intervalo $\tau \in [0, h]$ mediante el ciclo for:

```
for (float tau = 0; tau <= h; tau=tau+incretau){
```

Para la construcción de L_τ, se multiplica a la matriz L almacenada en `L` por el valor actual de τ, para esto se utiliza la función `Mat_esc` y se almacena el resultado en la matriz auxiliar `aux2`.

```
Mat_esc(L,tau,aux2);
```

Para el cálculo de $e^{L\tau}$, se evalúa la exponencial de la matriz L usando la función `MatExpPad`, que implementa aproximaciones de Padé y el resultado lo almacena en la variable `R`.

```
MatExpPad(aux2,R);
```

Con $e^{L\tau}$ almacenada en `R` y la condición inicial vectorizada en `CI`, se multiplica a $e^{L\tau} \cdot CI$, utilizando la función `mulmatvec2` y almacena el resultado en `Y`.

```
    mulmatvec2(R,CI,Y);
}
```

Este procedimiento se repite para cada valor de τ en el intervalo *[0, h]*, el bloque completo para $T(\tau) = e^{L\tau}CI$:

```
for (float tau = 0; tau <= h; tau=tau+incretau){
    Mat_esc(L,tau,aux2);
    MatExpPad(aux2,R);
    mulmatvec2(R,CI,Y);
}
```

Entonces `Y` contiene la solución vectorizada del sistema en el instante de tiempo actual τ.

3.2.2.7. Preparación y tratamiento de los resultados

Recordando que el proceso de graficación en Matlab se dividió en 3 fases, siendo la primera la que se lleva a cabo dentro de "Codigo_Principal.c", que consiste en tratar los datos para enviarlos vía UART a CoolTerm.

Entonces, primero para formatear y enviar los resultados como un archivo (.csv), a través de la interfaz UART se declaró un buffer de cadena.

```
char linea[256];  // Almacena la línea completa
char temp[32];    // Para agregar cada campo numérico
```

Después se generó un encabezado (csv) dinámico, con nombres de columna Y0, Y1, Y2, Y3… hasta $Yn < 2n^2$, esto para facilitar la importación del archivo a MATLAB, pues los datos ya tendrán etiquetas de columnas.

```
sprintf(linea, "tau");
```

```
        for (int i = 0; i < 2*n*n; i++) {
            sprintf(temp, ",Y%d", i);
            strcat(linea, temp);
        }
        strcat(linea, "\r\n");
        PRINTF("%s", linea);
```

Con los encabezados listos, se generan los pasos de tiempo, con `int pasos`, calcula cuántas iteraciones del bucle `for` son necesarias para cubrir el intervalo *[0, h]*, y con un ciclo `for` se ejecuta el ciclo tantas veces como pasos tenga el intervalo, y el índice del paso lo almacena en `k`, y calcula el valor real de τ, y lo convierte a milésimas de segundo, con `sprintf`, escribe el valor convertido en la cadena `línea` como primer elemento de la línea.

```
        int pasos = (int)(h / incretau);
        for (int k = 0; k < pasos; k++) {
            float tau = k * incretau;
            int tau_int = k * (int)(incretau * 1000);  // Escalar a
milésimas
            sprintf(linea, "%d", tau_int);
```

Posteriormente, para la transmisión de datos vía UART, se utilizó la función sprintf, sin habilitar el soporte para variables en punto flotante para mantener el uso de memoria de la FRDM-KL46Z lo más libre posible, por lo que fue necesario convertir los datos a enteros multiplicándolos por (100,000,000) para mantener la precisión.

```
            Mat_esc(L, tau, aux2);
            MatExpPad(aux2, R);
            mulmatvec2(R, CI, Y);

            for (int i = 0; i < 2*n*n; i++) {
                int yi_int = (int)(Y[i] * 100000000);
                sprintf(temp, ",%d", yi_int);
                strcat(linea, temp);
            }
```

Y, por último, se imprime la línea para enviarla.

```
            strcat(linea, "\r\n");
            PRINTF("%s", linea);
        }
```

3.3. Recepción y guardado de datos

La segunda fase del proceso de graficación consiste en, con ayuda de CoolTerm, recibir los datos que envía la tarjeta FRDM-KL46Z y guardarlos en un archivo (.csv).

Para configurar a CoolTerm, consulte el **Apéndice E**. Una vez que podemos visualizar los datos en la interfaz de CoolTerm, como se observa en la Figura 3.13, se debe esperar a que la tarjeta termine de ejecutar las iteraciones.

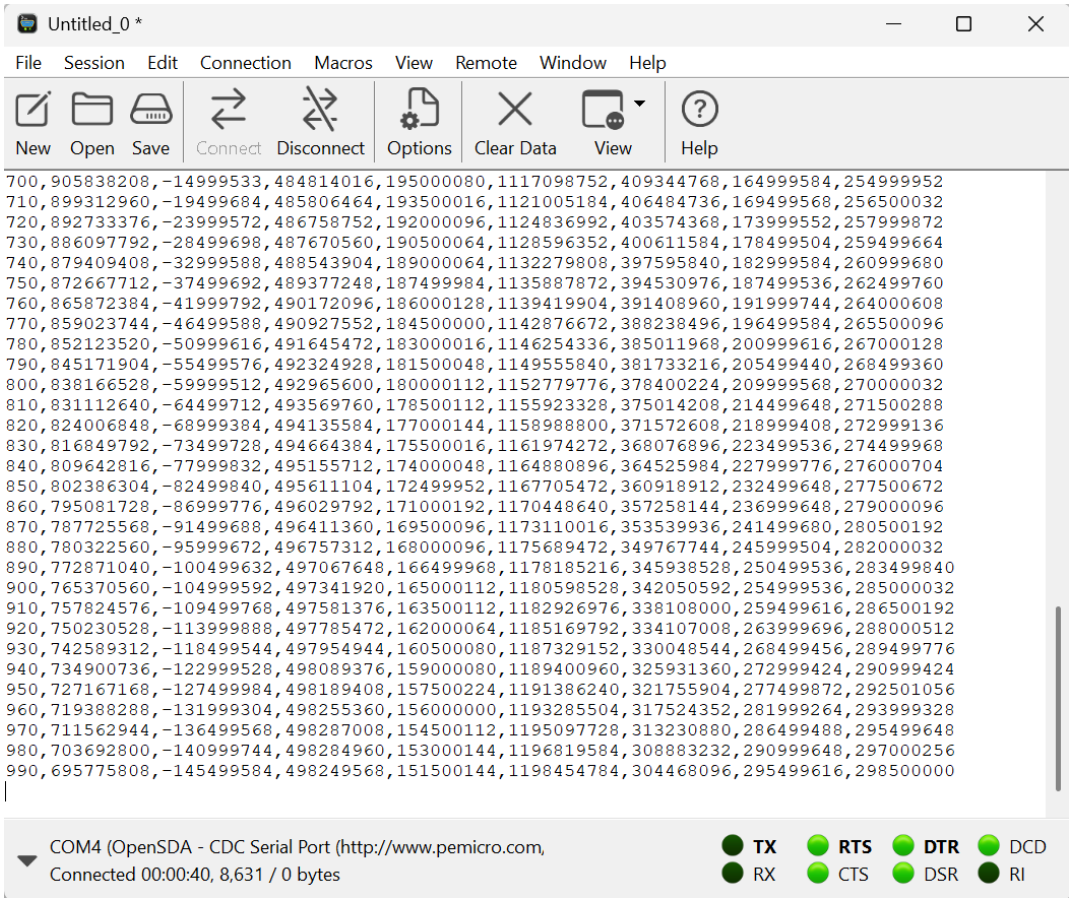
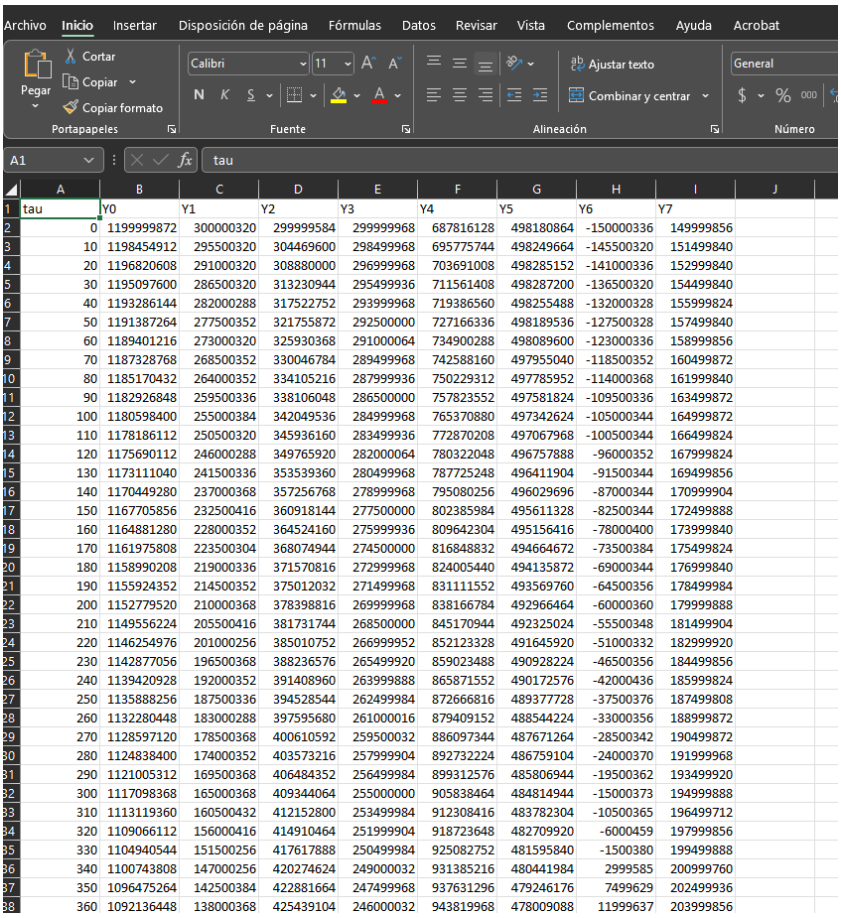


Figura 3.13: Resultados en la interfaz de CoolTerm.

Cuando se hayan terminado las iteraciones, se guardan los resultados en un archivo (.csv), como se observa en la Figura 3.14, siguiendo las instrucciones del **Apéndice E**. El archivo (.csv) se puede nombrar de cualquier manera, solo es importante que el nombre sea exactamente el mismo que se importe en el código "Graficador.m" en MATLAB, para el ejemplo se eligió el nombre "datos_prueba_1.csv".



	A	B	C	D	E	F	G	H	I	J
1	tau	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	
2	0	1199999872	300000320	299999584	299999968	687816128	498180864	-150000336	149999856	
3	10	1198454912	295500320	304469600	298499968	695775744	498249664	-145500320	151499840	
4	20	1196820608	291000320	308880000	296999968	703691008	498285152	-141000336	152999840	
5	30	1195097600	286500320	313230944	295499936	711561408	498287200	-136500320	154499840	
6	40	1193286144	282000288	317522752	293999968	719386560	498255488	-132000328	155999824	
7	50	1191387264	277500352	321755872	292500000	727166336	498189536	-127500328	157499840	
8	60	1189401216	273000320	325930368	291000064	734900288	498089600	-123000336	158999856	
9	70	1187328768	268500352	330046784	289499968	742588160	497955040	-118500352	160499872	
10	80	1185170432	264000352	334105216	287999936	750229312	497785952	-114000368	161999840	
11	90	1182926848	259500336	338106048	286500000	757823552	497581824	-109500336	163499872	
12	100	1180598400	255000384	342049536	284999968	765370880	497342624	-105000344	164999872	
13	110	1178186112	250500320	345936160	283499936	772870208	497067968	-100500344	166499824	
14	120	1175690112	246000288	349765920	282000064	780322048	496757888	-96000352	167999824	
15	130	1173111040	241500336	353539360	280499968	787725248	496411904	-91500344	169499856	
16	140	1170449280	237000368	357256768	278999968	795080256	496029696	-87000344	170999904	
17	150	1167705856	232500416	360918144	277500000	802385984	495611328	-82500344	172499888	
18	160	1164881280	228000352	364524160	275999936	809642304	495156416	-78000400	173999840	
19	170	1161975808	223500304	368074944	274500000	816848832	494664672	-73500384	175499824	
20	180	1158990208	219000336	371570816	272999968	824005440	494135872	-69000344	176999840	
21	190	1155924352	214500352	375012032	271499968	831111552	493569760	-64500356	178499984	
22	200	1152779520	210000368	378398816	269999968	838166784	492966464	-60000360	179999888	
23	210	1149556224	205500416	381731744	268500000	845170944	492325024	-55500348	181499904	
24	220	1146254976	201000256	385010752	266999952	852123328	491645920	-51000332	182999920	
25	230	1142877056	196500368	388236576	265499920	859023488	490928224	-46500356	184499856	
26	240	1139420928	192000352	391408960	263999888	865871552	490172576	-42000436	185999824	
27	250	1135888256	187500336	394528544	262499984	872666816	489377728	-37500376	187499808	
28	260	1132280448	183000288	397595680	261000016	879409152	488544224	-33000356	188999872	
29	270	1128597120	178500368	400610592	259500032	886097344	487671264	-28500342	190499872	
30	280	1124838400	174000352	403573216	257999904	892732224	486759104	-24000370	191999968	
31	290	1121005312	169500368	406484352	256499984	899312576	485806944	-19500362	193499920	
32	300	1117098368	165000368	409344064	255000000	905838464	484814944	-15000373	194999888	
33	310	1113119360	160500432	412152800	253499984	912308416	483782304	-10500365	196499712	
34	320	1109066112	156000416	414910464	251999904	918723648	482709920	-6000459	197999856	
35	330	1104940544	151500256	417617888	250499984	925082752	481595840	-1500380	199499888	
36	340	1100743808	147000256	420274624	249000032	931385216	480441984	2999585	200999760	
37	350	1096475264	142500384	422881664	247499968	937631296	479246176	7499629	202499936	
38	360	1092136448	138000368	425439104	246000032	943819968	478009088	11999637	203999856	

Figura 3.14: Resultados guardados en "datos_prueba_1.csv".

A continuación, se describe la tercera fase del proceso de graficación.

3.4. Graficación

La última fase del proceso de graficación consiste en importar los resultados almacenados “datos_prueba_1.csv”, tratar a los datos recibidos y graficarlos. En la Figura 3.15 se muestra la lógica principal utilizada para el desarrollo del código “Graficador.m” en MATLAB. El código completo se puede consultar en el **Apéndice D**.

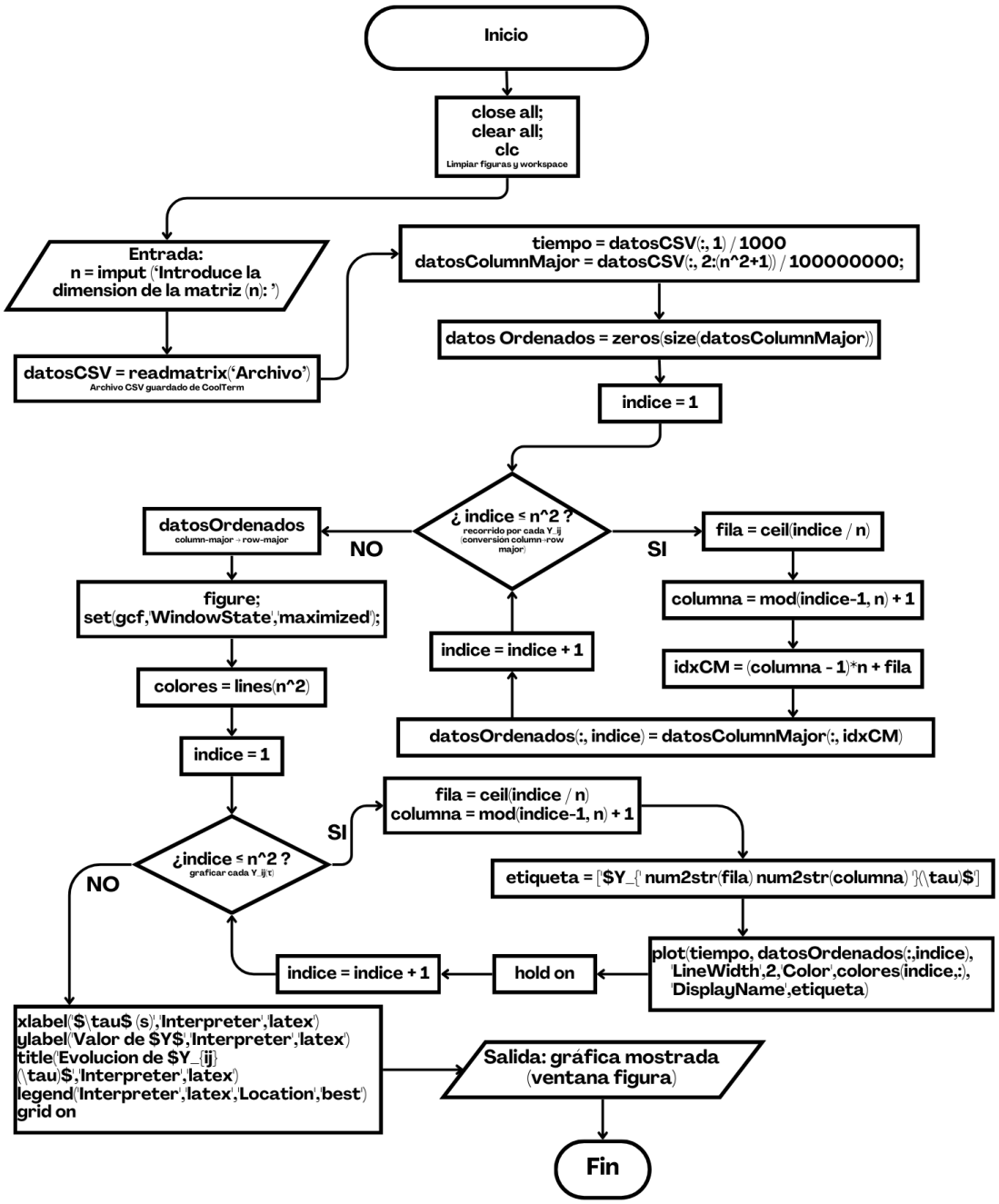


Figura 3.15: Diagrama de flujo del código Graficador.m en MATLAB.

Primero, se debe definir el tamaño de las matrices de entrada originales, esto se almacena en la variable `n`.

```
n = input('Introduce la dimensión de la matriz (n): ');
```

Luego se lee el archivo que guardamos (*datos_prueba_1.csv*) y se guarda en la variable `datosCSV`:

```
datosCSV = readmatrix('datos_prueba_1.csv');
```

Después de leer el archivo, recuperamos los valores de la primera columna llamada tau, ahí se almacenan los datos de tiempo en términos de (τ) que están en milisegundos, por lo que se deben convertir a segundos, esto se guarda en la variable `tiempo`.

```
tiempo = datosCSV(:,1) / 1000;
```

Y después solo recuperamos los valores de Y0 a Y3, correspondientes a las coordenadas de (Y_0, Y_1, Y_2, Y_3), puesto que estos son los valores que corresponden directamente a la matriz de Lyapunov tipo retardada, para matrices de entrada de dimensiones 2×2 (coordenadas correspondientes a las matrices de entrada), por ejemplo, para el caso de unas matrices de entrada de dimensión 3×3, la matriz resultado sería una de $2n^2 \times 2n^2$, donde $n = 3$, es decir, una matriz de 18×18. Sin embargo, los valores correspondientes a la matriz de Lyapunov tipo retardada son solo $(Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, Y_7, Y_8)$.

Para esto, primero, se leen todas las columnas, desde la columna 2 hasta $2n^2$, y recordando que al enviar los datos desde MCUXpresso IDE, se multiplicó por 100000000, por lo que se deben dividir entre el mismo número para no alterar los resultados y posteriormente guardarlos en la variable `datosColumnMajor`.

```
datosColumnMajor = datosCSV(:, 2:(n^2 + 1)) / 100000000;
```

Los datos enviados por la tarjeta FRDM-KL46Z se transmitieron en un vector unidimensional utilizando la convención column-major, que apila los elementos columna por columna, pero MATLAB interpreta en base a row-major, es decir, que apila fila por fila, por lo que es necesario un reordenamiento de la matriz, pues de dejarlo así, el etiquetado no coincidiría con los resultados de cada valor de *Y*, creando inconsistencias visuales.

Para esto, primero se crea una matriz vacía llamada `datosOrdenados` y la definimos del mismo tamaño que `datosColumnMajor`.

```
datosOrdenados = zeros(size(datosColumnMajor));
```

Y con un ciclo `for` se recorre cada posición de la matriz vectorizada; después calcula la fila a la que pertenece cada elemento y lo mismo con las columnas. De esta manera, los datos ordenados llenan la matriz `datosOrdenados`.

```
for indice = 1:n^2
    fila = ceil(indice / n);
    columna = mod(indice-1, n) + 1;
    idxCM = (columna-1)*n + fila;
    datosOrdenados(:, indice) = datosColumnMajor(:, idxCM);
end
```

Después de procesar los datos, se inicia la graficación, para esto, primero crea la figura y definen las dimensiones que, para una mejor visualización, se definió a la figura de tamaño de pantalla completa.

```
figure;
set(gcf,'WindowState','maximized');
```

Iniciando con la graficación, se crea una paleta de colores automática usando la función `lines()`, y se define que genere n^2 colores distintos para que no se repitan y se almacena en la variable `colores`.

```
colores = lines(n^2);
```

Igual que antes, se crea un ciclo `for` para recorrer cada elemento de la matriz, con `fila` y `columna` se descubre la posición de cada valor dentro de la matriz, para después asignarle su etiquetado correspondiente con `etiqueta`.

```
for indice = 1:n^2
    fila = ceil(indice / n);
    columna = mod(indice-1, n) + 1;
    etiqueta = ['$Y_{' num2str(fila) num2str(columna)
'}(\tau)$'];
```

Y con `plot` se generan las gráficas de los valores almacenados en `tiempo` (vector con los valores de τ) y `datosOrdenados`, con `LineWidth`, asignamos un valor 2 para hacer las líneas de las gráficas más gruesas, con `Color` asignamos un color de la paleta de colores llamada `colores` y con `DisplayName` asignamos el nombre que aparecerá en la leyenda. Terminamos con un `hold on` para que MATLAB no borre las gráficas y almacene todas en la misma figura.

```
    plot(tiempo, datosOrdenados(:,indice), 'LineWidth', 2, ...
```

```
            'Color', colores(indice,:), ...
            'DisplayName', etiqueta);
        hold on;
    end
```

Por último, se generan las etiquetas de los nombres de los ejes X, así como el título de la gráfica.

```
xlabel('$\tau$ (s)','Interpreter','latex');
ylabel('Valor de $Y$','Interpreter','latex');
title('Evolucion de $Y_{ij}(\tau)$','Interpreter','latex');

legend('Interpreter','latex','Location','best');
grid on;
```

Así, cuando se ejecute el código, en la ventana de comandos, MATLAB preguntará al usuario las dimensiones de las matrices de entrada, para graficar únicamente los resultados que corresponden a la matriz de Lyapunov en una sola figura.

Capítulo 4
Resultados

En este capítulo se muestran los resultados de ejecutar los códigos y funciones desarrolladas en el Capítulo 3 "Desarrollo" y en el **Apéndice A**, para el cálculo de la matriz de Lyapunov tipo retardada, ejecutándolos en la tarjeta de desarrollo FRDM-KL46Z.

Para ayudar a validar su funcionamiento, se usaron matrices (4.1) y (4.2) obtenidas del trabajo de Kharitonov y Zhabko en "*Lyapunov–Krasovskii Approach to the robust stability analysis of time-delay systems"* [14], en el cual se analizaron sistemas con retardo en el marco de la teoría de estabilidad de Lyapunov–Krasovskii.

4.1. Matrices de entrada

Se utilizaron las siguientes matrices de entrada:

$$A_0 = \begin{bmatrix} 0 & 1 \\ -1 & -2 \end{bmatrix}, \tag{4.1}$$

$$A_1 = \begin{bmatrix} 0 & 0 \\ -1 & 1 \end{bmatrix}. \tag{4.2}$$

Dichas matrices corresponden al ejemplo 1, presentado en la página 19 del artículo Kharitonov y Zhabko [14], el cual también incluye una gráfica de los resultados que obtuvieron. Esto permite comparar dichos resultados con los obtenidos de la tarjeta FRDM-KL46Z.

4.2. Resultados en la tarjeta FRDM-KL46Z

Para visualizar los resultados obtenidos al ejecutar el algoritmo en la tarjeta FRDM-KL46Z en tiempo real, en MCUXpresso IDE era necesario colocar banderas al final del código (antes de pasar a la siguiente iteración), como se muestra en la Figura 4.1, con el fin de parar la ejecución del código para poder visualizar los resultados obtenidos en dicha iteración, como se muestra en la Figura 4.2, proceso que resulta ineficiente y repetitivo, ya que, para obtener gráficas más precisas, es recomendable ejecutar alrededor de 100 iteraciones.

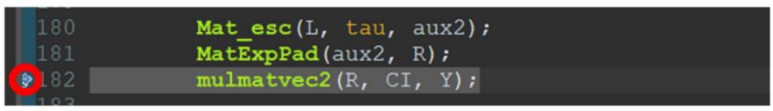


Figura 4.1: Bandera colocada al final de "Codigo_Principal.c" en MCUXpresso IDE.

Expression	Type	Value
Y	float [8]	0x1fffe668 <Y>
Y[0]	float	11.9682055
Y[1]	float	2.91000319
Y[2]	float	3.08879995
Y[3]	float	2.96999955
Y[4]	float	7.03691006
Y[5]	float	4.98285151
Y[6]	float	-1.4100033
Y[7]	float	1.52999842

Figura 4.2: Ventana "Expressions View" mostrando el contenido de Y después de una iteración.

En el siguiente punto se muestran los resultados en CoolTerm, antes de importarlos y graficarlos en MATLAB.

4.3. Resultados en CoolTerm

Gracias a CoolTerm, el proceso de recolección de resultados se optimizó, pues se eliminó la necesidad de colocar banderas para detener la ejecución y se obtienen los resultados de todas las iteraciones de manera continua, como se muestra en la Figura 4.3, y se guarda en "*datos_prueba_1.csv*", como se muestra en la Figura 4.4.

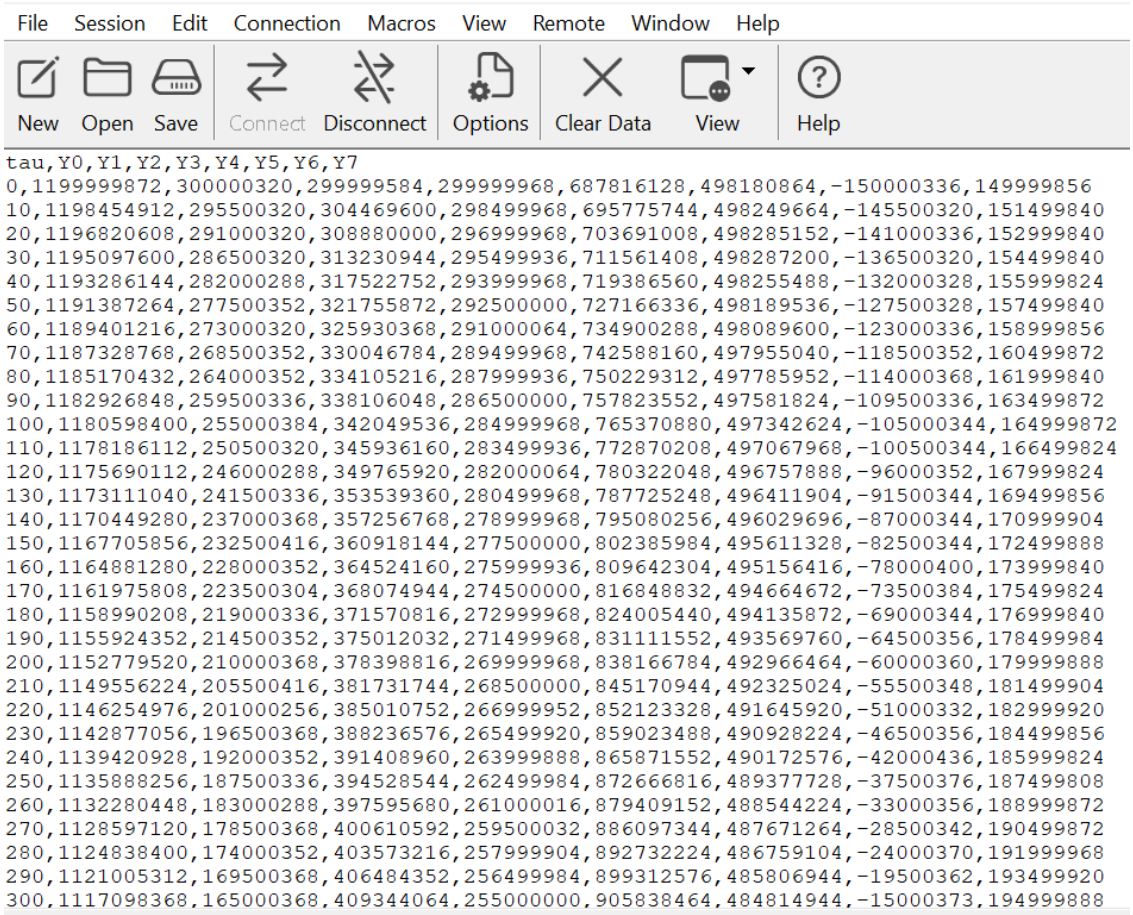


Figura 4.3: Resultados en la interfaz de CoolTerm.

	A	B	C	D	E	F	G	H	I
1		Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
2	0	1199999872	300000320	299999584	299999968	687816128	498180864	-150000336	149999856
3	10	1198454912	295500320	304469600	298499968	695775744	498249664	-145500320	151499840
4	20	1196820608	291000320	308880000	296999968	703691008	498285152	-141000336	152999840
5	30	1195097600	286500320	313230944	295499936	711561408	498287200	-136500320	154499840
6	40	1193286144	282000288	317522752	293999968	719386560	498255488	-132000328	155999824
7	50	1191387264	277500352	321755872	292500000	727166336	498189536	-127500328	157499840
8	60	1189401216	273000320	325930368	291000064	734900288	498089600	-123000336	158999856
9	70	1187328768	268500352	330046784	289499968	742588160	497955040	-118500352	160499872
10	80	1185170432	264000352	334105216	287999936	750229312	497785952	-114000368	161999840
11	90	1182926848	259500336	338106048	286500000	757823552	497581824	-109500336	163499872
12	100	1180598400	255000384	342049536	284999968	765370880	497342624	-105000344	164999872
13	110	1178186112	250500320	345936160	283499936	772870208	497067968	-100500344	166499824
14	120	1175690112	246000288	349765920	282000064	780322048	496757888	-96000352	167999824
15	130	1173111040	241500336	353539360	280499968	787725248	496411904	-91500344	169499856
16	140	1170449280	237000368	357256768	278999968	795080256	496029696	-87000344	170999904
17	150	1167705856	232500416	360918144	277500000	802385984	495611328	-82500344	172499888
18	160	1164881280	228000352	364524160	275999936	809642304	495156416	-78000400	173999840
19	170	1161975808	223500304	368074944	274500000	816848832	494664672	-73500384	175499824
20	180	1158990208	219000336	371570816	272999968	824005440	494135872	-69000344	176999840
21	190	1155924352	214500352	375012032	271499968	831111552	493569760	-64500356	178499984
22	200	1152779520	210000368	378398816	269999968	838166784	492966464	-60000360	179999888
23	210	1149556224	205500416	381731744	268500000	845170944	492325024	-55500348	181499904
24	220	1146254976	201000256	385010752	266999952	852123328	491645920	-51000332	182999920
25	230	1142877056	196500368	388236576	265499920	859023488	490928224	-46500356	184499856
26	240	1139420928	192000352	391408960	263999888	865871552	490172576	-42000436	185999824
27	250	1135888256	187500336	394528544	262499984	872666816	489377728	-37500376	187499808
28	260	1132280448	183000288	397595680	261000016	879409152	488544224	-33000356	188999872
29	270	1128597120	178500368	400610592	259500032	886097344	487671264	-28500342	190499872
30	280	1124838400	174000352	403573216	257999904	892732224	486759104	-24000370	191999968
31	290	1121005312	169500368	406484352	256499984	899312576	485806944	-19500362	193499920
32	300	1117098368	165000368	409344064	255000000	905838464	484814944	-15000373	194999888
33	310	1113119360	160500432	412152800	253499984	912308416	483782304	-10500365	196499712
34	320	1109066112	156000416	414910464	251999904	918723648	482709920	-6000459	197999856
35	330	1104940544	151500256	417617888	250499984	925082752	481595840	-1500380	199499888
36	340	1100743808	147000256	420274624	249000032	931385216	480441984	2999585	200999760
37	350	1096475264	142500384	422881664	247499968	937631296	479246176	7499629	202499936

Figura 4.4: Resultados guardados en "datos_prueba_1.csv".

A continuación, se muestra la graficación en MATLAB.

4.4. Graficación de resultados

Después de correr el código “Graficador.m” en MATLAB, asegurándonos de que el nombre “*datos_prueba_1.csv*” es correcto, primero nos preguntará el tamaño original de las matrices de entrada, como se muestra en la Figura 4.5, y después de ingresar el tamaño correspondiente, generará una gráfica como se muestra en la Figura 4.6.

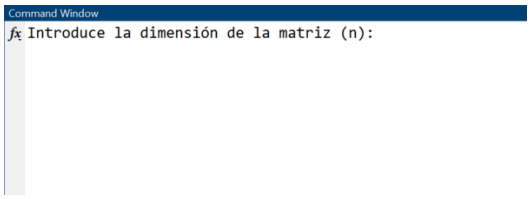


Figura 4.5: Ventana de comandos de MATLAB.

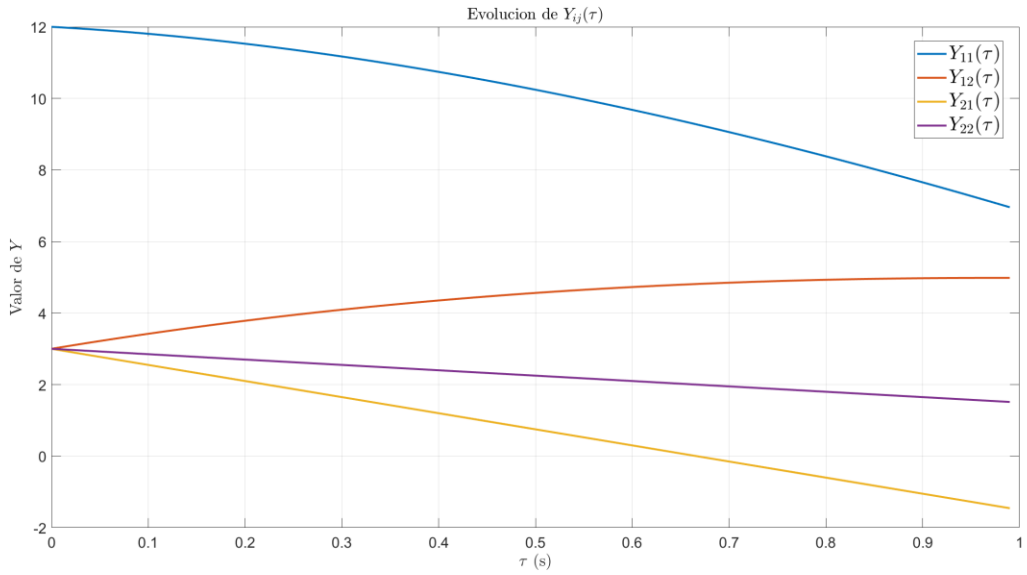


Figura 4.6: Gráfica en MATLAB de los resultados de la tarjeta FRDM-KL46Z.

4.5. Comparación de los resultados

Con la gráfica obtenida de MATLAB que se muestra en la Figura 4.6, se puede comparar de manera visual con la de Kharitonov y Zhabko [14], concluyendo que ambas son semejantes, por lo que los resultados son correctos, lo que valida el cálculo implementado en la FRDM-KL46Z. También es importante considerar que para esta prueba se consideró a un sistema lineal con un único retardo.

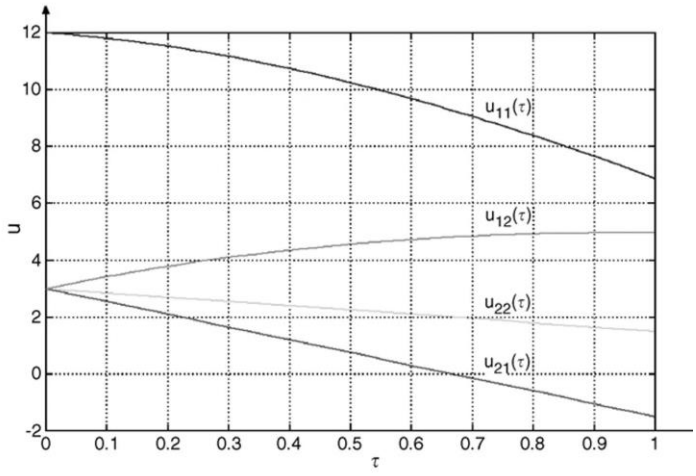


Figura 4.7: Gráfica de referencia. Imagen tomada de [14].

Se realizó el cálculo de la matriz de Lyapunov tipo retardada en MATLAB, tomando a las matrices de entrada (4.1) y (4.2), donde se obtuvo la siguiente gráfica:

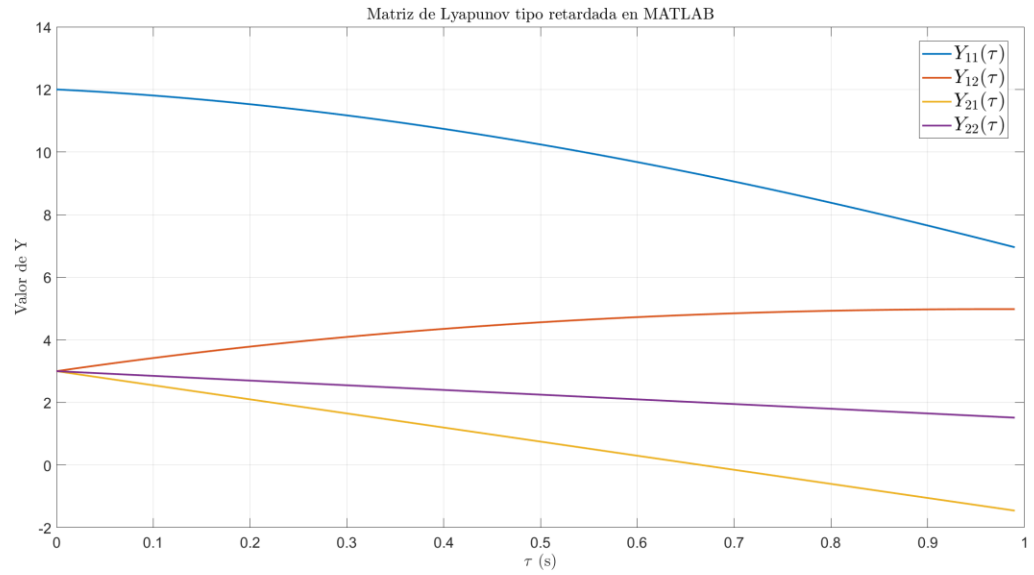


Figura 4.8: Gráfica de los resultados del cálculo de la matriz de Lyapunov tipo retardada en MATLAB.

4.6. Cálculo del error

Tomando como parámetro los resultados que se observan en la Figura 4.8, obtenidos de realizar el cálculo de la matriz de Lyapunov tipo retardada en MATLAB utilizando la función *expm*, se calculó el error de los resultados que se observan en la Figura 4.6, obtenidos del cálculo de la matriz de Lyapunov tipo retardada en la tarjeta FRDM-KL46Z, restando los resultados en cada iteración, donde al final se obtuvo la siguiente gráfica:

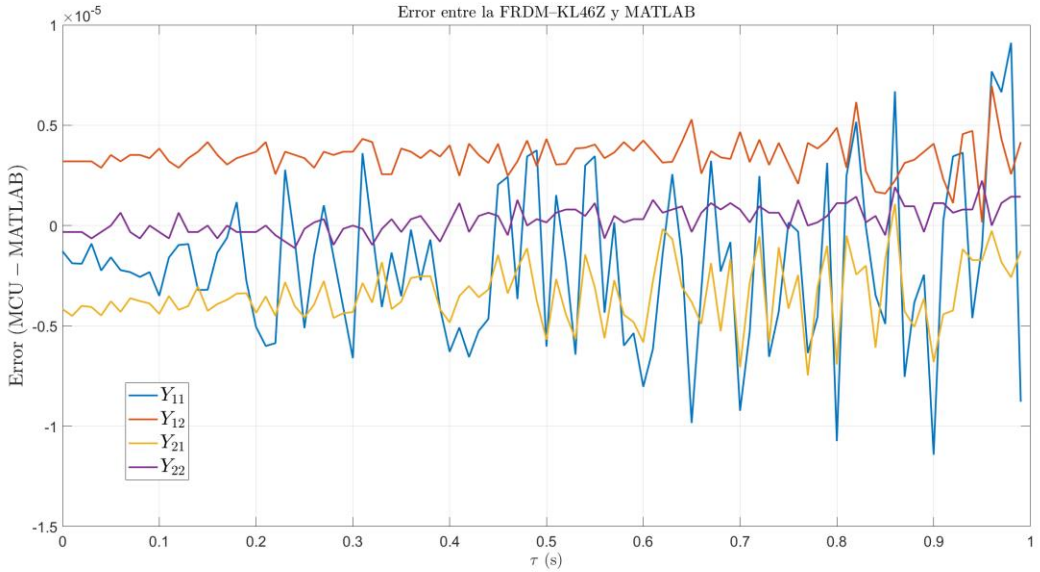


Figura 4.9: Error de los resultados.

Y tomando como índice la integral del error cuadrático (ISE), se obtuvo la siguiente gráfica:

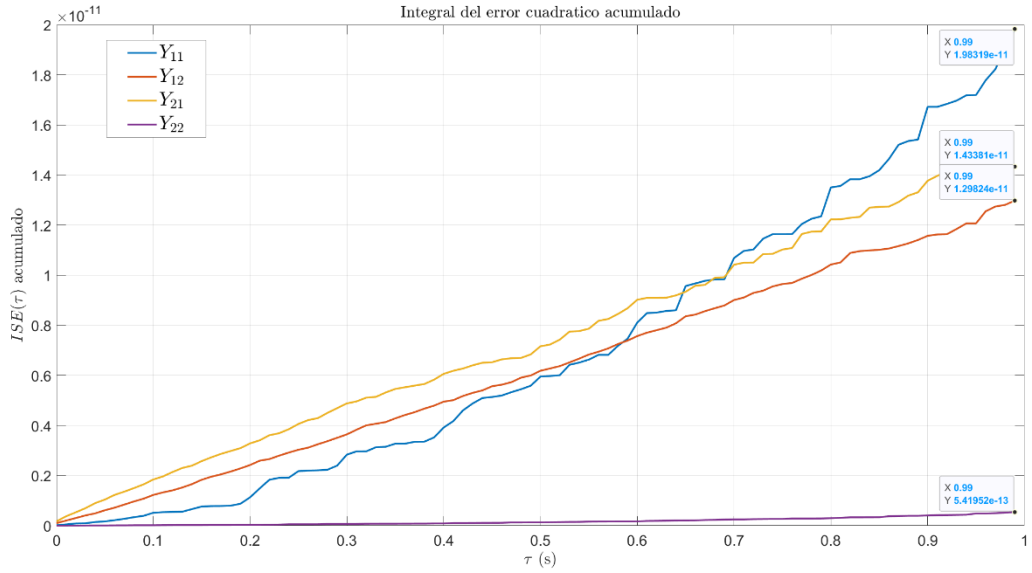


Figura 4.10: Gráfica de la integral del error cuadrático (ISE).

En donde se observa que las componentes: $Y_{11} = 1.98319 \times 10^{-11}$, $Y_{12} = 1.29824 \times 10^{-11}$, $Y_{21} = 1.43381 \times 10^{-11}$ y $Y_{22} = 5.41952 \times 10^{-13}$, valores que se consideran demasiado pequeños, demostrando que el cálculo realizado en la tarjeta FRDM-KL46Z es suficientemente exacto.

Conclusiones y trabajo futuro

Con el proyecto terminado, se concluye que:

- Es posible implementar el cálculo de la matriz de Lyapunov tipo retardada en la tarjeta de desarrollo considerada de bajo costo, FRDM-KL46Z, mediante la programación en lenguaje C y el uso de MCUXpresso IDE, cumpliendo el objetivo principal del proyecto.

- Analizando la gráfica obtenida en MATLAB y comparándola con la de Kharitonov y Zhabko (2003) [14], se verifica que los resultados coinciden, por lo que se confirma que el cálculo de la matriz de Lyapunov tipo retardada en la tarjeta FRDM-KL46Z funciona correctamente.

- La integral del error cuadrático (ISE) entre los resultados del cálculo en MATLAB y los cálculos ejecutados en la tarjeta FRDM-KL46Z es demasiado pequeño, por lo que se considera que los resultados obtenidos en la tarjeta son suficientemente exactos.

- La comunicación entre MCUXpresso IDE y MATLAB fue exitosa, pues en conjunto permitieron desde la ejecución de los códigos en la tarjeta FRDM-KL46Z, el envío de los resultados y la visualización gráfica de los mismos.

5.1. Trabajos futuros

Analizando las posibilidades de mejora, se analizan los siguientes puntos:

- Se determina que el cálculo funcionaría en tarjetas o microcontroladores con características superiores a la FRDM-KL46Z, siempre que sean compatibles con el lenguaje de programación C y cambiando el entorno de desarrollo (IDE), debido a que el procedimiento desarrolla cada operación que necesita, lo que permitiría analizar también matrices de tamaños superiores en caso de que las capacidades de la tarjeta FRDM-KL46Z se vean sobrepasadas.

- También se analiza la posibilidad de automatizar el proceso completo de trabajo, desde el cálculo de la matriz de Lyapunov tipo retardada hasta la graficación de los resultados, para reducir la intervención del usuario y mejorar el flujo del proceso.

- Con el cálculo de la matriz de Lyapunov realizado, se pueden implementar leyes de control que incluyen a dicha matriz a sistemas reales con retardo puntual, debido a que se hizo más accesible al ejecutarla en la tarjeta FRDM-KL46Z.

Referencias

[1] E. Fridman, “Tutorial on Lyapunov-based methods for time-delay systems”, *European Journal of Control*, vol. 20, no. 6, pp. 271–283, 2014.

[2] G. Batista and G. Takács, “QPMPC algorithm on an ARM Cortex M4 embedded microcontroller”, in 10th International Doctoral Seminar (IDS), Varaždin, Croatia, pp. 5–8, Sep. 2015.

[3] D. R. D. S. Medeiros and L. H. Fernandes, “Distributed genetic algorithms for Low-Power, Low-Cost and Small-Sized memory devices”, *Electronics*, vol. 9, no. 11, pp. 1–32, nov. 2020.

[4] S. Javed, K. Ishaque, J. Shek, and S. J. Rind, “A low-cost microcontroller implementation of fuzzy controller for renewable energy converters”, *International Transactions on Electrical Energy Systems*, vol. 2025, pp. 1–9, 2025.

[5] C. Cuvas, O. J. Santos–Sánchez, P. Ordaz and L. Rodriguez-Guerrero, “Suboptimal control for systems with commensurate and distributed delays of neutral type”, *Int. J. Robust Nonlinear Control*, vol. 32, no. 6, pp. 3190–3205, Apr. 2022.

[6] J. M. Ortega–Martínez, O. J. Santos–Sánchez, L. Rodríguez–Guerrero, H. Romero-Trejo, and J. P. Ordaz–Oliver, “Adaptative nonlinear optimal control for a banana dehydration process”, *Int. J. Innov. Comput. Inf. Control*, vol. 14, no. 6, pp. 2055–2070, Dec. 2018.

[7] L. Juárez, S. Mondié, and L. Vite, “Nested Stabilization for Connected Cruise Control via the Delay Lyapunov Matrix”, *Proc. 16th. Int. Conf. on Electrical Engineering, Computing Science and Automatic Control (CCE)*, Mexico, pp. 1–6, Sept. 2019.

[8] M. A. Gomez, W. Michels, and S, Mondié “Design of delay-based output-feedback controllers optimizing a quadratic cost function via the delay Lyapunov matrix” *Automatica*, vol. 107, pp. 146–153, Aug. 2019.

[9] The MathWorks, Inc., “lyap” *MATLAB Help Center*, 2025 [Online]. Available: https://www.mathworks.com/help/control/ref/lyap.html. [Accessed: Jun. 4, 2025].

[10] T. Penzl, “LYAPACK: A MATLAB Toolbox for Large Lyapunov and Riccati Equations, Model Reduction Problems, and Linear–Quadratic Optimal Control Problems – Users’ Guide, Version 1.0”. *Netlib*, Nov. 1999. [Online]. Available: https://www.netlib.org/lyapack/guide.pdf. [Accessed: Jun. 4, 2025].

[11] E. Fridman, Introduction to Time-Delay Systems: Analysis and Control. Cham, Switzerland: Birkhäuser, 2014.

[12] A. M. Lyapunov, The general problem of the stability of motion. A. T. Fuller, Trans. London, U.K., and Washington DC, U.S.A: Taylor & Francis, 1992.

[13] A. Castaño-Hernández, "Control lineal de sistemas de tráfico con retardos," M.S. thesis, Univ. Autónoma del Estado de Hidalgo, Pachuca, Hgo., México, 2020.

[14] V. L. Kharitonov and A. P. Zhabko, "Lyapunov–Krasovskii approach to the robust stability analysis of time-delay systems", *Automatica*, vol. 39, pp. 15-20, January 2003.

[15] NXP Semiconductors, "MCUXpresso Integrated Development Environment (IDE)", 2025. [Online]. Available: https://www.nxp.com/mcuxpresso/ide [Accessed: May 30, 2025].

[16] Embarcados, "Primerios pasos com MCUXpresso IDE", 2019 [Online]. Available: https://embarcados.com.br/primeiros-passo-com-mcuxpresso-ide/. [Accessed: May 30, 2025].

[17] R. Meier, "The Meier's Freeware Page", 2025 [Online]. Available: https://freeware.the-meiers.org. [Accessed: Jun 12, 2025].

[18] MacUpdate, "Download CoolTerm for Mac | MacUpdate", 2024 [Online]. Available: https://coolterm.macupdate.com/. [Accessed: Jun 12, 2025].

[19] The MathWorks, Inc., "What is MATLAB?". [Online]. Available: https://www.mathworks.com/discovery/what-is-matlab.html. [Accesed: Jun. 15, 2025].

[20] 1000marcas.net, "MATLAB logo", 2022. [Online]. Available: https://1000marcas.net/matlab-logo/. [Accessed: Jun. 15, 2025].

[21] The MathWorks, Inc., "MATLAB Product Description". [Online]. Available: https://www.mathworks.com/help/matlab/learn_matlab/product-description.html. [Accessed: Jun. 15, 2025].

[22] The MathWorks, Inc., "Products and Services". [Online]. Available: https://www.mathworks.com/products.html. [Accessed: Jun. 15, 2025].

[23] The MathWorks, Inc., "MATLAB". [Online]. Available https://www.mathworks.com/products/matlab.html. [Accesed: Jun. 15, 2025].

[24] H. Sayama, Introduction to the Modeling and Analysis of Complex Systems. Geneseo, NY: Open SUNY Textbooks, 2015.

[25] O. J. Santos Sánchez. (2009, Mar.). *Sistemas con retardos* [Online]. Available: https://www.uaeh.edu.mx/campus/icbi/investigacion/sistemas/seminarios/Fech_09_03_06b/Presentantacion06_03b.pdf [Accessed: Jul. 14, 2025].

[26] Zdynamics.org. (2023). *Estabilidad de sistemas dinámicos* [Online]. Available: https://www.zdynamics.org/docs/Estabilidad.pdf [Accessed: Jun. 23, 2025].

[27] J. P. Hespanha, *LINEAR SYSTEMS THEORY*, 2nd. ed. Princeton, NJ: Princeton University Press, 2018.

[28] V. Kharitonov, *Time-delay systems: Lyapunov functionals and matrices*. Springer Science & Business Media, 2012.

[29] S. S. Epp, *Matemáticas discretas con aplicaciones*, 4th. ed. México: Cengage Learning, 2012.

[30] D. C. Lay, *Álgebra Lineal y sus Aplicaciones*, 3rd. ed. México: Pearson Educación, 2007.

[31] H. Anton, *Introducción al álgebra lineal*, 3rd. ed. México: LIMUSA Wiley, 2003.

[32] M. G. Eberle, "Producto de Kronceker y sus aplicaciones", Ph.D. dissertation, Universidad Nacional del Sur, Bahía Blanca, Argentina, 2021.

[33] R. A. Horn and C.R. Johnson, *Matrix Analysis*, 2nd. ed. New York: Cambridge Univesity Press, 2013.

[34] S. I. Grossman and J. J. Flores Godoy, *Álgebra Lineal*, 7th. ed. Mexico: McGraw-Hill Interamericana, 2012.

[35] G. H. Goub and C. F. Van Loan, *Matrix Computations*. 4th. ed. Baltimore: Jhons Hopkins University Press, 2013.

[36] Freescale Semiconductor, Inc., *FRDM-KL46Z User's Manual, Rev. 1.0*. [Online]. Available: https://www.farnell.com/datasheets/1728679.pdf. [Accessed: May. 25, 2025].

[37] Arduino S.r.l., *Arduino Uno R3, User Manual* [Online]. Available: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf. [Accessed: May. 20, 2025].

[38] Microchip Technology Inc., *PIC16F882/883/884/885/886/887 Data Sheet*, DS40001291H. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/DataSheets/40001291H.pdf. [Accessed: May. 28, 2025].

[39] Microchip Technology Inc., "PIC16F887 Product Page", 2025. [Online]. Available: https://www.microchip.com/en-us/product/pic16f887. [Accessed: May. 28, 2025].

Apéndice A Funciones auxiliares para el cálculo de la matriz de Lyapunov tipo retardada

Aquí se presentan las funciones desarrolladas para el cálculo de la matriz de Lyapunov, las cuales se utilizaron de manera auxiliar en otras funciones más complejas y en "Codigo_Principal.c".

A.1. Función transpuesta

La transposición de matrices es básica en el álgebra matricial y una operación fundamental para el cálculo de la matriz de Lyapunov tipo retardada, ya que en ella se utilizan términos como A_1^T, recordando que las matrices pasan de $m \times n$ (2.26), a $n \times m$ (2.27). En la Figura A.1 se muestra la lógica principal utilizada para el desarrollo de la función transpuesta.

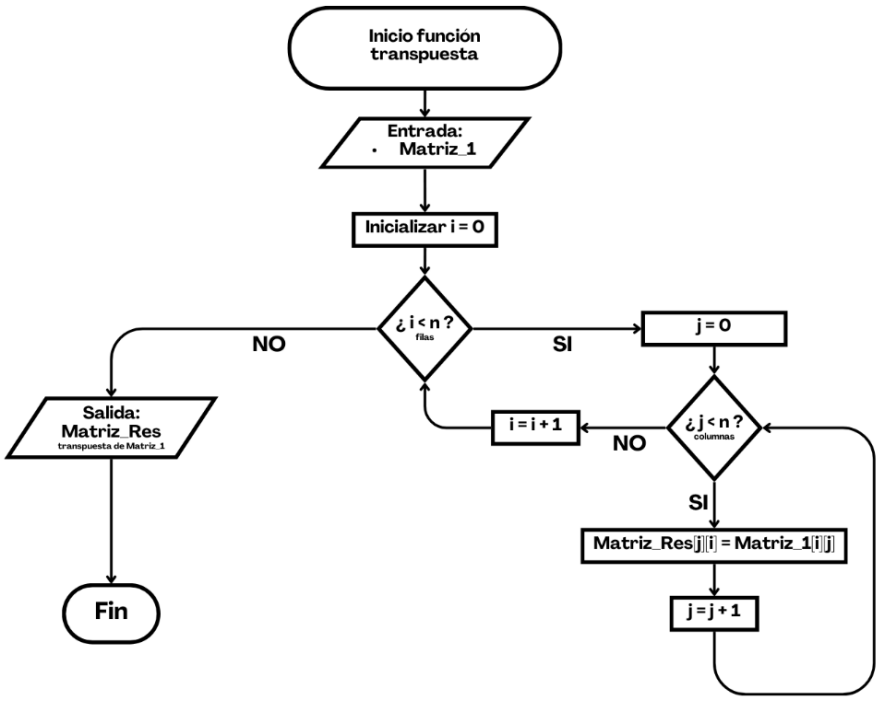


Figura A.1: Diagrama de flujo de la función transpuesta.

Código de la función transpuesta:

```
void transpuesta(float Matriz_1[n][n], float
     Matriz_Res[n][n]) {
```

```
        for (int i = 0; i < n; i++)
                        {
                            for (int j = 0; j < n; j++)
                            {
                              Matriz_Res[j][i] = Matriz_1[i][j];
                            }
                        }
    }
```

Donde `Matriz_1[n][n]`es la matriz de entrada de $n \times n$, `Matriz_Res[j][i]` y es la matriz transpuesta (de salida), primero se inicia un ciclo `for i`, que recorrerá las filas de la matriz de entrada, y después un ciclo `for j`, que recorrerá las columnas de cada fila en la matriz de entrada. En `Matriz_Res[j][i] = Matriz_1[i][j]`es donde ocurre la transposición, toma los valores de la fila i y la columna j de `Matriz_1[n][n]` y los coloca en la fila `j` y columna `i` de `Matriz_Res`.

A.2. Función ceros1

La inicialización de matrices con ceros suele utilizarse como bloques auxiliares en la construcción de matrices, la función ceros1 está programada para trabajar con matrices de dimensiones $n^2 \times n^2$. En la Figura A.2 se muestra la lógica principal utilizada para el desarrollo de la función ceros1.

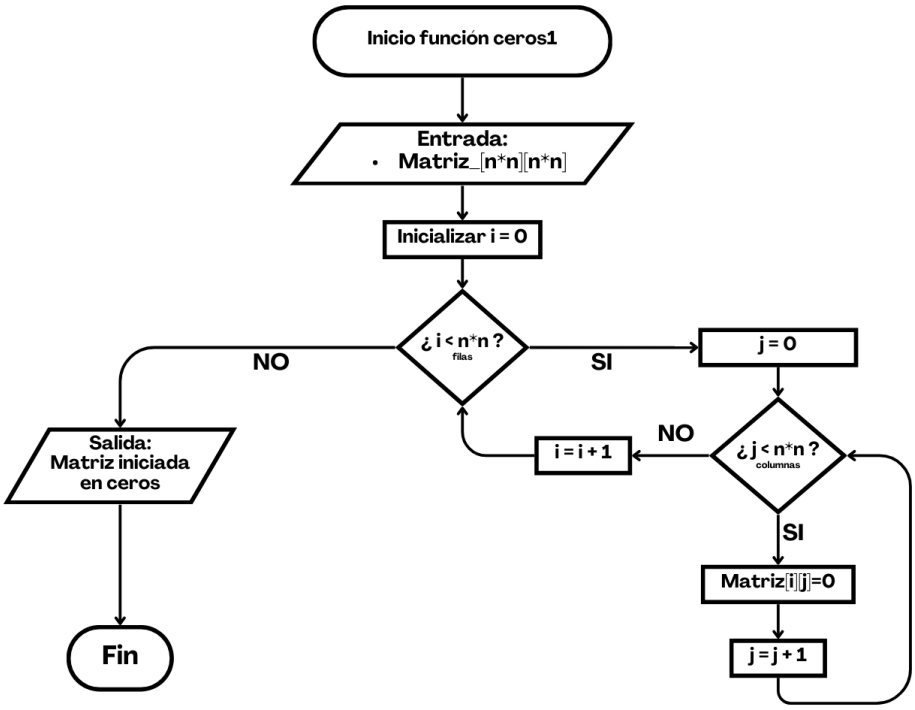


Figura A.2: Diagrama de flujo de la función ceros1.

La función ceros1, se programó manualmente, evitando utilizar bibliotecas como <String> que consumen memoria de manera innecesaria.

Código de la función ceros1.

```
void ceros1(float Matriz[n*n][n*n]){
    for (int i = 0; i < n*n; i++){
        for (int j = 0; j < n*n; j++){
            Matriz[i][j] = 0;
        }
    }
}
```

Donde `Matriz`, es la matriz de entrada de $n^2 \times n^2$, después se inicia un ciclo `for i`, que recorre las filas y dentro de él, un ciclo `for j`, que recorre las columnas de cada fila, y en cada iteración se ejecuta la línea `Matriz[i][j] = 0;` que asigna el valor 0 en la posición correspondiente, de manera que al final `Matriz` queda llena de 0.

A.3. Función ceros2

Mientras que la función ceros1 genera matrices de $n \times n$, la función ceros2 genera matrices de $2n^2 \times 2n^2$, que sirven para preparar matrices que funcionarán como acumuladores en procesos de sumas sucesivas o la aproximación de Padé.

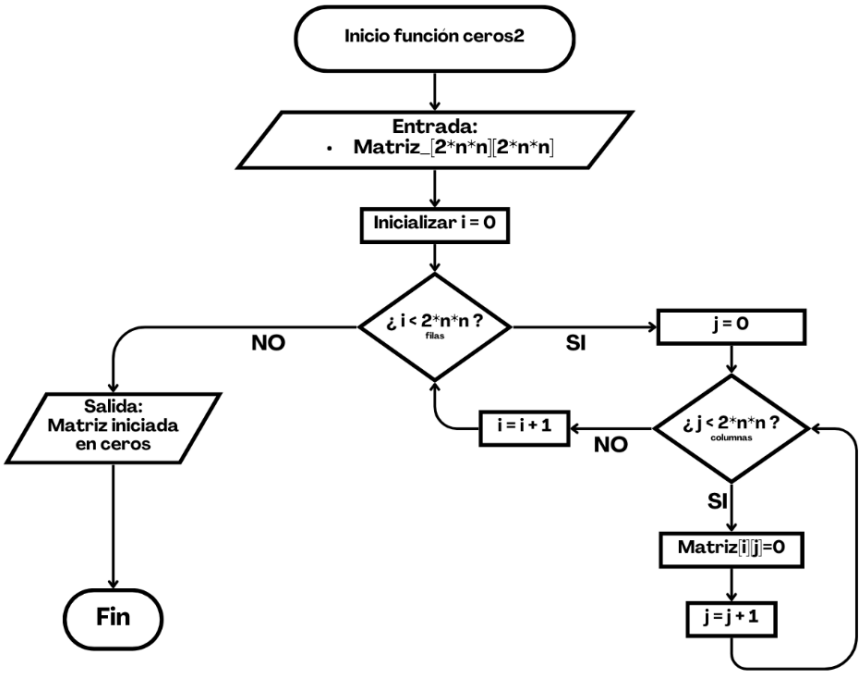


Figura A.3: Diagrama de flujo de la función ceros2.

A.4. Función identidad

El código para generar matrices identidad se basó en la ecuación (2.51), dichas matrices se utilizaron en diversos procesos durante el cálculo de la matriz de Lyapunov tipo retardada. La función identidad genera matrices de $n \times n$. En la **Error! Reference source not found.** se muestra la lógica principal utilizada para el desarrollo de la función identidad.

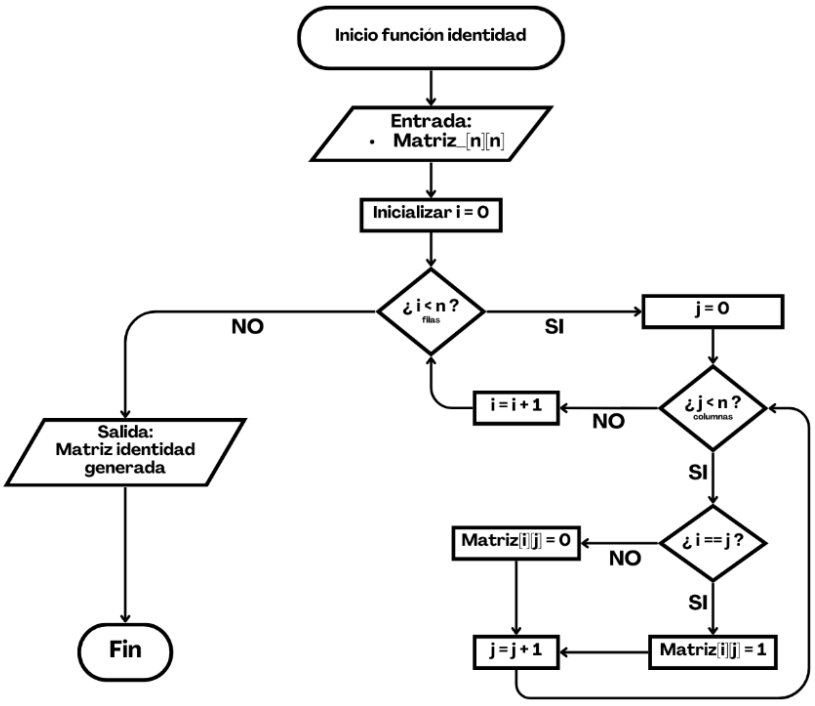


Figura A.4: Diagrama de flujo de la función identidad.

Código de la función Identidad:

```
void Identidad(float Matriz[n][n]){
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            if (i == j)
                Matriz[i][j] = 1;
            else
                Matriz[i][j] = 0;
        }
    }
}
```

Donde `Matriz[n][n]` es la matriz de entrada de $n \times n$, que se llenará con los valores de la matriz identidad; primero se inicia un ciclo `for i` que recorrerá las filas de la matriz y dentro de él se inicia un ciclo `for j` para recorrer las columnas de cada fila.

En cada iteración se evalúa la condición `if (i == j)`, si esta se cumple, `Matriz[i][j] = 1;` le asignará el valor 1 y si esta no se cumple, `Matriz[i][j] = 0;` le asignará el valor 0.

A.5. Función Identidad1

De la misma manera que la función identidad, la función Identidad1 está basada en la ecuación (2.51), por lo que la lógica de operación es similar, pero la función Identidad1 crea matrices de tamaño $n^2 \times n^2$.

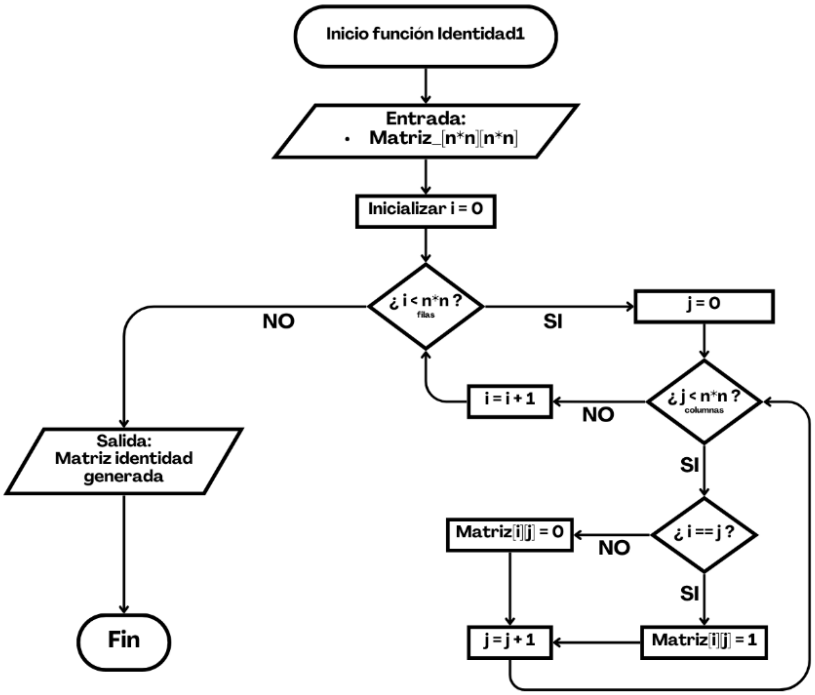


Figura A.5: Diagrama de flujo de la función Identidad1.

Código de la función Identidad1:

```
void Identidad1(float Matriz[n*n][n*n]){
    for (int i = 0; i < n*n; i++){
        for (int j = 0; j < n*n; j++){
            if (i == j)
                Matriz[i][j] = 1;
            else
                Matriz[i][j] = 0;
        }
    }
}
```

Donde `Matriz[n*n][n*n]` es la matriz de entrada de $n^2 \times n^2$, que se llenará con los valores típicos de una matriz identidad; primero inicia un ciclo `for i` para recorrer todas las filas de la matriz y dentro de él inicia un ciclo `for j` para recorrer todas las columnas de la matriz.

En cada iteración se evalúa la condición `if (i == j)` que, si se cumple, `Matriz[i][j] = 1;` asignará el valor 1 a la posición que está siendo evaluada y, si no, `Matriz[i][j] = 0;` asignará el valor 0 a la posición que está siendo evaluada.

A.6. Función Identidad2

De igual manera que las funciones identidad e Identidad1, la función Identidad2 está basada en la ecuación (2.51), pero la matriz Identidad2 genera matrices de $2n^2 \times 2n^2$. En la Figura A.6 se muestra la lógica principal utilizada para el desarrollo de la función Identidad2.

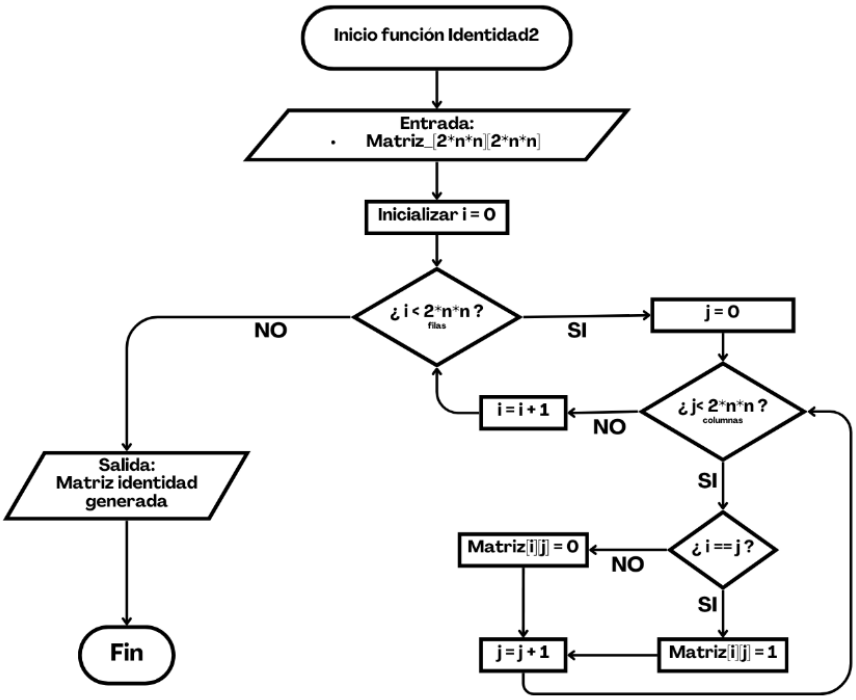


Figura A.6: Diagrama de flujo de la función Identidad2.

Código de función Identidad2:

```
void Identidad2(float Matriz[2*n*n][2*n*n]){
    for (int i = 0; i < 2*n*n; i++){
        for (int j = 0; j < 2*n*n; j++){
            if (i == j)
                Matriz[i][j] = 1;
            else
                Matriz[i][j] = 0;
```

```
            }
        }
    }
```

Donde `Matriz[2*n*n][2*n*n]` es la matriz de entrada de $2n^2 \times 2n^2$ que se llenará con los valores de una matriz identidad ampliada. Primero inicia un ciclo `for i`, que recorrerá las filas de la matriz, y dentro de él se inicia un ciclo `for j` para recorrer las columnas de cada fila, en cada iteración se evalúa la condición `if (i == j)`, si se cumple, `Matriz[i][j] = 1;` le asignará el valor 1, mientras que, si no se cumple, `Matriz[i][j] = 0;` le asignará el valor 0.

A.7. Función ceil

Debido a que no se puede contar con la función ceil de la librería de `<math.h>` se programó una función ceil propia basada en la ecuación (2.23). En la Figura A.7 se muestra la lógica de programación utilizada para el desarrollo de la función.

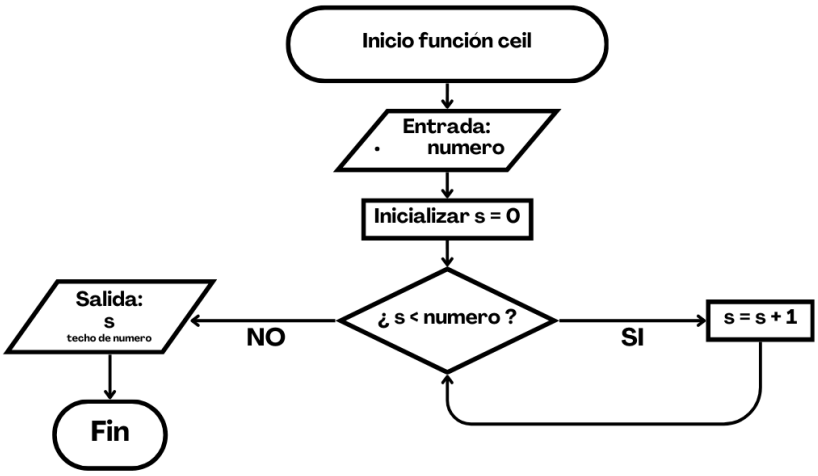


Figura A.7: Diagrama de flujo de la función ceil.

Código de la función Ceil:

```
int ceil(float numero) {
int s=0;
while (s<numero){
    s=s+1;
     }
return s;
}
```

Que inicializa una variable entera `s = 0` y se compara con el número `s<numero` y aumenta en `1` su valor, repite el proceso hasta que `s` ya no sea menor que `numero` y entonces retorna el último valor alojado en `s`.

A.8. Función Mat_esc

La multiplicación de una matriz por un escalar se basó en la ecuación (2.30). La lógica de programación utilizada para el desarrollo de la función Mat_esc se muestra en la Figura A.8.

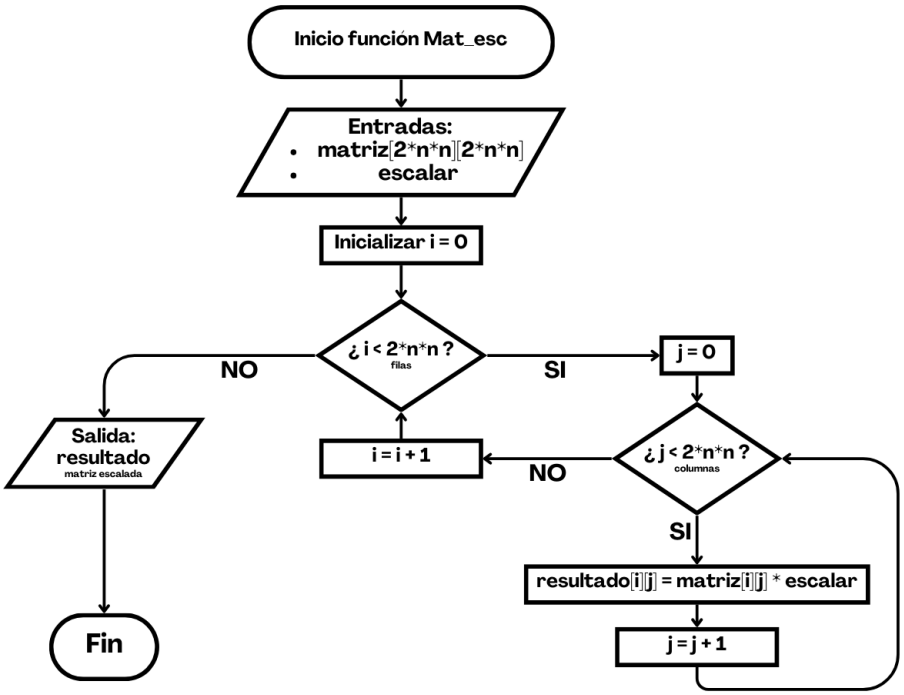


Figura A.8: Diagrama de flujo de la función Mat_esc.

Código de la función Mat_esc:

```
void Mat_esc(float matriz[2*n*n][2*n*n], float escalar, float resultado[2*n*n][2*n*n]) {
   for(int i = 0; i < 2*n*n; i++) {
      for(int j = 0; j < 2*n*n; j++) {
         resultado[i][j] = matriz[i][j] * escalar;
      }
   }
}
```

Donde `matriz [2*n*n][2*n*n]` es la matriz de entrada, `escalar` es el valor por el que se desea multiplicar cada elemento de la matriz de entrada `resultado[2*n*n][2*n*n]` es la matriz de salida donde se almacena la multiplicación de la matriz por el escalar.

Primero se inicia un ciclo `for i`, que recorre las filas de la matriz, y dentro de él un ciclo `for j` que recorre las columnas de la matriz, y en cada iteración, la línea de código `resultado[i][j] = matriz[i][j] * escalar;` multiplica el elemento correspondiente de la matriz por el valor de `escalar` y lo almacena en su posición correspondiente de la matriz de salida.

A.9. Función fact

El cálculo del factorial es básico en álgebra, no solo matricial, sino general, la función fact se basa en la ecuación (2.25). En la Figura A.9 se muestra la lógica principal utilizada para programar la función fact.

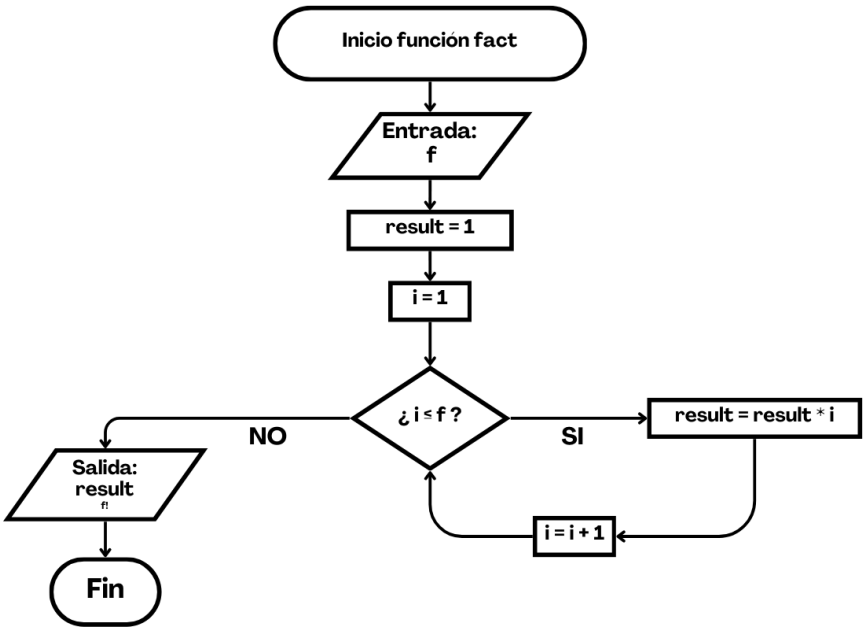


Figura A.9: Diagrama de flujo de la función fact.

Código de la función fact:

```
unsigned int fact(int f) {
    unsigned int result = 1;
    for (int i = 1; i <= f; i++) {
        result=result*i;
    }
    return result;
}
```

Donde `f` es un número entero positivo al cual se calculará su factorial, y `result` es la variable donde se almacenará el resultado de la operación. Primero se inicia `result` en 1, después un

ciclo `for i` que itera desde 1 hasta el valor alojado en `f` multiplicando en cada iteración el valor acumulado de la iteración anterior por el entero de la iteración actual; siempre que $i < f$, cuando ya no se cumpla la condición, retornará el valor actual de `f`.

A.10. Función asigna

Durante el cómputo de la matriz de Lyapunov tipo retardada se necesitó guardar el estado intermedio de una matriz para utilizarla en algún otro momento en otra operación o iteración, por lo que se programó la función asigna, que copia el contenido de una matriz de $2n^2 \times 2n^2$ en otra. En la Figura A.10 se muestra la lógica principal utilizada para el desarrollo del código de la función asigna.

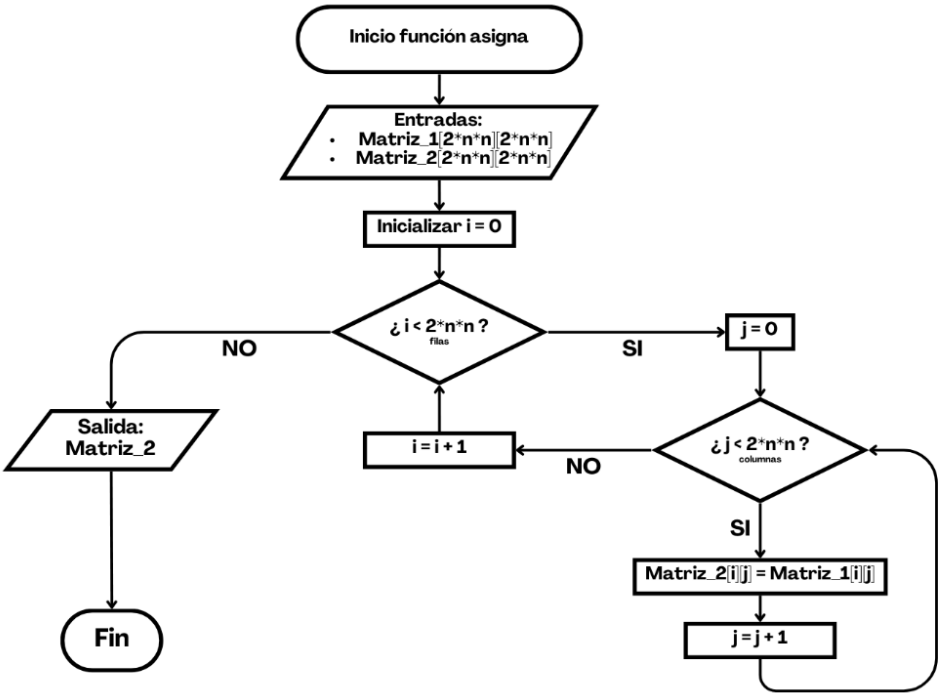


Figura A.10: Diagrama de flujo de la función asigna.

Código de la función asigna.

```
void asigna(float Matriz_1[2*n*n][2*n*n], float
Matriz_2[2*n*n][2*n*n]) {
     for (int i = 0; i < 2*n*n; i++)
                    {
                        for (int j = 0; j < 2*n*n; j++)
                        {
                        Matriz_2[i][j] = Matriz_1[i][j];
                        }
                    }
}
```

Donde `Matriz_1[2*n*n][2*n*n]` es la matriz de entrada, cuyos valores se van a copiar, `Matriz_2[2*n*n][2*n*n]`, es la matriz de salida, donde se van a almacenar los valores copiados. Primero se ejecuta un ciclo `for i`, que recorrerá las filas de la matriz de entrada, y dentro de él se ejecuta un ciclo `for j` que recorrerá las columnas y en cada iteración, la línea de código `Matriz_2[i][j] = Matriz_1[i][j];` copiará el valor que se está iterando.

A.11. Función summat

La función summat está basada en la ecuación (2.36) y suma matrices de $n \times n$. En la Figura A.11 se muestra la lógica utilizada para desarrollar el código de la función.

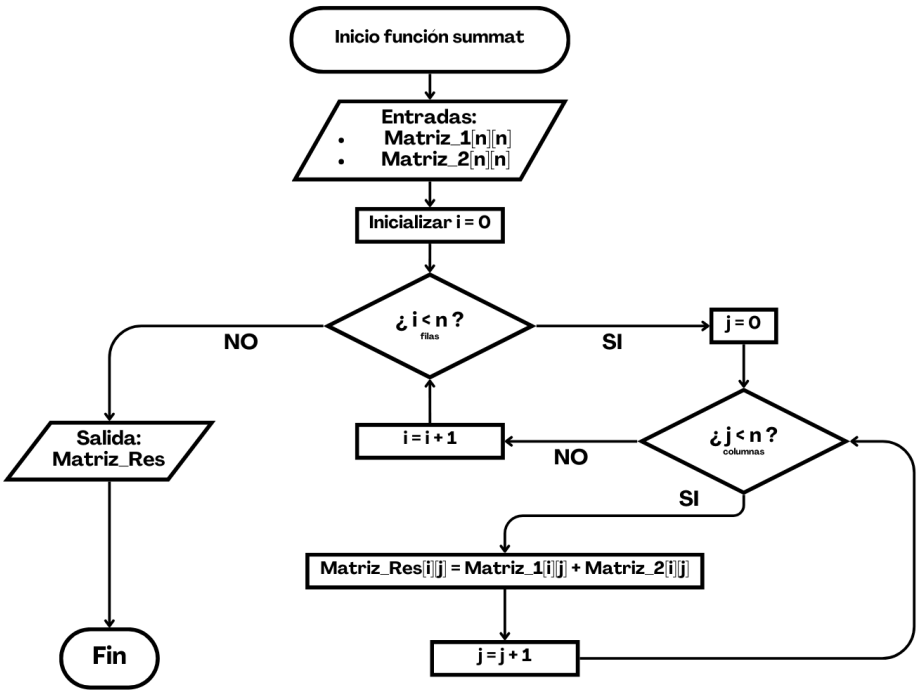


Figura A.11: Diagrama de flujo de la función summat.

Código de la función summat:

```
void summat(float Matriz_1[n][n], float Matriz_2[n][n], float
Matriz_Res[n][n]) {
     for (int i = 0; i < n; i++)
                    {
                        for (int j = 0; j < n; j++)
                        {
                        Matriz_Res[i][j] = Matriz_1[i][j] +
Matriz_2[i][j];
```

```
                    }
                }
    }
```

Donde `Matriz_1[n][n]` y `Matriz_2[n][n]` son las matrices de entrada que se van a sumar, ambas de tamaño $n \times n$, el programa inicia un ciclo `for i`, que recorre las filas de ambas matrices, y dentro de él ejecuta un ciclo `for j` para recorrer las columnas de ambas matrices y, en cada iteración, la línea de código `Matriz_Res[i][j] = Matriz_1[i][j] + Matriz_2[i][j];` va sumando los elementos de la iteración y almacenándolos en la posición respectiva de la matriz de salida `Matriz_Res[i][j]`.

A.12. Función mulmat

La función mulmat esta desarrollada para trabajar con matrices de $n \times n$. En la Figura A.12 se muestra la lógica principal utilizada para el desarrollo de la función mulmat.

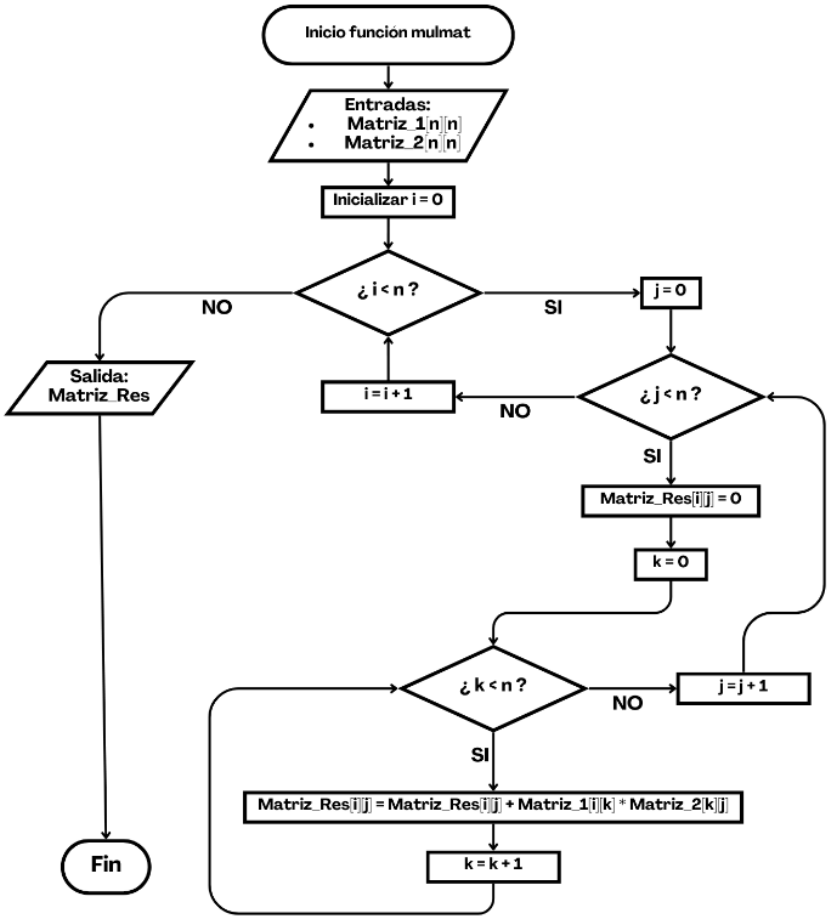


Figura A.12: Diagrama de flujo de la función mulmat.

Código de la función mulmat:

```
void mulmat(float Matriz_1[n][n], float Matriz_2[n][n], float
Matriz_Res[n][n]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            Matriz_Res[i][j] = 0;
            for (int k = 0; k < n; k++) {
                Matriz_Res[i][j] += Matriz_1[i][k] *
Matriz_2[k][j];
            }
        }
    }
}
```

Donde `Matriz1` y `Matriz2` son las matrices que se van a multiplicar (de entrada), y `Matriz_Res` es la matriz donde se guardará el resultado (de salida), y posteriormente se inician 3 ciclos for anidados, donde `for i` recorre las filas y `for j` recorre las columnas de `Matriz_Res`, para conocer la posición en la que se está iterando, y con `for k`, realiza el cálculo de la línea `Matriz_Res[i][j] += Matriz_1[i][k] * Matriz_2[k][j];` y lo almacena antes de pasar a la siguiente iteración.

A.13. Función mulmatvec2

La función mulmatvec2 fue programada para operar con matrices de dimensiones $2n^2 \times 2n^2$ y vectores de tamaño $2n^2$, esta basada en la ecuación (2.31). En la Figura A.13 se muestra la lógica principal utilizada para el desarrollo de la función mulmatvec2.

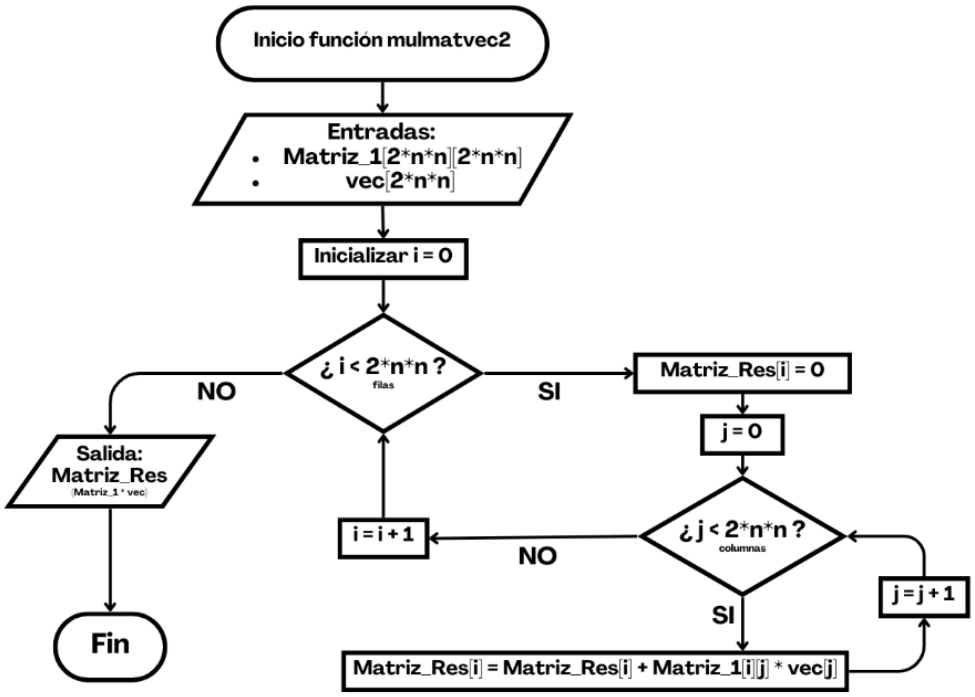


Figura A.13: Diagrama de flujo de la función mulmatvec2.

Apéndice B Código funciones.h completo

```
#define n 2

void mulmat(float Matriz_1[n][n], float Matriz_2[n][n], float
Matriz_Res[n][n]) {
     for (int i = 0; i < n; i++)
                    {
                        for (int j = 0; j < n; j++)
                        {
                        Matriz_Res[i][j] = 0;
                            for (int k = 0; k < n; k++)
                             Matriz_Res[i][j] += Matriz_1[i][k]
* Matriz_2[k][j];
                        }
                    }
}

void mulmat2(float Matriz_1[2*n*n][2*n*n], float
Matriz_2[2*n*n][2*n*n], float Matriz_Res[2*n*n][2*n*n]) {
     for (int i = 0; i < 2*n*n; i++)
                    {
                        for (int j = 0; j < 2*n*n; j++)
                        {
                        Matriz_Res[i][j] = 0;
                            for (int k = 0; k < 2*n*n; k++)
                             Matriz_Res[i][j] += Matriz_1[i][k]
* Matriz_2[k][j];
                        }
                    }
}

void mulmatvec2(float Matriz_1[2*n*n][2*n*n], float vec[2*n*n],
float Matriz_Res[2*n*n]) {
     for (int i = 0; i < 2*n*n; i++)
                    {
                           Matriz_Res[i] = 0;
                        for (int j = 0; j < 2*n*n; j++)
                        {
                            Matriz_Res[i] = Matriz_Res[i] +
Matriz_1[i][j] *vec[j];
                        }
                    }
}

void summat(float Matriz_1[n][n], float Matriz_2[n][n], float
Matriz_Res[n][n]) {
     for (int i = 0; i < n; i++)
                    {
                        for (int j = 0; j < n; j++)
                        {
```

```
                        Matriz_Res[i][j] = Matriz_1[i][j] +
Matriz_2[i][j];
                        }
                    }
}
void summat2(float Matriz_1[2*n*n][2*n*n], float
Matriz_2[2*n*n][2*n*n], float Matriz_Res[2*n*n][2*n*n]) {
     for (int i = 0; i < 2*n*n; i++)
                    {
                        for (int j = 0; j < 2*n*n; j++)
                        {
                        Matriz_Res[i][j] = Matriz_1[i][j] +
Matriz_2[i][j];
                        }
                    }
}

void asigna(float Matriz_1[2*n*n][2*n*n], float
Matriz_2[2*n*n][2*n*n]) {
     for (int i = 0; i < 2*n*n; i++)
                    {
                        for (int j = 0; j < 2*n*n; j++)
                        {
                        Matriz_2[i][j] = Matriz_1[i][j];
                        }
                    }
}
void transpuesta(float Matriz_1[n][n], float Matriz_Res[n][n]) {
     for (int i = 0; i < n; i++)
                    {
                        for (int j = 0; j < n; j++)
                        {
                      Matriz_Res[j][i] = Matriz_1[i][j];
                        }
                    }
}

int ceil(float numero) {
int s=0;
while (s<numero){
    s=s+1;
     }
return s;
}

void Identidad(float Matriz[n][n]){
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            if (i == j)
                Matriz[i][j] = 1;
            else
                Matriz[i][j] = 0;
```

```
        }
    }
}
void Identidad1(float Matriz[n*n][n*n]){
    for (int i = 0; i < n*n; i++){
        for (int j = 0; j < n*n; j++){
            if (i == j)
                Matriz[i][j] = 1;
            else
                Matriz[i][j] = 0;
        }
    }
}
void Identidad2(float Matriz[2*n*n][2*n*n]){
    for (int i = 0; i < 2*n*n; i++){
        for (int j = 0; j < 2*n*n; j++){
            if (i == j)
                Matriz[i][j] = 1;
            else
                Matriz[i][j] = 0;
        }
    }
}

void ceros1(float Matriz[n*n][n*n]){
    for (int i = 0; i < n*n; i++){
        for (int j = 0; j < n*n; j++){
            Matriz[i][j] = 0;
        }
    }
}

void ceros2(float Matriz[2*n*n][2*n*n]){
    for (int i = 0; i < 2*n*n; i++){
        for (int j = 0; j < 2*n*n; j++){
            Matriz[i][j] = 0;
        }
    }
}

void inversa(float Matriz_1[n][n], float Matriz_Res[n][n]) {
     float Matriz[n][n];
     float pivote;
     float aux;
     for (int i=0;i< n;i++)
               for (int j=0;j< n;j++){
                 Matriz[i][j]=Matriz_1[i][j];
                  if (i!=j)
                      Matriz_Res[i][j]=0;
                  else
                      Matriz_Res[i][j]=1;
               }
```

```
        for (int i=0;i< n;i++){
                pivote=Matriz[i][i];
                for(int k=0;k<n;k++){
                    Matriz[i][k]=Matriz[i][k]/pivote;
                    Matriz_Res[i][k]=Matriz_Res[i][k]/pivote;
                }
                for(int j=0; j<n; j++){
                    if (i!=j)
                    {
                        aux=Matriz[j][i];
                        for (int k=0; k<n; k++){
                            Matriz[j][k]=Matriz[j][k]-
aux*Matriz[i][k];
                            Matriz_Res[j][k]=Matriz_Res[j][k]-
aux*Matriz_Res[i][k];
                        }
                    }
                }
            }
}


void inversa2(float Matriz_1[2*n*n][2*n*n], float
Matriz_Res[2*n*n][2*n*n]) {
      float Matriz[2*n*n][2*n*n];
      float pivote;
      float aux;
      for (int i=0;i<2*n*n;i++)
                for (int j=0;j< 2*n*n;j++){
                  Matriz[i][j]=Matriz_1[i][j];
                   if (i!=j)
                       Matriz_Res[i][j]=0;
                   else
                       Matriz_Res[i][j]=1;
                }
        for (int i=0;i< 2*n*n;i++){
                pivote=Matriz[i][i];
                for(int k=0;k<2*n*n;k++){
                    Matriz[i][k]=Matriz[i][k]/pivote;
                    Matriz_Res[i][k]=Matriz_Res[i][k]/pivote;
                }
                for(int j=0; j<2*n*n; j++){
                    if (i!=j)
                    {
                        aux=Matriz[j][i];
                        for (int k=0; k<2*n*n; k++){
                            Matriz[j][k]=Matriz[j][k]-
aux*Matriz[i][k];
                            Matriz_Res[j][k]=Matriz_Res[j][k]-
aux*Matriz_Res[i][k];
                        }
                    }
```

```
                }
            }
}

float norma1(float Matriz_1[2*n*n][2*n*n]) {
     float suma, norma1;
     float vec[2*n*n];
     for (int i = 0; i < 2*n*n; i++){
          suma=0;
        for (int j = 0; j < 2*n*n; j++)
                        {
                        if (Matriz_1[j][i]<0)
                              suma=suma-Matriz_1[j][i];
                        else
                              suma=suma+Matriz_1[j][i];
                        }
                        vec[i]=suma;
                    }
     norma1=vec[0];
     for (int i = 1; i < 2*n*n; i++){
           if (vec[i]>norma1)
                norma1=vec[i];
     }
     return norma1;
}

unsigned int fact(int f) {
    unsigned int result = 1;
    for (int i = 1; i <= f; i++) {
        result=result*i;
    }
    return result;
}


void Mat_esc(float matriz[2*n*n][2*n*n], float escalar, float
resultado[2*n*n][2*n*n]) {    // Cambié float a void porque no
hay retorno
    for(int i = 0; i < 2*n*n; i++) {
        for(int j = 0; j < 2*n*n; j++) {
            resultado[i][j] = matriz[i][j] * escalar;
        }
    }
}


void productoKronecker(float Matriz_1[n][n], float
Matriz_2[n][n], float Matriz_Res[n*n][n*n]) {
    for (int i = 0; i < n*n; i++) {
        for (int j = 0; j < n*n; j++) {
            Matriz_Res[i][j] = Matriz_1[i / n][j / n] *
Matriz_2[i % n][j % n];
```

```
        }
    }
}

void MatrizPotencia(float res[2*n*n][2*n*n], float
base[2*n*n][2*n*n], int exponente) {
    float temp[2*n*n][2*n*n];
    Identidad2(res);
    for (int exp = 1; exp <= exponente; exp++) {
     for (int i = 0; i < 2*n*n; i++) {
           for (int j = 0; j < 2*n*n; j++) {
                temp[i][j] = 0;
                for (int k = 0; k < 2*n*n; k++) {
                    temp[i][j] =  temp[i][j]+res[i][k] *
base[k][j];
                }
            }
        }
        for (int i = 0; i < 2*n*n; i++) {
            for (int j = 0; j < 2*n*n; j++) {
                res[i][j] = temp[i][j];
            }
        }
    }
}

void MatExpPad(float Matriz[2*n*n][2*n*n], float
Res[2*n*n][2*n*n]){
     //sección de matriz exponencial PADE
          int m;
         float norma, tolpade = 0.5;
         float fac1;
         int p = 6, q = 6;
         float Lm[2*n*n][2*n*n];
         float NP[2*n*n][2*n*n];
         float DP[2*n*n][2*n*n];
         float aux1[2*n*n][2*n*n];
         float aux2[2*n*n][2*n*n];
         float aux3[2*n*n][2*n*n];
         float temp[2*n*n][2*n*n];

         ceros2(NP);
         ceros2(DP);

         Identidad2(aux1);
         norma=norma1(Matriz);
         m = ceil(norma/tolpade);

         if (norma>=tolpade)
          {

                 Mat_esc(Matriz,1.0/m,Lm);
```

```
          }

          else
          {
            asigna(Matriz,Lm);
          }

          // arma N de Pade, NP

          for (int j=0;j<=p;j++){
                 fac1 =  1.0*fact(p+q-
j)*fact(p)/(1.0*fact(p+q)*fact(j)*fact(p-j));
                 Mat_esc(aux1,fac1,aux2);
                 summat2(NP,aux2,NP);
                 mulmat2(aux1,Lm,aux3);
                 asigna(aux3,aux1);

          }

          // arma D de Pade, DP

          Identidad2(aux1);
          bool ban=1;
          for (int j=0;j<=q;j++){
           if (ban){
                    fac1 = 1.0*fact(p+q-
j)*fact(q)/(1.0*fact(p+q)*fact(j)*fact(q-j));
                   ban=0;
           }
           else
           {
                  fac1 = -1.0*fact(p+q-
j)*fact(q)/(1.0*fact(p+q)*fact(j)*fact(q-j));
                  ban=1;
           }

                   Mat_esc(aux1,fac1,aux2);
                   summat2(DP,aux2,DP);
                   mulmat2(aux1,Lm,aux3);
                   asigna(aux3,aux1);
             }

          // obtiene R de Pade, R11
          inversa2(DP,temp);
          mulmat2(temp,NP,aux3);
          asigna(aux3,temp);
          MatrizPotencia(aux3,temp,m);
          asigna(aux3,Res);
}
```

Apéndice C Código_Principal.c completo

```
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL46Z4.h"
#include "fsl_debug_console.h"
#include <funciones.h>

#define n 2

    float A0[n][n] = {{0, 1}, {-1, -2}};
    float A1[n][n] = {{0, 0}, {-1, 1}};
    float h=1;
    float Wvec[2*n*n] = {0, 0, 0, 0, -3, 0, 0, -3};

    float resul0[n][n];
    float resul1[n][n];
    float aux1[n*n][n*n];
    float aux11[n*n][n*n];
    float aux12[n*n][n*n];
    float aux21[n*n][n*n];
    float aux22[n*n][n*n];
    float L[2*n*n][2*n*n];
    float M[2*n*n][2*n*n];
    float N[2*n*n][2*n*n];

    float Iden[n][n];
    float R[2*n*n][2*n*n];
    float CI[2*n*n];
    float Y[2*n*n];
    float aux2[2*n*n][2*n*n];
    float incretau=0.01;

int main(void) {
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
#ifndef BOARD_INIT_DEBUG_CONSOLE_PERIPHERAL
    BOARD_InitDebugConsole();
#endif

    //PRINTF("Hello World\r\n");

    volatile static int s = 0 ;

    Identidad(Iden);

    transpuesta(A0, resul0);
```

```
transpuesta(A1, resul1);

productoKronecker(resul0, Iden, aux11);
for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
        L[i][j] = aux11[i][j];
    }
}

productoKronecker(resul1, Iden, aux12);
for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
        L[i][j+n*n] = aux12[i][j];
    }
}

productoKronecker(Iden, resul1, aux21);
for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
        L[i+n*n][j] = -aux21[i][j];
    }
}

productoKronecker(Iden, resul0, aux22);
for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
        L[i+n*n][j+n*n] = -aux22[i][j];
    }
}

Identidad1(aux1);
for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
        M[i][j] = aux1[i][j];
    }
}

ceros1(aux1);
for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
        M[i][j+n*n] = aux1[i][j];
    }
}

for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
        M[i+n*n][j] = aux11[i][j];
    }
}

for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
```

```
        M[i+n*n][j+n*n] = aux12[i][j];
    }
}

ceros1(aux1);
for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
        N[i][j] = aux1[i][j];
    }
}

Identidad1(aux1);
for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
        N[i][j+n*n] = -aux1[i][j];
    }
}

for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
        N[i+n*n][j] = aux21[i][j];
    }
}

for (int i = 0; i < n*n; i++){
    for (int j = 0; j < n*n; j++){
        N[i+n*n][j+n*n] = aux22[i][j];
    }
}

Mat_esc(L,h,aux2);
MatExpPad(aux2,R);
mulmat2(N,R,aux2);
summat2(M,aux2,R);
inversa2(R,aux2);
mulmatvec2(aux2,Wvec,CI);

char linea[256];
char temp[32];

// Imprimir encabezado CSV
sprintf(linea, "tau");
for (int i = 0; i < 2*n*n; i++) {
    sprintf(temp, ",Y%d", i);
    strcat(linea, temp);
}
strcat(linea, "\r\n");
PRINTF("%s", linea);

// Enviar los datos
int pasos = (int)(h / incretau);
for (int k = 0; k < pasos; k++) {
```

```
        float tau = k * incretau;
        int tau_int = k * (int)(incretau * 1000);
        sprintf(linea, "%d", tau_int);

        Mat_esc(L, tau, aux2);
        MatExpPad(aux2, R);
        mulmatvec2(R, CI, Y);

        for (int i = 0; i < 2*n*n; i++) {
            int yi_int = (int)(Y[i] * 100000000);
            sprintf(temp, ",%d", yi_int);
            strcat(linea, temp);
        }

        strcat(linea, "\r\n");
        PRINTF("%s", linea);
    }

    while(1) {
        s++ ;
        __asm volatile ("nop");
    }
    return 0 ;
}
```

Apéndice D Código Graficador.m completo

```
close all
clear all
clc

%Preguntar la dimensión de la matriz
n = input('Introduce la dimensión de la matriz (n): ');

%Leer el archivo CSV
datosCSV = readmatrix('datos_prueba_1.csv');

%Separar la primera columna (?) y los datos de la matriz ===
tiempo = datosCSV(:,1) / 1000;               % Convierte ? de
milésimas a segundos

datosColumnMajor = datosCSV(:, 2:(n^2 + 1)) / 100000000;   %
Datos en column-major

% === 4?? Convertir de column-major a row-major ===
datosOrdenados = zeros(size(datosColumnMajor));

for indice = 1:n^2
    fila = ceil(indice / n);
    columna = mod(indice-1, n) + 1;
    idxCM = (columna-1)*n + fila;
    datosOrdenados(:, indice) = datosColumnMajor(:, idxCM);
end

% === 5?? Crear la figura ===
figure;
set(gcf,'WindowState','maximized');

% Graficar
colores = lines(n^2);

for indice = 1:n^2
    fila = ceil(indice / n);
    columna = mod(indice-1, n) + 1;
    etiqueta = ['$Y_{' num2str(fila) num2str(columna)
'}(\tau)$'];

    plot(tiempo, datosOrdenados(:,indice), 'LineWidth', 2, ...
         'Color', colores(indice,:), ...
         'DisplayName', etiqueta);
    hold on;
end
```

Apéndice E Guía

Aquí se detallarán configuraciones del software principal MCUXpresso IDE, así como los softwares auxiliares CoolTerm y MATLAB, así como los procesos realizados por el usuario para graficar los resultados.

E.1. SDK de la FRDM-KL46Z

MCUXpresso se puede descargar de manera gratuita desde el portal oficial de NXP. Durante la instalación, es recomendable no cambiar la ubicación sugerida por el instalador y permitir que el sistema configure automáticamente las variables de entorno necesarias.

Para utilizar la tarjeta FRDM-KL46Z con MCUXpresso IDE, es necesario descargar el archivo SDK (Software Development Kit), esto lo hacemos con el siguiente procedimiento:

Abrir MCUXpresso IDE. / Download and Install SDKs.

Se selecciona la opción para ordenar la lista por tarjetas (board) y no por procesadores (processors).

1. En la lista se selecciona la tarjeta FRDM-KL46Z.

2. Se selecciona la opción de instalar (Install).

De esta manera se descargará e instalará el SDK de la tarjeta FRDM-KL46Z; el proceso funciona para todas las tarjetas que aparecen en el listado de Board.

E.2. Creación del Proyecto

Con MCUXpresso IDE listo y el SDK instalado, ya se puede empezar a trabajar en la tarjeta FRDMKL46Z. Para iniciar, se crea un proyecto:

1. Abrir MCUXpresso IDE.

2. Seleccionar la opción IDE.

Y se crea siguiendo los pasos:

File / New / New Project / MCUXpresso IDE Project / Create a New C/C++ Project / Next.

Se abrirá la ventana “Board and/or Device selection page”, donde se seleccionará la tarjeta en la que se va a trabajar, en este caso la FRDMKL46Z, una vez seleccionada, se selecciona “Next”.

Se abrirá la ventana “Configure the project”, donde se puede cambiar el nombre del proyecto, y se revisa que las opciones estén seleccionadas de la siguiente manera:

- Device Packages: MKL46Z256VMC4.
- Board: Default board files.
- Project Type: C Project.
- Project Options: UART / Import other files
- Components:
  - CMSIS Include.
  - Drivers.
  - Operating Systems.
  - Others.
  - Project Template.
  - Utilities.

Una vez que todo esté correctamente seleccionado, se puede cambiar el nombre al deseado, por ejemplo: “Codigo_Principal_Matriz_Lyapunov_tipo_retardada”, y se selecciona la opción “Use default location” y “Finish”, de esta manera se crea el proyecto nombrado “Codigo_Principal_Matriz_Lyapunov_tipo_retardada”, ya configurado para trabajar con la tarjeta FRDM-KL46Z.

E.3. Importación de los códigos

Para importar los códigos (Codigo_Principal_Matriz_Lyapunov_tipo_retardada) y (Funciones), hay varias maneras, ya sea copiando y pegando directamente los códigos sobre archivos nuevos o importándolos.

Para importarlos, dentro de MCUXpresso, se selecciona la opción IDE y se siguen los pasos:

File / Import / Next

Se abrirá la ventana “Import Projects from File System or Archive” y en la opción “Import source”, con la opción “Archive”, se abre una ventana de explorador de archivos, donde se buscará la carpeta donde se encuentra el archivo que se desea importar y, una vez seleccionado el archivo, se selecciona “Abrir” en la ventana del explorador de archivos y la opción “Select All” para importar todos los archivos y evitar errores en la construcción de los códigos, y por último seleccionamos “Finish”.

De esta manera, en el espacio de trabajo del menú “Project Explorer” de MCUXpresso IDE, aparecerá un proyecto con el nombre del archivo que se importó, dentro de él se encuentra el proyecto con los códigos “Codigo_Principal_Matriz_Lyapunov_tipo_retardada” y “Funciones” listos.

E.4. Construcción y ejecución del Codigo_Principal.c

Para ejecutar el programa principal, es necesario asegurarnos de que no contenga errores, tanto en “Codigo_Principal_Matriz_Lyapunov_tipo_retardada” como en “Funciones”. Para esto, primero abrimos los códigos en el espacio de trabajo del menú “Project Explorer”, doble clic en la carpeta con el nombre del archivo, doble clic en la carpeta source y abrimos el archivo “funciones.h”, viéndose la ruta así:

Codigo_Principal_Matriz_Lyapunov_tipo_retardada / source / funciones.h

Con el archivo abierto, buscamos la opción “build”, que está representada con un martillo, al seleccionar la opción, MCUXpresso ejecutará la construcción del programa, lo que asegura que el código está escrito correctamente y no tiene errores o posibles warnings. Para el archivo “Codigo_Principal_Matriz_Lyapunov_tipo_retardada.c”, se siguen los mismos pasos.

Cuando ambos códigos están correctamente construidos, podemos correr el código “Codigo_Principal_Matriz_Lyapunov_tipo_retardada.c” sobre la tarjeta FRDM-KL46Z, el archivo “funciones.h” no se debe correr sobre la tarjeta, pues es mandado a llamar dentro del “Codigo_Principal_Matriz_Lyapunov_tipo_retardada.c”. Para esto, buscamos el menú “debug”, representado con un escarabajo de color verde, y se despliega su menú dando clic sobre el triángulo al lado de “debug” y se sigue la ruta:

Debug Configurations / GDB PEMicro Interface Debugging / PEmicro Debugger

Dentro de este menú revisamos que las siguientes opciones estén seleccionadas:

- Interface: OpenSDA Embedded Debug – USB Port
- USB1 – OpenSDA (99FFCE09)

- Core: M0

Y en los demás submenús se deja todo seleccionado como viene por default; después, en la parte de abajo, seleccionamos “Apply” y después “Debug”, así MCUXpresso compilará y cargará a la tarjeta FRDM-KL46Z después abrirá la vista de depuración, donde, antes de ejecutarse, se pueden colocar breakpoints, ver y modificar valores o hacer un análisis antes de correrlo.

Una vez que ya se revisó todo, se presiona el botón “run”, y con el programa CoolTerm, ya configurado, creará un archivo donde se visualizan y guardan los resultados en tiempo real.

E.5. Captura de datos con CoolTerm

Para exportar los resultados obtenidos, es necesario configurar la aplicación de CoolTerm para capturar los datos enviados vía UART y guardarlos en un archivo de hoja de datos (extensión .csv).

Primero es necesario comprobar el puerto donde está conectada la FRDM-KL46Z, para eso se ingresa al buscador de dispositivos en el buscador de Windows o presionando Win + X y seleccionando el administrador de dispositivos, después se sigue la ruta:

Administrador de dispositivos / Puertos (COM y LPT)

Y el puerto que diga OpenSDA es el correspondiente, por ejemplo (COM4).

Administrador de dispositivos / Puertos (COM y LPT) / OpenSDA – CDC (COM4)

Después, en CoolTerm, se selecciona el puerto correspondiente en Options / Port y se seleccionan los datos:

- Port: COM4.
- Baud rate: 115200.
- Bits de datos: 8.
- Paridad: Ninguna.
- Stop bits: 1.

De esta manera, se configuran los parámetros de la comunicación con la tarjeta FRDM-KL46Z y puede comenzar el proceso de captura de datos.

Para iniciar la captura de datos, se selecciona *Connect* para iniciar la conexión con la tarjeta y verificamos que esté correctamente conectada y se configura el archivo que contendrá los resultados, siguiendo los pasos:

Connection / Capture to text file / Start

Abrirá una ventana donde se debe guardar el archivo con un nombre cualquiera, pero es importante darle la extensión (.csv), por ejemplo *datos_prueba_1.csv* y cambiar el tipo de archivo a *All Files(*.*)* y se guarda el archivo. De esta manera, el archivo generado será una hoja de cálculo con cada resultado en cada celda correspondiente, lo que facilitará su importación y graficación en MATLAB.

Con el archivo ya configurado, regresando a MCUXpresso, se corre el código en debug, de esta manera, los datos los enviará a CoolTerm en tiempo real y, por último, cuando el código termine de ejecutarse, se regresa a:

Connection / Capture to text file / Stop

En este punto, el archivo ya contiene los resultados y podemos cerrar tanto CoolTerm como MCUXpresso.

E.6. Graficación

En MATLAB, se crea un nuevo documento usando *ctrl+n* y se pega el código que se encuentra en el **Apéndice D** y presionando *F5*, se corre el código, MATLAB nos pedirá nombrar al archivo, puede ser cualquier nombre, en este caso se llamará "Graficador.m" y es importante guardarlo en la misma carpeta donde se tiene a *"datos_prueba_1.csv"*. MATLAB editor nos dirá que el archivo no está en la carpeta actual ni ruta de MATLAB, por lo que nos dará 2 opciones:

1. Change Folder

2. Add to Path

Para este caso es indistinto escoger alguna de las opciones, ya que ambas permitirán que el archivo se guarde en la carpeta especificada y tenga acceso a *"datos_prueba_1.csv"*, lo que permitirá graficar de cualquier manera.

Cuando se ejecute el código, la ventana de comandos tendrá el texto:

"Introduce la dimensión de la matriz (n):"

Ahí se debe escribir la dimensión de las matrices de entrada, para este caso, las matrices de entrada son de $n \times n$, donde $n = 2$, una vez que se introduzca, MATLAB graficará los resultados y abrirá una figura de tamaño pantalla completa con la gráfica generada.