BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA ELECTRÓNICA

MAESTRÍA EN CIENCIAS DE LA ELECTRÓNICA
OPCIÓN EN AUTOMATIZACIÓN

“Análisis del efecto de aceleración del proceso de diseño de sistemas electrónicos usando la teoría de control”

T E S I S

Presentada para obtener el título de:
Maestro en Ciencias de la Electrónica, Opción en Automatización

Presenta:

Lic. en Ing. Jorge Espinosa García*

Directores:

Dr. Alexandre Zemliak (FCFM-BUAP)

Dr. José Fernando Reyes Cortés (FCE-BUAP)

Puebla, México. **Enero 2021**

* Becario CONACYT.

Agradecimientos

Quiero agradecer a todas las personas que tuvieron alguna influencia en mí durante el desarrollo de esta tesis y en el transcurso de mi desarrollo en la maestría: ¡Muchísimas gracias!

Entre todas esas personas se encuentran mis padres, mi hermano y mis tíos. Gracias a mis padres por su grande apoyo, de todo tipo, en mayor medida durante el periodo de la pandemia. Gracias hermano por ese libro que me prestaste, una pequeña parte de él contribuye a este trabajo sin que supieras que así sería, además de que me trajo motivación; también gracias por animarme, quizás lo hacías sin darte cuenta. Eres un gran hermano. Agradezco también a mis tíos Fernando y Edna por su hospitalidad durante mi estancia aquí en la ciudad de Puebla.

Quiero agradecer a todos mis compañeros de la maestría que estuvieron apoyándome de una u otra forma, con consejos, alegrías, cooperación, solidaridad y aprecio. En especial, quiero agradecer a mi amigo Lorenzo por su amistad, su solidaridad, su escucha, sus consejos, su apoyo y su desprendimiento.

También quiero agradecer a todos esos amigos que estuvieron ahí conmigo, en las buenas y en las malas, varios desde lejos pero presentes.

A mi asesor Alexandre Zemliak por su trabajo, paciencia y por todo su apoyo para el desarrollo de esta tesis, gracias.

Gracias a todos mis profesores por el conocimiento y habilidades proporcionados durante la maestría. Al jurado, por sus consejos y su crítica constructiva. Y a la Coordinación de la Maestría por su apoyo.

¡Gracias BUAP!

Índice general

Índice de figuras

Índice de cuadros

Resumen

El presente trabajo de tesis tiene como objetivo general analizar las características de un efecto de aceleración del proceso de diseño de sistemas electrónicos con el propósito de definir estrategias de diseño óptimas que minimicen el tiempo computacional. Para esto se aplicará una metodología de diseño llamada Metodología General de Diseño MGD, la cual aplica conceptos de la teoría de control, y que través del uso de métodos matemáticos será posible encontrar soluciones de diseño a circuitos electrónicos pasivos con 1 resistencia y 2 resistencias, y también a un circuito activo compuesto por un transistor y resistencias en sus terminales. La aplicación se desarrolló a través de códigos escritos en $C++$ que calculan las soluciones de diseño, miden el tiempo computacional y mandan la información a que la procese MATLAB para que automáticamente genere las gráficas de las proyecciones de retratos de fase y de otras figuras con información de las curvas. Se mostrarán estos resultados gráficos y numéricos, se describirán las comparaciones del comportamiento de las trayectorias de distintos procesos de diseño. Se analizarán los resultados obtenidos y se concluirá con la descripción de algunas características que pueden ser útiles para el futuro desarrollo de algoritmos óptimos de control para diseñar circuitos electrónicos en un tiempo mínimo.

Capítulo 1

Introducción

1.1. Antecedentes

La teoría de control para resolver problemas de optimización, llamada teoría de control óptimo, ha sido aplicada desde sus inicios para resolver problemas prácticos, como por ejemplo, el movimiento de una nave espacial (Guzmán, 2000). Otra de las aplicaciones del control óptimo ha sido en la rama de la electrónica, en donde una de las problemáticas más grandes es la del diseño de sistemas complejos, por ejemplo, los circuitos VLSI, con el problema de reducir el tiempo de diseño por computadora (Zemliak, 2002). Puede ser necesario mucho tiempo de trabajo para diseñar nuevos circuitos y sistemas integrados, por lo que, el solucionar el problema de reducir el tiempo de diseño es un problema muy importante.

El diseño tradicional de los sistemas analógicos se divide principalmente en 2 partes: 1) el análisis del modelo matemático del sistema que puede ser representado a través de ecuaciones algebraicas o diferenciales y 2) el procedimiento de optimización, el cual consiste en alcanzar el punto óptimo de una función objetivo. Tanto el tiempo de procesamiento computacional para el análisis de un sistema, como el tiempo de la parte de optimización son más grandes mientras el tamaño del sistema sea mayor. Para poder resolver la primera parte del problema de diseño, la de minimizar el tiempo del análisis, se pueden utilizar métodos poderosos ya existentes. Debido a que la matriz de circuitos de gran-escala es muy dispersa, uno de esos métodos corresponde a un conjunto de técnicas especiales de matriz dispersa, utilizadas exitosamente para este propósito. Otro de estos métodos, con el que se reduce el tiempo de cómputo requerido en la solución de ecuaciones lineales y no lineales, se basa en las técnicas de descomposición. La segunda parte del problema, correspondiente al procedimiento de optimización, se puede resolver a través de métodos numéricos aplicados a la optimización con restricciones y también, sin restricciones Zemliak (2001), por ejemplo, aplicados a la optimización del diseño de circuitos VLSI (R. K. Brayton, 1981,Ruehli y col., 1982, Massara, 1991).

La estrategia de diseño de sistemas en su forma tradicional, que se ha de llamar en lo sucesivo “Estrategia Tradicional de Diseño” (ETD), incluye dos partes principales: la formulación y solución de las ecuaciones del sistema físico, como restricciones con un vector de K variables independientes más M dependientes (Zemliak, 2002) y la aplicación de algún tipo de procedimiento de optimización para el cumplimiento de los objetivos de diseño. La ETD no permite

reducir mucho el tiempo de cómputo cuando el tamaño y complejidad del sistema crecen. Otra metodología para el diseño de circuitos y sistemas, cuya representación está dada por ecuaciones no lineales, es la solución del problema sin restricciones llamada "Estrategia Tradicional de Diseño Modificada" (ETDM). Aquí el sistema de ecuaciones no es resuelto, en lugar de ello, se utilizan las llamadas funciones de penalidad en el procedimiento de optimización, simulando el sistema original.

1.2. Estado del arte o estado actual del tema en el ámbito nacional e internacional

Uno de los problemas más importantes de diseño de sistemas complejos, como el diseño de circuitos VLSI, es el problema de la reducción del tiempo de diseño por computadora. Hoy en día es necesario gastar hasta meses de trabajo usando computadoras con los multiprocesadores más modernos para diseñar nuevos circuitos. En la actualidad, se usan métodos poderosos que reducen el tiempo de análisis de un circuito. La matriz de un circuito grande es una estructura muy dispersa ("sparse matrix") y gracias a esta propiedad existen diferentes métodos especiales para matrices dispersas (Bunch y D.J. Rose, 1976, Duff y Reid, 1979 y Osterby y Zlatev, 1983). La otra posibilidad para reducir el número del cálculo computacional para ecuaciones lineales y no lineales es el uso de una técnica de descomposición. La partición de la matriz de un circuito en forma de bloques diagonales puede ser hecha por medio de la ruptura de las ramas de circuito (Wu, 1976) o por medio de la ruptura de los nodos (Sangiovanni-Vincentelli y col., 1977) y junto con los algoritmos de la solución directa pueden dar la solución del problema. La extensión de los métodos directos puede ser preparada por medio de la descomposición jerárquica y de la representación por macro modelos (Rabat y col., 1985). La otra posibilidad para poder hacer esta descomposición para sistemas no lineales incluye una técnica especial de iteraciones que fue realizada, por ejemplo, en Ruehli y col. (1982) para análisis temporal y simulación de circuitos. Una teoría más general para el diseño de sistemas es la "Metodología General de Diseño" (MGD), para sistema de ecuaciones no lineales (Zemliak, 1999). Con base en esta metodología es posible comparar los deferentes tipos de estrategias de diseño. La metodología general de diseño produce un conjunto de 2^M diferentes estrategias de diseño, en el cual la ETD y la ETDM son el primero y último elemento de este conjunto, respectivamente. Por otro lado, existe un enfoque aún más general que genera un número infinito de estrategias de diseño diferentes. Es claro que entre estas estrategias hay una o unas pocas estrategias óptimas. Defínase un algoritmo de diseño óptimo como una estrategia que alcanza los objetivos de diseño en un mínimo de tiempo computacional. Desde el punto de vista de esta teoría desarrollada, el problema de la construcción del algoritmo de diseño óptimo es un problema de tiempo- mínimo de la teoría de control óptimo. La técnica de optimización utilizada para la optimización y diseño de circuitos también influye mucho en el tiempo de cómputo total. Los métodos numéricos fueron desarrollados tanto para optimización sin restricciones como para optimización con restricciones (Fletcher, 1981). Se supone que en el futuro los métodos de análisis y métodos de optimización van a desarrollarse más.

Sin embargo, existe otra perspectiva para reducir el tiempo necesario para el diseño de

sistemas grandes. La reformulación de un problema de optimización de circuitos fue ofrecida, en nivel heurístico, hace unas décadas (Kashirskiy, 1976 y Kashirskiy y Trokhimenko, 1979). Este proceso fue llamado como optimización general y estuvo basado en la idea de ignorar las leyes de Kirchhoff para una parte del sistema o incluso para el sistema completo, siendo en este último caso donde todas las ecuaciones del sistema inicial desaparecen. Esta técnica fue desarrollada en un aspecto práctico para optimización de un circuito de microondas (Rizzoli y col., 1990) y para la síntesis de un circuito analógico de alta ejecución (Ochotta y col., 1996).

1.3. Planteamiento del problema y justificación

Uno de los principales problemas del diseño de un sistema grande, es el excesivo tiempo de cómputo que es necesario para alcanzar el punto óptimo del proceso de diseño. Este problema tiene una gran importancia debido a que existen demasiadas aplicaciones, por ejemplo, en el diseño de circuitos electrónicos integrados VLSI. Desde el punto de vista del tiempo de cómputo, la estrategia de diseño óptimo puede ser definida como una estrategia que alcanza el valor óptimo de la función objetivo del proceso de diseño en un tiempo de cómputo mínimo. Una de las preguntas principales en este camino es cómo es posible encontrar condiciones especiales que permitan construir el algoritmo óptimo en tiempo. La respuesta a esta pregunta permite una reducción significativa del tiempo de cómputo. La idea de usar la teoría de control óptimo fue desarrollada en unos trabajos (Zemliak, 2001) utilizando el método de optimización gradiente, luego para el método de Newton y para el método de Davidon-Fletcher-Powell (Fletcher, 1981). Esta concepción generaliza el proceso de diseño y produce diferentes estrategias y diferentes trayectorias de diseño dentro del mismo procedimiento de optimización. El problema de la búsqueda del algoritmo óptimo en tiempo se define como un problema del tiempo mínimo de la teoría de control óptimo. Resultados numéricos del diseño de varios circuitos electrónicos muestran efectividad potencial de una nueva metodología de proceso de diseño de un sistema electrónico (Zemliak, 2014). En estos trabajos fue demostrado que en el caso general la nueva formulación del problema de diseño no tiene dependencia del método de optimización y se observa solo un cambio cuantitativo de los resultados cuando se cambió el método de optimización. Con base de un efecto de aceleración (Zemliak, 2004) aparece una buena posibilidad para reducir el tiempo de diseño de sistemas electrónicos. Para poder usar este efecto hay que estudiar las propiedades y la estructura del posible algoritmo óptimo de diseño.

1.4. Propuesta para la solución del problema planteado

En este trabajo se propone que la teoría de control óptimo es aplicada para el diseño de circuitos electrónicos. Esta teoría permite hacer una generalización del problema de diseño. En el marco de la metodología generalizada aparece un conjunto de distintas estrategias de diseño que tienen distinto tiempo computacional. Se puede definir el problema de búsqueda de estrategias dentro de este conjunto. Aplicando la metodología generalizada de diseño se puede definir el proceso de diseño de circuitos electrónicos como un sistema dinámico controlable. Un instrumento principal para cambiar el procedimiento de diseño es un vector de control

introducido artificialmente. Cambiando los componentes de este vector se puede obtener un efecto especial de aceleración del proceso de diseño. El análisis de las características de este efecto permite obtener una ganancia muy grande de una estrategia especial comparando con la estrategia tradicional. Para obtener esta ganancia hay que analizar las características del efecto de aceleración y estudiar las condiciones necesarias y suficientes de la existencia de este efecto.

1.5. Objetivo general

Analizar las características de un efecto de aceleración del proceso de diseño de sistemas electrónicos para definir las estrategias de diseño óptimas para minimizar el tiempo computacional.

1.5.1. Objetivos particulares

- Estudiar los métodos matemáticos y algoritmos para el análisis de sistemas electrónicos no lineales.
- Obtener conocimiento sobre una nueva metodología de diseño de sistemas electrónicos con base de la teoría de control.
- Desarrollar algoritmos de diseño de circuitos electrónicos con base de la estrategia general de diseño usando vector de control como instrumento principal.
- Desarrollar los programas para computadoras compatibles con PC IBM para estrategias de diseño que realizan un efecto de aceleración.
- Estudiar las condiciones necesarias y suficientes para obtener un efecto de aceleración.
- Preparar las recomendaciones para reducir el tiempo de cómputo.
- Demostrar la ganancia de tiempo de cómputo aplicando un efecto de aceleración.
- Publicar los resultados.

Capítulo 2

Marco Teórico

2.1. Estrategia Tradicional de Diseño (ETD)

En este capítulo se presenta la estrategia tradicional de diseño. El proceso de diseño para un sistema con esta metodología se puede definir como el problema de minimizar una función objetivo tradicional $C(\mathbf{X})$, donde $\mathbf{X} \in \mathbb{R}^N$, con un sistema de restricciones. Se asume que el mínimo de la función objetivo tradicional $C(\mathbf{X})$ garantiza que se obtienen los valores apropiados de los elementos del circuito y que el conjunto de restricciones es el modelo matemático del sistema.

La topología del sistema físico corresponde con la descripción dada por el modelo matemático constituido por un sistema de ecuaciones, en su caso más general, no lineales. Tales ecuaciones relacionan variables independientes con dependientes y para algún circuito electrónico puede ser modelado como se muestra a continuación:

$$g_j(\mathbf{X}) = 0, \qquad j = 1, 2, \ldots, M \tag{2.1}$$

donde K y M son el número de variables independientes y dependientes respectivamente, considerando que N es el número total de variables en el sistema y además que $N = K + M$.

El vector $\mathbf{X}$ está compuesto por dos partes: $\mathbf{X} = (\mathbf{X}', \mathbf{X}'')$, donde $\mathbf{X}' \in \mathbb{R}^K$ es el vector de variables independientes y $\mathbf{X}'' \in \mathbb{R}^M$es el vector de variables dependientes. Cabe señalar que esta separación es relativa, ya que cualquier variable puede ser definida como independiente o dependiente. Si se describe un sistema electrónico resulta natural definir los elementos del sistema como variables independientes y los parámetros físicos, como voltajes o corrientes, como variables dependientes, pero esto no es una limitación.

El proceso de optimización para minimizar la función objetivo tradicional $C(\mathbf{X})$ está definido en general por la siguiente ecuación vectorial para el proceso de dos pasos:

$$\mathbf{X}^{s+1} = \mathbf{X}^s + t_s H^s \tag{2.2}$$

con las restricciones dadas en la ecuación 2.1, y donde s es el número de iteración o ciclo en turno, t_s es el parámetro de escala para la iteración, $t_s \in \mathbb{R}^1$, el vector H es una funcional o función de $C(\mathbf{X})$ que representa la dirección de cambio descendente de la función objetivo, en

este caso, se asume que una estrategia descendente es aplicada. La estructura matemática de la función H está determinada por el método de optimización que se emplea, por ejemplo, algunos métodos de optimización son el método del gradiente, el método de Newton o el método de Davidon-Fletcher-Powell (DFP). En el presente trabajo se utilizó el método de gradiente. Este proceso es una formulación típica de un problema de optimización con restricciones, donde el sistema definido por la ecuación 2.1 es resuelto en cada paso del proceso de optimización con M variables dependientes y K independientes.

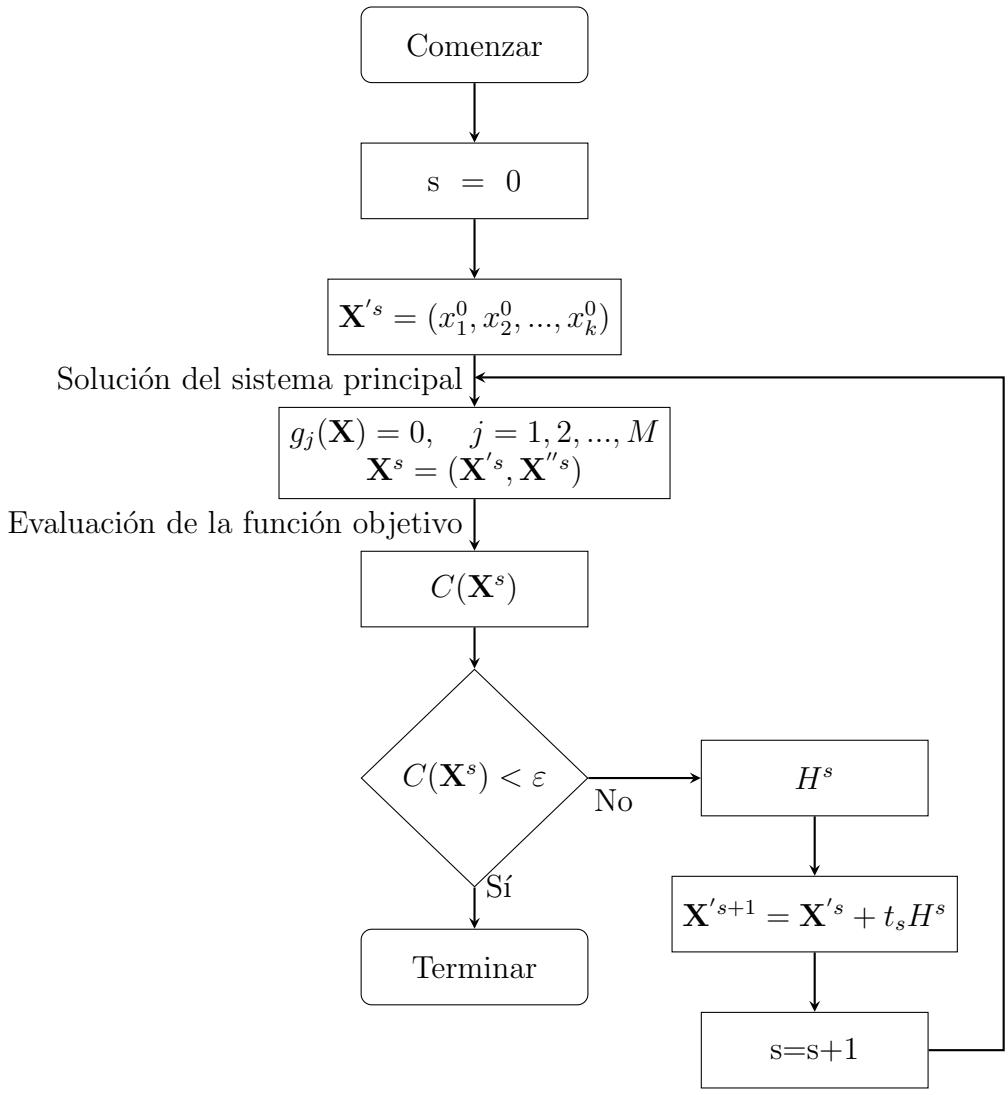


Figura 2.1: Estrategia tradicional de diseño

2.2. Solución del sistema no lineal

2.2.1. Método de iteraciones simples

Sea $||argumento||$ la notación utilizada para representar cualquier tipo de norma: si el argumento es matriz, se trata de alguna norma matricial; si el argumento es vector, entonces la norma es vectorial (norma euclidiana, norma-1, norma-∞, etc.).

El método de iteraciones simples (también llamado método de punto fijo, método $\mathbf{X} = g(\mathbf{X})$ o método de iteración de punto fijo) es utilizado para obtener la raíz del sistema de ecuaciones $\mathbf{F}(\mathbf{X}) = \mathbf{0}$, con $\mathbf{X}$ como vector.

Basándonos en la información dada en Nieves y Domínguez (2014), Gerald y Wheatley (2004) y en Henrici (1964) se describe en el siguiente párrafo este método que estriba en llevar al sistema de ecuaciones $\mathbf{F}(\mathbf{X}) = \mathbf{0}$ a la forma

$$\mathbf{X} = \mathbf{G}(\mathbf{X}) \tag{2.3}$$

donde $\mathbf{X} \in \mathbb{R}^n$ y $\mathbf{G}(\mathbf{X}) : \mathbb{R}^n \to \mathbb{R}^n$. Un punto $\mathbf{X}$ con el cual la ecuación 2.3 se cumple se llama punto fijo (Clapp, 2015), el cual denotaremos con $\overline{\mathbf{X}}$. Para encontrar el punto fijo se lleva la ecuacion 2.1 a la forma de la ecuación 2.2, de este modo se tiene un algoritmo que utliza la ecuación 2.4 de manera recursiva para encontrar la (k+1)-ésima estimación que sea aproximadamente igual al punto fijo: $\mathbf{X}^{k+1} \approx \overline{\mathbf{X}}$ (Nieves & Domínguez, 2014). El punto fijo es la solución al sistema de ecuaciones $\mathbf{F}(\mathbf{X}) = \mathbf{0}$. Según Henrici (1964), para ejecutar el algoritmo, primero se elige un valor arbitrario $\mathbf{X}^0$ esperando con suerte dar con el valor de la raíz, en caso contrario se calcula $\mathbf{X}^1$ y se vuelve a revisar si es raíz, de no ser así, el proceso se repite hasta que $\mathbf{X}^{k+1}$ converja lo suficiente a la raíz $\overline{\mathbf{X}}$, o lo que es lo mismo, que $\overline{\mathbf{X}} \approx \mathbf{G}(\overline{\mathbf{X}})$.

$$\mathbf{X}^{k+1} = \mathbf{G}\left(\mathbf{X}^k\right) \tag{2.4}$$

Tómese en cuenta que la secuencia $\{\mathbf{X}^{k+1}\}$ que se elige debe estar bien definda sobre algún intervalo, converger y su límite ser solución a la ecuación 2.3 (Henrici, 1964). Para verificar las últimas dos cualidades de la secuencia $\{\mathbf{X}^{k+1}\}$ puede utilizarse el teorema de punto fijo de Banach (principio de contracción, véase el apéndice A) comprobando que la secuencia es una contracción definida dentro de un espacio métrico completo, no vacío, por ejemplo, en un intervalo de $\mathbb{R}^n$ (Clapp, 2015). Para comprobar que una función es contracción, se puede aplicar la desigualdad de la ecuación A.5 de la definción de contracción o su ecuación equivalente en el espacio $\mathbb{R}$ (ecuación A.2), condición Lipschitz con la constate $L < 1$. También se puede probar que la secuencia es una contracción si cualquier norma matricial del Jacobiano de $\mathbf{G}(\mathbf{X})$ aplicada en $\mathbf{X}$ es menor o igual que una constante Lipschitz menor que 1 (Wang, 2015), vea la ecuación 2.5.

$$||J_{\mathbf{G}(\mathbf{X})}|| \leq L < 1 \tag{2.5}$$

Por lo que una condición suficiente pero no necesaria es que la norma espectral, la norma-∞ o la norma-1 aplicadas al Jacobiano de la función $G(X)$ en $\mathbf{X}$ sean menor que 1.

La manera de obtener la ecuación 2.3 afecta la convergencia del proceso iterativo. En caso de existir la convergencia, ésta es de primer orden (Nieves & Domínguez, 2014).

2.2.2. Método de Newton-Raphson

El método de Newton-Raphson es un procedimiento que resuelve un sistema no lineal de n ecuaciones y n incógnitas ($n \in \mathbb{N}$) (Nieves & Domínguez, 2014):

$$\mathbf{F}(\mathbf{X}) = \mathbf{0}, \tag{2.6}$$

donde

$$\mathbf{F}(\mathbf{X}) = \begin{bmatrix} f_1(\mathbf{X}) \\ f_2(\mathbf{X}) \\ \vdots \\ f_n(\mathbf{X}) \end{bmatrix} \tag{2.7}$$

con $\mathbf{X}$ como vector de n variables

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{2.8}$$

Este método da la solución aproximada al sistema de ecuaciones (2.6), basándose en el principio de linealización sucesiva. La linealización sucesiva requiere que un problema no lineal difícil sea reemplazado por una secuencia de problemas lineales más simples cuyas soluciones converjan a la solución del problema no lineal (Miranda & Fackler, 2002). Se reemplaza a la función vectorial no lineal $\mathbf{F}(\mathbf{X})$ con su aproximación en series de Taylor de primer orden (ecuación 2.10), quedando de esta forma un problema lineal de búsqueda de raíces (Miranda & Fackler, 2002). Esa aproximación (sistema de ecuaciones lineales 2.10) está formada por la matriz jacobiana (matriz de derivadas parciales (ver ecuación 2.9)) y el vector de funciones $\mathbf{F}(\mathbf{X})$, quedando de manera desarrollada como se muestra en la ecuación 2.12 (Nieves & Domínguez, 2014).

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{2.9}$$

$$J\mathbf{h} = -\mathbf{F}, \tag{2.10}$$

En la ecuación 2.10, $\mathbf{h}$ es la diferencia de dos valores de $\mathbf{X}$ de consecutivos instantes de tiempo: $k+1$ y k, véase la ecuación 2.11.

$$\mathbf{h} = \mathbf{X}^{k+1} - \mathbf{X}^k, \tag{2.11}$$

$$\begin{aligned} \tfrac{\partial f_1}{\partial x_1}h_1 + \tfrac{\partial f_1}{\partial x_2}h_2 + \cdots + \tfrac{\partial f_1}{\partial x_n}h_n &= -f_1 \\ \tfrac{\partial f_2}{\partial x_1}h_1 + \tfrac{\partial f_2}{\partial x_2}h_2 + \cdots + \tfrac{\partial f_2}{\partial x_n}h_n &= -f_2 \\ &\vdots \\ \tfrac{\partial f_n}{\partial x_1}h_1 + \tfrac{\partial f_n}{\partial x_2}h_2 + \cdots + \tfrac{\partial f_n}{\partial x_n}h_n &= -f_n \end{aligned} \tag{2.12}$$

$$[\ J\ |\ -\mathbf{f}\ ] \tag{2.13}$$

De acuerdo a Nieves y Domínguez (2014) se tiene que para resolver este problema lineal de búsqueda de raíces se han de seguir los siguientes pasos:

1. Determinar el máximo número de iteraciones a permitirse en el procedimiento.

2. Definir el criterio de convergencia a utilizarse.

3. Fijar el valor inicial de $\mathbf{X}$.

4. Evaluar la matriz jacobiana aumentada (2.13) del sistema de ecuaciones en $\mathbf{X}_0$ actual.

5. Calcular el nuevo valor de $\mathbf{X}$ a través de la ecuación (2.14).

6. Verificar si el nuevo valor converge a la solución.

   a) Si converge, entonces el procedimiento termina.

   b) Si no converge y el número de iteraciones es menor al máximo, entonces repetir los pasos 4, 5 y 6.

   c) Si no converge y el número máximo de iteraciones se ha alcanzado, entonces comenzar el procedimiento desde el paso 3.

El nuevo valor del vector $\mathbf{X}$ se obtiene al resolver el sistema de ecuaciones 2.12 y al aplicar la ecuación 2.14, conforme a la descripción dada por Nieves y Domínguez (2014).

$$\mathbf{X}^{k+1} = \mathbf{X}^k + \mathbf{h}^k \tag{2.14}$$

El método de Newton se puede describir mediante el diagrama de flujo de la figura 2.2. Obsérvese que como condición de convergencia se ha utlizado a $||\mathbf{X}_n - \mathbf{X}|| > \epsilon$, con un pequeño valor de $\epsilon > 0$.

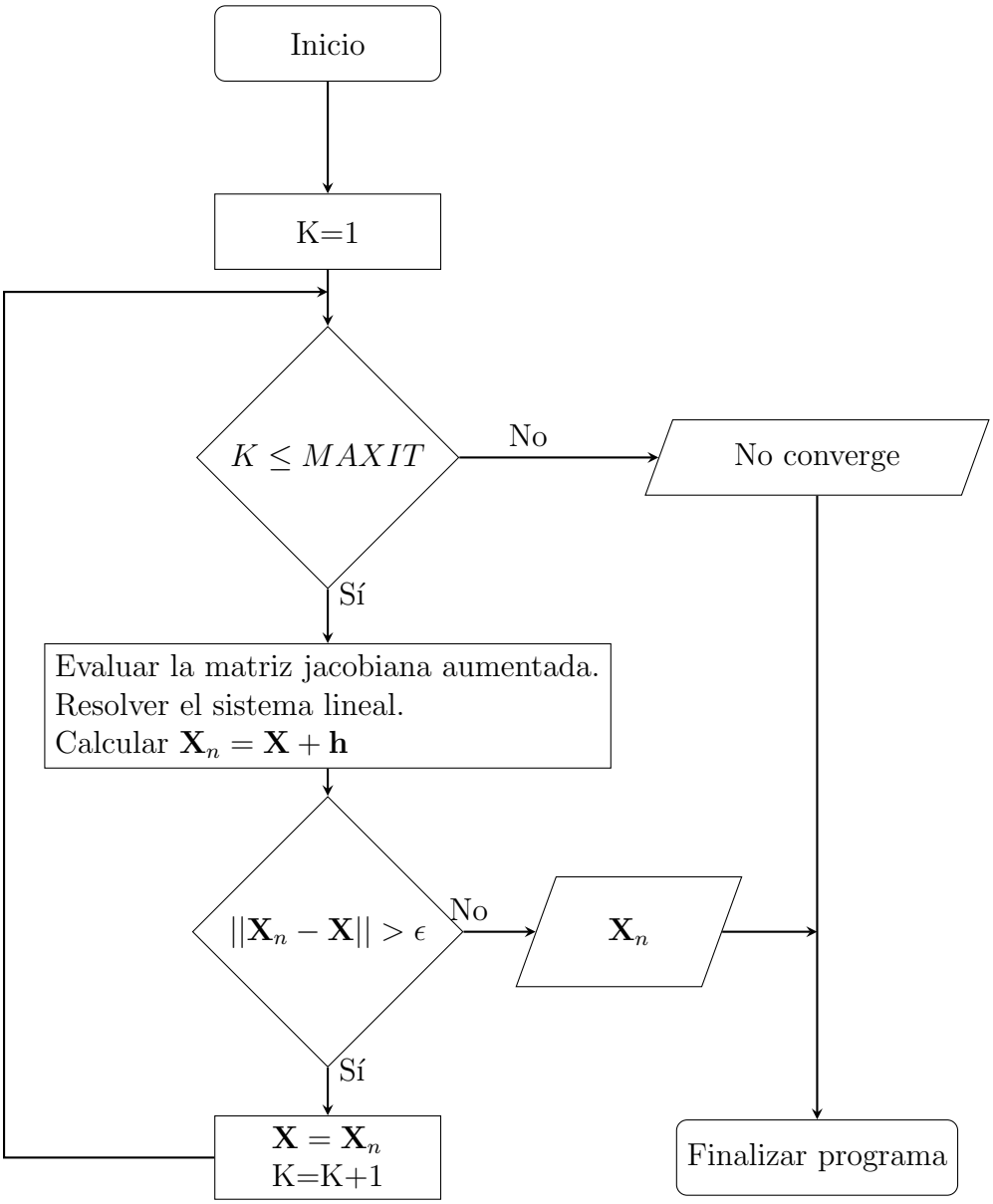


Figura 2.2: Diagrama de flujo del método de Newton

Se deben tomar en consideración las fallas que tiene este método, el cual a veces no converge (Nieves & Domínguez, 2014). En el caso bidimensional, si el método en lugar de converger oscila, se puede deber a que alguna de las raíces no es real, la raíz es un punto de inflexión o las iteraciones quedaron atrapadas en alguna zona porque el valor inicial estuvo muy lejos de la raíz buscada (Nieves & Domínguez, 2014). Este método es poco confiable para problemas de múltiples variables, cuando no se ha elejido un valor inicial cercano al valor final, pues en esos abundan los puntos de silla, lo que ocasiona que el método avance hacia algún punto de silla y luego diverja (James & Dyke, 2018).

2.3. Integración de ecuaciones diferenciales no lineales

2.3.1. Método de Euler

El método de Euler es un método numérico utilizado para la resolución de un problema de valor inicial, esto es: resolver una ecuación diferencial de primer grado (ecuación 2.15), dando una condición inicial (ecuación 2.16) (valores conocidos) y el valor x_f donde se quiere conocer el valor de la variable dependiente $y(x_f)$. La solución general a la ecuación 2.15 es $F(x, y, c) = 0$ "representa una familia de curvas en el plano x-y"(Nieves & Domínguez, 2014); cambiando el valor de la constante c se obtiene cada una de esas curvas. A este método también se le conoce como método de Euler-Cauchy y puede verse como el caso de orden uno del algoritmo de Taylor (Henrici, 1964).

$$\frac{dy}{dx} = f(x, y) \tag{2.15}$$

$$y(x_0) = y_0 \tag{2.16}$$

El método de Euler puede explicarse de manera geométrica. Consiste en utilizar la ecuación 2.17 con la que se genera una función aproximada a la curva $F(x, y, c) = 0$ "por medio de una serie de segmentos de línea recta"(Nieves & Domínguez, 2014). Se traza una recta con una pendiente definida por la ecuación 2.15 a partir del punto $P_i = (x_i, y_i)$ y se interseca con la abscisa de x_{i+1}; la intersección es el punto siguiente $P_{i+1} = (x_{i+1}, y_{i+1})$ del algoritmo de Euler. Tómese en cuenta que la distancia entre cada valor de x_i consecutiva es $h = x_{i+1} - x_i$, cuyo intervalo se define con la ecuacíón 2.18, la cual indica que el intervalo comprendido entre el valor incial x_0 y el valor final x_f se divide en n intervalos, con $n \in \mathbf{N}$. El algoritmo comienza en $i = 0$ y se repite la evaluación de la ecuación 2.17 al aumentar la iteración i en $i = i + 1$ hasta $i = n$ para obtener $y(x_n) = y_n$. En la figura 2.3 se compara una solución exacta $F(x, y, c) = 0$ con la solución numérica obtenida con el método de Euler.

$$y_{i+1} = y_i + hf(x_i, y_i) \tag{2.17}$$

$$h = \frac{x_f - x_0}{n} \tag{2.18}$$

Tómese en cuenta que y_n es una aproximación del valor buscado $y(x_f) = y_f$, $y(x_f) \approx y(x_n)$. La solución exacta se da en la ecuación 2.19 a través de la aplicación de la serie de Taylor a la función $y(x)$ (James, 2015).

$$y_{i+1} = y_i + hf(x, y) + O(h^2) \tag{2.19}$$

El término $O(h^2)$ corresponde a todos los términos que involucran potencias de h mayores o iguales que 2 dentro de la serie de Taylor (James, 2015). El error $O(h^2)$ es local, pero después de 'muchos' pasos el error global es $O(h)$ (Gerald & Wheatley, 2004).

En la figura 2.4 se explica el método de Euler mediante un diagrama de flujo.

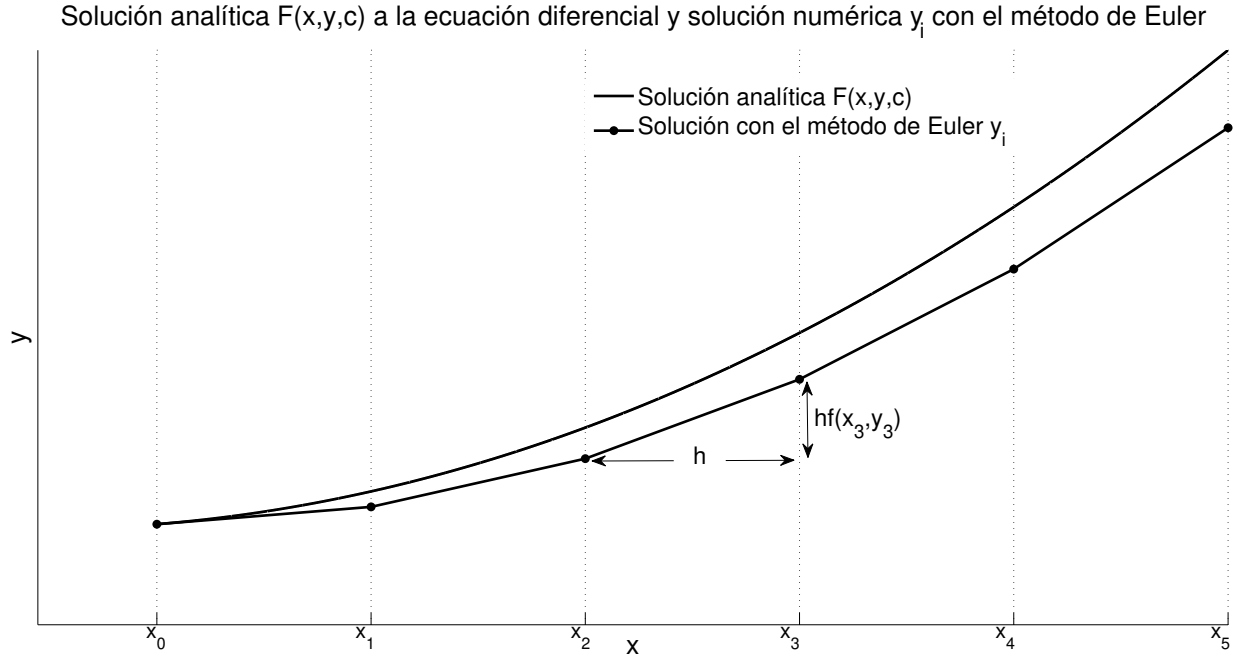


Figura 2.3: Solución analítica y solución numérica con el método de Euler

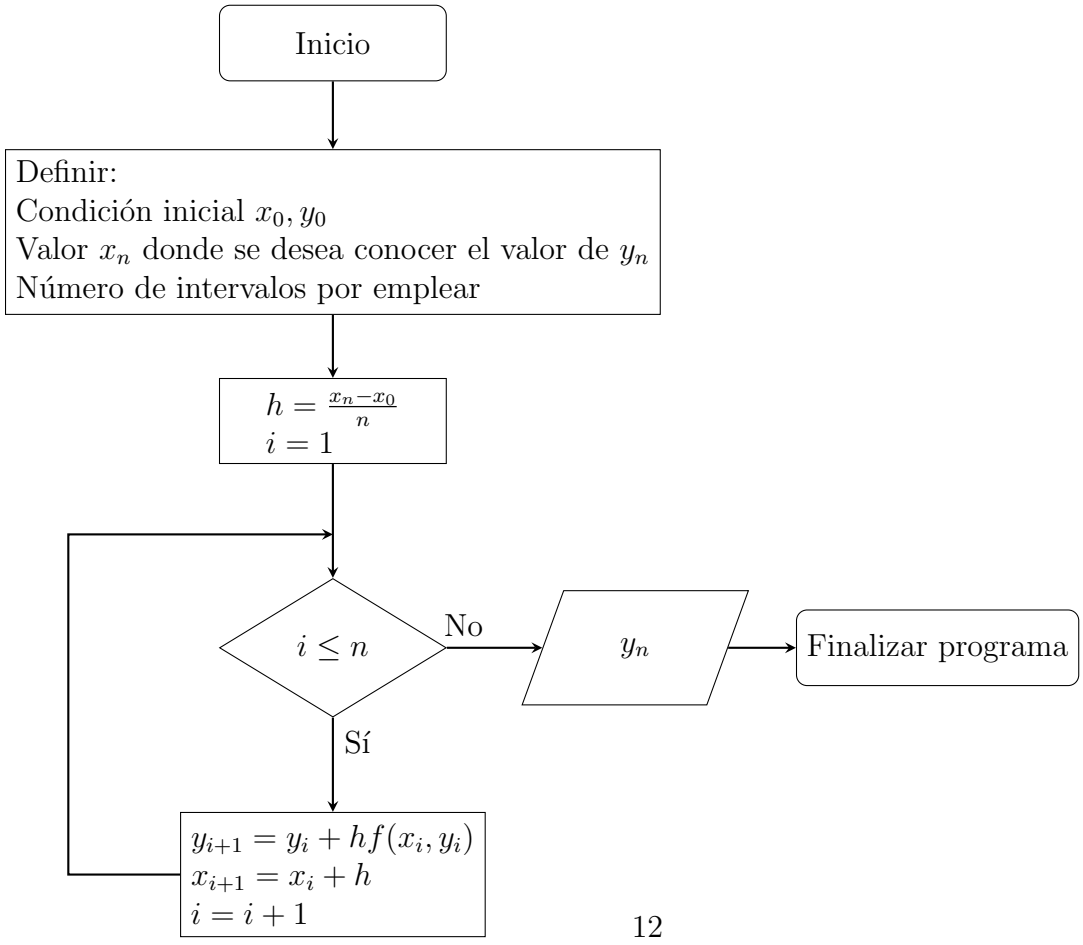


Figura 2.4: Diagrama del Método de Euler

El método que se ha utilizado como guía en esta trabajo ha sido el método de Euler (el más simple de los métodos de Runge-Kutta) debido a su fácil aplicación y menor número de cálculos, en comparación a otros métodos como los métodos de Runge-Kutta, explicados en el apéndice B. Para éstos, se requiere el cálculo de derivadas de mayor orden que con el método de Euler, el cual sólo requiere de la primera derivada. Aunque los otros métodos de Runge-Kutta pueden resultar más precisos que el de Euler.

2.4. Optimización como herramienta de diseño

2.4.1. Métodos de búsqueda directa

En los métodos de búsquda directa, sus restricciones son manejadas de una manera explícita y no requieren de la información del gradiente de la función objetivo (Rabat y col., 1985 y Baeyens y col., 2016). Entre estos métodos se tiene el de programación lineal, el método simplex y su variante tabular, todos éstos descritos en el apéndice C.

2.4.2. Métodos de búsqueda indirecta

Con los métodos de búsqueda indirecta, el problema con restricciones es resuelto como una secuencia de problemas de minimización sin restricciones (Rao, 1996). Entre estos métodos se tienen los de 1er. orden, como el método del gradiente explicado en la siguiente sección, y los métodos de 2do. orden, tales como el método de Newton-Raphson aplicado a problemas de optimización y el método Cuasi-Newton. Los dos últimos explicados en el apéndice D.

2.4.2.1. Métodos de 1er. orden

Gradiente

El método de gradiente descrito y utilizado en este trabajo es el llamado método del descenso más rápido o método del descenso de máxima pendiente. Éste es un método que encuentra el mínimo de una función objetivo, elegiendo para cada iteración la dirección de búsqueda local más inclinada, es decir, la dirección en la cual la función objetivo tiene un mayor cambio (James y Dyke, 2018 y Nieves y Domínguez, 2014).Para ello, se obtiene el gradiente del punto actual (Nieves & Domínguez, 2014); si el gradiente no está disponible, otra posibilidad es evaluarlos de manera numérica (James & Dyke, 2018). Matemáticamente se puede representar el vector gradiente de la función objetivo $\mathbf{z}$ como $\mathbf{D} = \nabla \mathbf{z}$, el cual da la dirección del ascenso más brusca, pero como se quiere la dirección de descenso más rápida, entonces se multiplica el gradiente por menos uno, obteniéndose la ecuación 2.20.

$$\mathbf{d} = -\nabla \mathbf{z} \tag{2.20}$$

Después de encontrar la dirección más adecuada se procede por avanzar una cierta distancia para calcular el nuevo punto a partir del cual se buscará la nueva dirección; de este modo, se sigue iterativamente hasta alcanzar el mínimo de la función objetivo (Nieves & Domínguez,

2014). En el presente trabajo se está utilizando la forma de la ecuación 2.21 para elegir el valor del próximo punto en cada iteración.

$$\mathbf{X}^{s+1} = \mathbf{X}^s + t_s \mathbf{d}^s \tag{2.21}$$

En James y Dyke (2018) se afirma que este método tiene la gran ventaja de ser simple y seguro, pero con la desventaja de ser muy lento, en particular cerca del punto óptimo.

2.5. Teoría General de Diseño

2.5.1. La Estrategia Tradicional de Diseño Modificada

En la estrategia tradicional modificada, el problema de la optimización con restricciones se transforma a uno sin restricciones para K variables, si el sistema 2.1 es resuelto para M variables dependientes del vector $\mathbf{X}$. El problema de optimización sin restricciones tiene lugar en el espacio $\mathbb{R}^K$ de variables independientes (Zemliak, 2001), donde el sistema 2.1 se resuelve en cada paso del proceso de optimización, en este caso:

$$\mathbf{X'}^{s+1} = \mathbf{X'}^{s} + t_s H^s \tag{2.22}$$

Para sistemas electrónicos, el carácter específico de los procesos de diseño asegura que no es necesario satisfacer las condiciones establecidas en el sistema de ecuaciones 2.1 para todos los pasos del proceso de optimización, es suficiente satisfacerlas en el punto final del proceso de diseño. El problema de resolver las ecuaciones 2.1 y 2.22 puede redefinirse de tal forma que las variables dependientes sean consideradas con el mismo comportamiento de las variables independientes, así el problema de optimización sin restricciones no requiere solucionar el sistema en cada paso, pero también se puede hacer uso de un método de funciones de compensación o penalidad (penalti functions) que se deben incluir en la función objetivo del sistema. Así la función vectorial H, además de ser una función de la función objetivo tradicional $C(\mathbf{X})$, también lo será de la función de compensación adicional $\varphi(\mathbf{X})$. La estructura de la función de compensación incluye la información del sistema de ecuaciones 2.1 y por lo general está definida de la forma mostrada por la ecuación 2.23:

$$\varphi\left(\mathbf{X}^s\right) = \frac{1}{\varepsilon} \sum_{j=1}^{M} g_j^2\left(\mathbf{X}^s\right) \tag{2.23}$$

En la estrategia de diseño tradicional modificada el proceso de optimización sin restricciones tiene lugar en el espacio $\mathbb{R}^N$, no es necesario resolver un sistema de ecuaciones en cada paso de iteración, pero la función objetivo en este caso es del tipo $F(\mathbf{X})$, que es definida como una función aditiva:

$$F\left(\mathbf{X}\right) = C\left(\mathbf{X}\right) + \varphi(\mathbf{X}) \tag{2.24}$$

La estrategia tradicional modificada es una estrategia de diseño distinta a la tradicional, y produce otra trayectoria en el espacio $\mathbb{R}^N$. El proceso se puede observar en la figura 2.5.

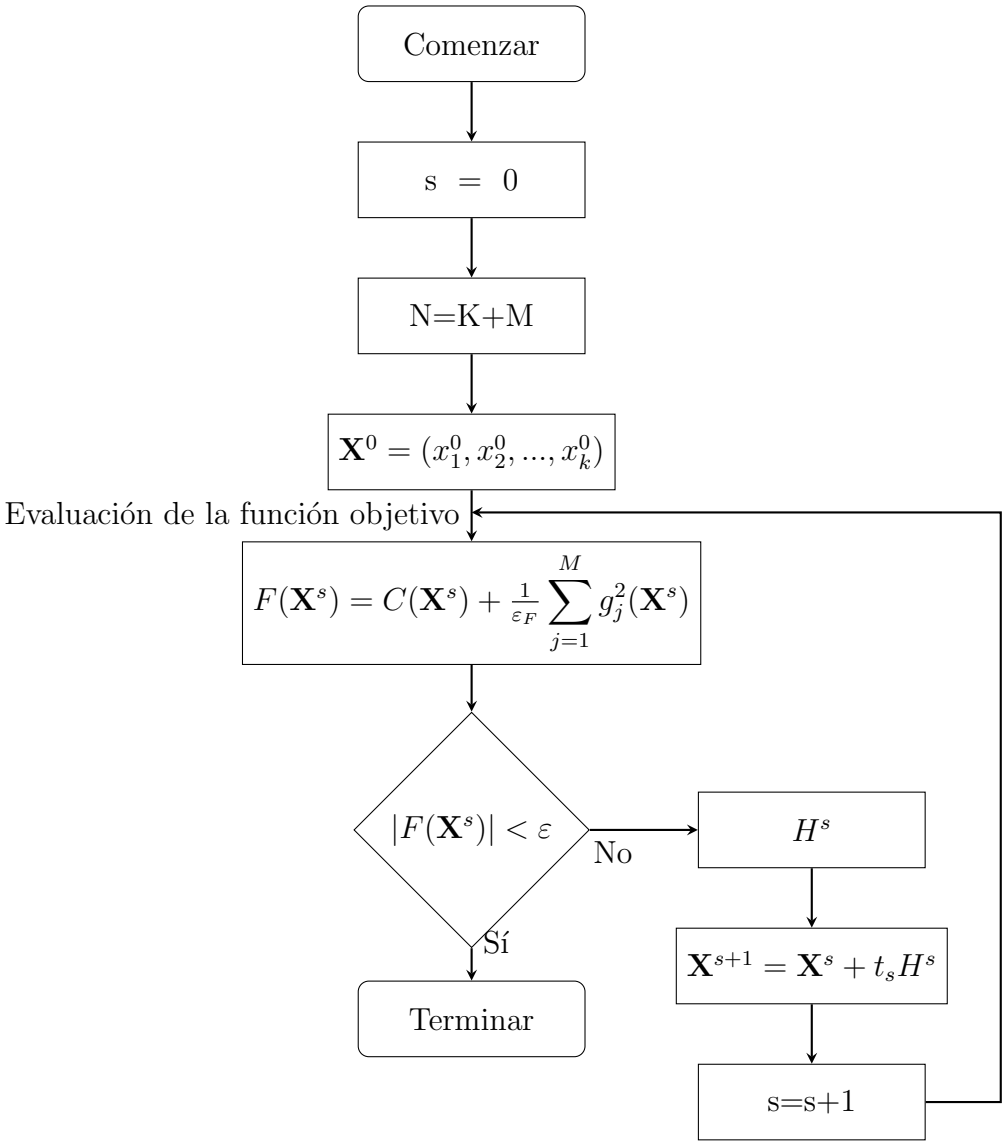


Figura 2.5: Estrategia de diseño tradicional modificada

2.5.2. La Metodología General de Diseño

Se puede generalizar la idea de agregar funciones de compensación para una parte del sistema de ecuaciones 2.1 y dejar la otra parte del sistema definida como un conjunto de restricciones (Guzmán, 2000) de tal forma que ahora la función de compensación incluya Z elementos:

$$\varphi\left(\mathbf{X}^s\right) = \frac{1}{\varepsilon}\sum_{i=1}^{Z} g_i^2(\mathbf{X}^s) \tag{2.25}$$

donde $Z \in [0, M]$, y el número de ecuaciones del sistema ahora está dado por $M - Z$:

$$g_j\left(\mathbf{X}\right) = 0, \quad j = Z+1, Z+2, \ldots, \ M \tag{2.26}$$

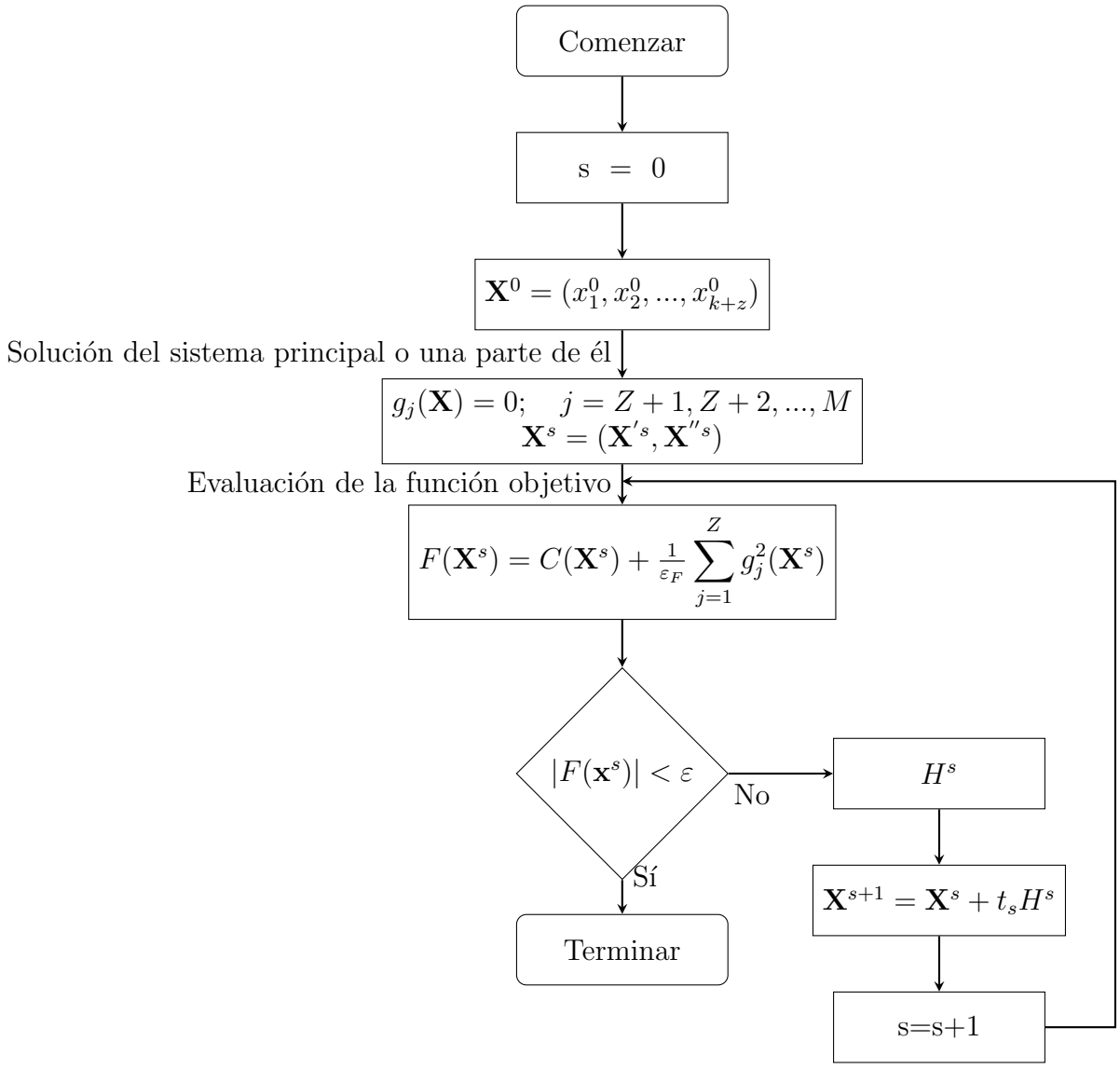


Figura 2.6: Diagrama de flujo de la metodología general

Es posible observar que la metodología general incluye a la estrategia tradicional de diseño (ETD) y la estrategia tradicional modificada de diseño (ETMD). Nótese que la metodología general produce 2^M estrategias de diseño, donde Z varía en el intervalo cerrado $[0, M]$. Para cada estrategia producida se tendrá una distinta trayectoria, donde la primera corresponderá a la estrategia tradicional y la última con la tradicional modificada. Todas las estrategias se producen dentro de un mismo procedimiento de optimización que se genera en el espacio $\mathbb{R}^{K+Z}$. El diagrama de flujo de la figura 2.6 ilustra el proceso completo. Cabe señalar que, el número de variables dependientes M aumenta conforme la complejidad del sistema se incrementa, eso ocasiona que la cantidad de estrategias de diseño crezca exponencialmente (Zemliak, 2001).

2.5.3. Sistematización de la MGD mediante el control óptimo

2.5.3.1. Control en tiempo mínimo

La optimización del proceso de diseño puede obtenerse formulando el problema de diseño como un problema de control óptimo. Es posible definir la metodología general de diseño usando las ecuaciones 2.2 y 2.26 con un valor variable para el parámetro Z durante el proceso. Es posible cambiar el número de variables independientes y el número de términos de la función de compensación en cualquier punto del proceso de optimización. Lo más conveniente, es introducir un vector de funciones especiales $U = (u_1, u_2, \ldots, u_m)$, donde $u_j \in \Omega;\ \Omega = 0; 1$. Estas funciones representan a las funciones de control del proceso de diseño y generalizan la estrategia de diseño. Cuando $u_j = 0$, la ecuación número j está presente en la ecuación 2.26 y el término $g_j^2(\mathbf{X})$ es eliminado de la función de compensación (ecuación 2.23); y cuando $u_j = 1$ ocurre lo opuesto (Zemliak, 2001, 2002). El modelo del sistema toma la forma:

$$(1 - u_j)g_j(\mathbf{X}) = 0; \quad j = 1, 2, ..., m \tag{2.27}$$

Y la función de compensación:

$$\varphi(\mathbf{X}) = \frac{1}{\varepsilon}\sum_{j=0}^{M} u_j \bullet g_j^2(\mathbf{X}) \tag{2.28}$$

El vector H que representa la dirección del movimiento, es función de $\mathbf{X}$ y de $U : H = F(\mathbf{X}, U)$. Las variables de control u_j modifican sus valores en función del punto actual del proceso de optimización. El número de trayectorias de diseño producidas es infinita y sólo una de ellas, o unas cuantas, cumplirán con los requisitos del diseño en tiempo mínimo de cómputo. El problema de la búsqueda de la estrategia de diseño óptimo es formulado como un problema de tiempo mínimo, propio de la teoría de control.

La idea de la formulación del problema de diseño como un problema en tiempo mínimo de la teoría de control no depende del algoritmo de optimización empleado (Zemliak, 2001). Para este trabajo se seleccionó el método de optimización del gradiente, por ser un método simple de orden 1, que casi siempre converge, además de arrojar buenos resultados para los ejemplos tratados en esta investigación.

2.5.3.2. Proceso de diseño en forma continua

El proceso de diseño con la formulación de la teoría de control óptimo puede prepararse en forma continua sustituyendo la ecuación para pasos discretos 2.2 por la ecuación diferencial:

$$\frac{d\mathbf{X}}{dt} = f(\mathbf{X}, U) \tag{2.29}$$

Donde la función $f(\mathbf{X}, U)$ representa al vector de la dirección del movimiento H y depende directamente de la función objetivo generalizada $F(\mathbf{X}, U)$, lo cual significa que el problema principal del proceso de diseño se puede formular como el problema de la integración de la ecuación 2.29 con las condiciones dadas por la ecuación 2.27. La estructura de la función H depende del algoritmo de optimización usado. En tales condiciones el problema de la búsqueda del proceso de diseño en tiempo óptimo se formula como un problema típico en tiempo óptimo de la teoría de control para la ecuación diferencial 2.29, en donde el término de la derecha depende del algoritmo de optimización aplicado a la función objetivo (ecuación 2.24) y en la función de compensación (ecuación 2.28) y de las condiciones adicionales dadas por la ecuación 2.27. En este contexto, el propósito del control óptimo es llevar a la función vectorial $f(\mathbf{X}, U)$ a cero para minimizar el tiempo total de cómputo.

2.5.3.3. MGD y el método del gradiente (aplicación del método del gradiente a la MGD)

La estructura de la función H para el método del gradiente está dada por:

$$H \equiv f\left(F\left(\mathbf{X}, U\right)\right) = -F'(\mathbf{X}, U) \tag{2.30}$$

Las siguientes ecuaciones determinan la forma discreta de éste método para cada componente del vector $\mathbf{X}$:

$$x_i^{s+1} = x_i^s + t_s f_i\left(\mathbf{X}, U\right); \quad i = 1, 2, \ldots, N \tag{2.31}$$

Y

$$\left(1 - u_j\right) g_j\left(\mathbf{X}\right) = 0 \;\; ; \quad j = 1, 2, \ldots, M \tag{2.32}$$

Donde los componentes $f_i(\mathbf{X}, U)$ están dados por las expresiones:

$$f_i\left(\mathbf{X}, U\right) = -\frac{\delta}{\delta x_i} F\left(\mathbf{X}, U\right); \quad i = 1, 2, \ldots, K \tag{2.33}$$

Y

$$f_i\left(\mathbf{X}, U\right) = -u_{i-K}\frac{\delta}{\delta x_i} F\left(\mathbf{X}, U\right) + \frac{\left(1 - u_{i-K}\right)}{t_s}\left\{-x_i^s + \eta_i\left(\mathbf{X}\right)\right\}, \quad i = K+1, K+2, \ldots, N \tag{2.34}$$

Donde:

$$F\left(\mathbf{X}, U\right) = C\left(\mathbf{X}\right) + \frac{1}{\varepsilon}\sum_{j=1}^{M} u_j g_j^2(\mathbf{X}) \tag{2.35}$$

$\eta_i(\mathbf{X})$ es la función implícita ($x_i^{s+1} = \eta_i(\mathbf{X})$) determinada por la ecuación 2.32, y el operador $\frac{\delta}{\delta x_i}$ tiene el siguiente significado:

$$\frac{\delta}{\delta x_i}\omega(\mathbf{X}) = \frac{\partial \omega(\mathbf{X})}{\partial x_i} + \sum_{p=K+1}^{K+M} \frac{\partial \omega(\mathbf{X})}{\partial x_p}\frac{\partial x_p}{\partial x_i} \tag{2.36}$$

Las variables de control u_j dependen, en general, del número de paso s. La ecuación con subíndice j es eliminada de la ecuación 2.32 y la variable dependiente x_{k+j} se transforma a independiente cuando $u_j = 1$. Este parámetro independiente queda definido por las ecuaciones 2.33 y 2.34 cuando el parámetro x_{K+j} es independiente. Por otra parte, cuando $u_j = 0$ la ecuación 2.33 con el miembro de la derecha dado por la ecuación 2.34 se transforma en la identidad $x_i^{s+1} = x_i^{s+1}$, por lo tanto en este paso del proceso de optimización x_i es dependiente y su valor corriente debe obtenerse directamente de la ecuación 2.32.

2.5.3.4. MGD y el método de Newton-Raphson

El método de Newton-Raphson se utiliza en los programas desarrollados en este trabajo para resolver el sistema de ecuaciones que describen al circuito electrónico en cuestión, para calcular las nuevos valores de las variables dependientes. Dependiendo de la estrategia de diseño seleccionada dentro de la metodologia general de diseño, la estructura del sistema de ecuaciones cambiará, teniéndose un número diferente de ecuaciones e incógnitas dependiendo de la estrategia utilizada durante el paso de iteración actual.

Durante la aplicación de este método (descrito previamente en la sección "Solución del sistema no lineal") se requiere el cálculo de la matriz jacobiana que puede obtenerse a través de derivadas numéricas. Las derivadas a utilizarse son seleccionadas en función de los valores del vector de control de la MGD, matemáticamente se puede ver esto en la ecuación 2.37.

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(1-u_1) & \frac{\partial f_1}{\partial x_2}(1-u_2)(1-u_1) & \cdots & \frac{\partial f_1}{\partial x_n}(1-u_n)(1-u_1) \\ \frac{\partial f_2}{\partial x_1}(1-u_1)(1-u_2) & \frac{\partial f_2}{\partial x_2}(1-u_2) & \cdots & \frac{\partial f_2}{\partial x_n}(1-u_n)(1-u_2) \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1}(1-u_1)(1-u_n) & \frac{\partial f_n}{\partial x_2}(1-u_2)(1-u_n) & \cdots & \frac{\partial f_n}{\partial x_n}(1-u_n) \end{bmatrix} \tag{2.37}$$

Capítulo 3

Aspectos de implementación

3.1. Plataforma de implementación

El código del programa realizado se escribió en Dev-C++. El código completo se encuentra en el Apéndice E de este archivo.

Para medir el tiempo en el que se genera la información de cada trayectoria de los retratos de fase se utiliza la función *clock* (). Entonces, primero se mide la hora de ejecución $T = clock\,()$ y al finalizar el método iterativo se utiliza $T = clock\,() - T$ para medir el tiempo de ejecución. En el código del programa desarrollado esto se implementa así:

```
191    T=clock();
192
193    for (it = 0; it <= 100000; it=it+1)
194    {
195        if(pco<=it && conpco==1) u=0;
196        EGD();
197    }
198
199    tiempo=clock()-T;
```

Figura 3.1: Implementación de la función *clock*()

La información obtenida por el programa desarrollado en lenguaje C++ se vincula con MATLAB para generar las gráficas requeridas. Para ello se agregó la librería $\#include$ "$engine.h$" al comienzo del código. También es necesario modificar las opciones del proyecto de DEV-C++, tales como los directorios y las direcciones de las librerías libeng.lib y libmx.lib (ver figura 3.2).

Además, se agregó la línea de código siguiente para abrir la ventana de comandos de MATLAB: $Engine * ep = engOpen\,(NULL)$. También, dentro del código se utilizaron las funciones siguientes:

$engEvalString(\ )$ Necesaria para escribir líneas de código en lenguaje MATLAB.

$engPutVariable(\ )$ Para poner variables dentro del espacio de trabajo de MATLAB.

$mxCreateDoubleMatrix\,(\ )$ Se utiliza para crear matrices tipo mxArray.

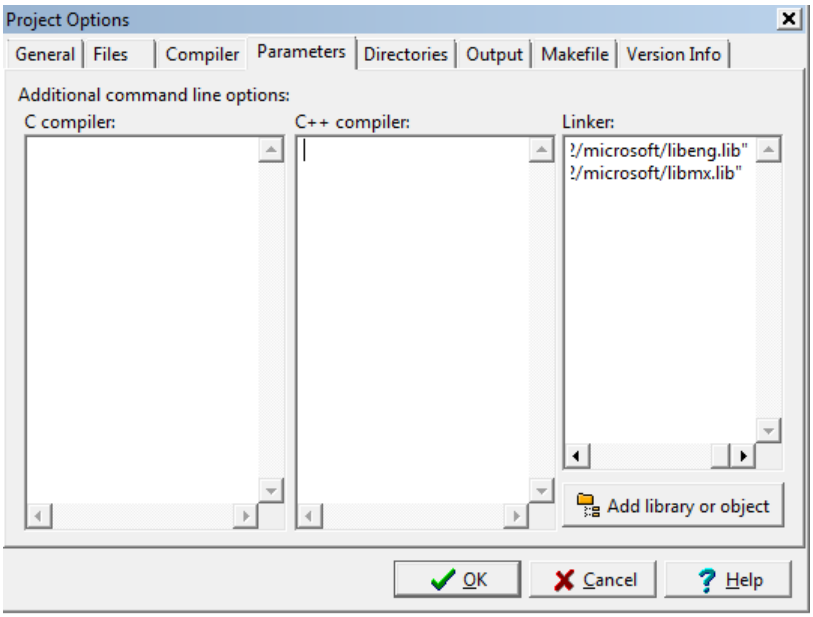


Figura 3.2: Ventana de opciones del proyecto en DEV-C++

3.2. El tiempo de ejecución

El tiempo de ejecución del proceso de diseño de sistemas electrónicos, es el tiempo requerido por el CPU para encontrar la solución de diseño. Para medir ese tiempo se ha utilizado en el código realizado en $C++$ el archivo de cabecera $\#include < time.h >$, el cual incluye las función $clock(\ )$ que regresa el tiempo consumido por el programa. Para calcular el tiempo de procesamiento únicamente de la parte del programa correspondiente a la MGD , se llamó a la función $clock()$ justo antes de comenzar la MGD y se guarda el valor devuelto, luego, al final de esa parte del código se tiene otra llamada a la misma función y se comparan los dos valores (cplusplus.com, 2020), obteniéndose el tiempo buscado. Otra manera de medir el tiempo fue multiplicando el tiempo de un paso por el número de pasos de tiempo utilizados.

Sin embargo, esta función no midió tiempos menores a 1 ms, lo cual es un problema debido a que el paso de tiempo de las estrategias de diseño solió ser de menor tamaño. Para solucionar esto, se optó por procesar cada programa una cierta cantidad de iteraciones IT, medir el tiempo transcurrido (T) y obtener el tiempo de un paso t_paso a través de la ecuación 3.1.

$$t_{paso} = \frac{T}{IT} \tag{3.1}$$

Capítulo 4

Efecto de aceleración del proceso de diseño

4.1. Efecto de aceleración en circuitos pasivo de un nodo

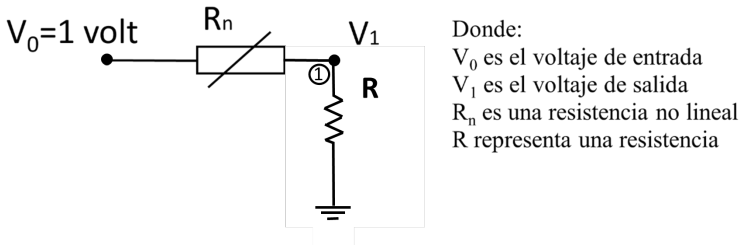


Figura 4.1: Diagrama del circuito de 1 nodo

El diagrama del circuito (figura 4.1) está formado por una resistencia no lineal y una resistencia lineal. Se ha fijado un voltaje de entrada $V_0 = 1\ volt$, y la salida del circuito es el voltaje V_1.

4.1.1. Ecuaciones que describen al circuito

Las ecuaciones que describen matemáticamente al sistema son las siguientes:

$$R_n = a + b(V_1 - V_0)^2 \tag{4.1}$$

$$g\left(R,\ V_1\right) \equiv \left(R_n + R\right) V_1 - RV_0 = 0 \tag{4.2}$$

$$\Longleftrightarrow g\left(R,\ V_1\right) \equiv \left(a + b\left(V_1 - V_0\right)^2 + R\right) V_1 - RV_0 = 0 \tag{4.3}$$

Donde

$$g\left(R,\ V_1\right) = 0 \tag{4.4}$$

cuando las variables $R,\ V_1$ satisfacen al sistema de una ecuación. Las variables del sistema se pueden reescribir en relación con las variables de estado x_1 y x_2 como:

$$R = {x_1}^2 \tag{4.5}$$

$$V_1 = x_2 \tag{4.6}$$

con $x_1,\ x_2\ in\mathbb{R} \Longrightarrow R \in \mathbb{R}_+ \cup 0$. Por limitaciones físicas, la resistencia sólo puede tener valores positivos, es por eso por lo que se toma $R = {x_1}^2$, de este modo se asegura que para cualquier valor real de la variable de estado x_1, la resistencia R será siempre mayor o igual que cero.

El vector de estados queda como sigue:

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{4.7}$$

con x_1 como variable independiente (pues está relacionada con la resistecia R) y x_2 dependiente (de los paramétros del sistema y de la resistencia R), aunque más adelante se verá que la propiedad de ser independiente o dependiente cambiará de acuerdo con la estrategia elegida para resolver el sistema.

Entonces la ecuación $g\left(R,\ V_1\right)$ queda como:

$$g\left(x_1,\ x_2\right) = \left(a + b\left(x_2 - V_0\right)^2 + {x_1}^2\right) x_2 - {x_1}^2 V_0 = 0 \tag{4.8}$$

Los parámetros de la ecuación son los siguientes: $a = 1$, $b = 0.5$, $V_0 = 1\ volt$. Por lo que

$$g\left(x_1,\ x_2\right) = \left(1 + 0.5\left(x_2 - 1\right)^2 + {x_1}^2\right) x_2 - {x_1}^2 = 0 \tag{4.9}$$

El valor deseado para la variable de salida x_2 es $V_d = 0.2\ volts$. Si $g\left(x_1,\ x_2\right) = 0$, entonces se habrá alcanzado $V_1 = V_d$.

4.1.2. ETD

Para resolver el sistema por medio de la ETD (Estrategia Tradicional de Diseño), $K = 1$, $M = 1$ y se define la función objetivo como:

$$C(\mathbf{X}) = \left(x_2 - V_d\right)^2 \tag{4.10}$$

4.1.3. ETDM

En caso de utilizar la ETDM (Estrategia Tradicional de Diseño Modificada), se considera que todas las variables son independientes (o sea K=2, M=0), que no dependen del sistema, por lo que se agrega una función de penalidad. Para este ejemplo se eligió:

$$\varphi\left(\mathbf{X}\right) = g^2(\mathbf{X}) \tag{4.11}$$

De este modo se tiene que la función objetivo es de la forma:

$$F\left(\mathbf{X}\right) = \ C\left(\mathbf{X}\right) + \varphi(\mathbf{X}) \tag{4.12}$$

$$\Longrightarrow F\left(\mathbf{X}\right) = \ \left(x_2 - V_d\right)^2 + g^2(\mathbf{X}) \tag{4.13}$$

4.1.4. MGD

Al usarse la MGD (Metodología General de Diseño), se consideran todas las combinaciones en las que $N = K + M = 2$. Esto es, $K = 1$ y $M = 1$, además del caso $K = 2$ y $M = 0$. Por lo que, para este ejemplo, se consideran las 2 estrategias anteriores (ETD y $ETDM$). Para cambiar de una a otra, dentro de la función objetivo se utiliza una variable artificial $u \in 0, 1$, y cuyo valor se selecciona antes de resolver el problema, véase la ecuación 4.14.

$$F(\mathbf{X}) = C(\mathbf{X}) + u\varphi(\mathbf{X}) \tag{4.14}$$

Para resolver el problema con la perspectiva de control de tiempo mínimo, se considera la ecuación anterior, pero la variable $u \in 0, 1$ se toma como una variable de control que va a cambiar a través del tiempo, con la finalidad de conmutar de una estrategia a otra en el paso de tiempo que uno desee o indicado por algún algoritmo, mientras se va resolviendo el sistema. Para este ejemplo, en este trabajo, se comienza con $u = 1$, por lo que se tiene que utilizar la ETDM hasta que se alcance un número de pasos especificado, que es cuando se cambia a $u = 0$, provocando que ahora se use la ETD.

4.1.5. Resultados

Los resultados que se obtuvieron son para la Metodología General de Diseño (MGD), que, con el vector de control u, contiene a la estrategia tradicional de diseño (ETD) cuando $u = 0$ y a la Estrategia Tradicional de Diseño Modificada (ETDM), si $u = 1$. Se presentan resultados para distintos valores de los parámetros del circuito que, con diferentes condiciones iniciales para algunas de las variables independientes, se generan familias de curvas distintas para cada estrategia. Luego se muestran algunas condiciones para que exista el efecto de aceleración.

4.1.5.1. Valores iniciales

Gráficas de ETD

Con los parámetros presentados en la tabla 4.1 y cambiando las condiciones iniciales de $x_2 = [-3, 3]$ se obtuvieron las gráficas 4.2 y 4.3.

Cuadro 4.1: Parámetros utilizados en el diseño del circuito de 1 nodo

$a = 1$	Constante "a" de resistencia variable R_n
$b = 0$	Constante "b" de resistencia variable R_n
$paso = 10^{-3}$	Parámetro de escala
$eps = 10^{-8}$	Parámetro de comparación del criterio de convergencia de la MGD
$power = 10^{-8}$	Parámetro de comparación del criterio de convergencia del MNR
$dx_1 = 10^{-5}$	Diferencial de x_1
$dx_2 = 10^{-5}$	Diferencial de x_2
V_d =0.2	Valor deseado para x_2
$V_0 = 1$	Voltaje de entrada

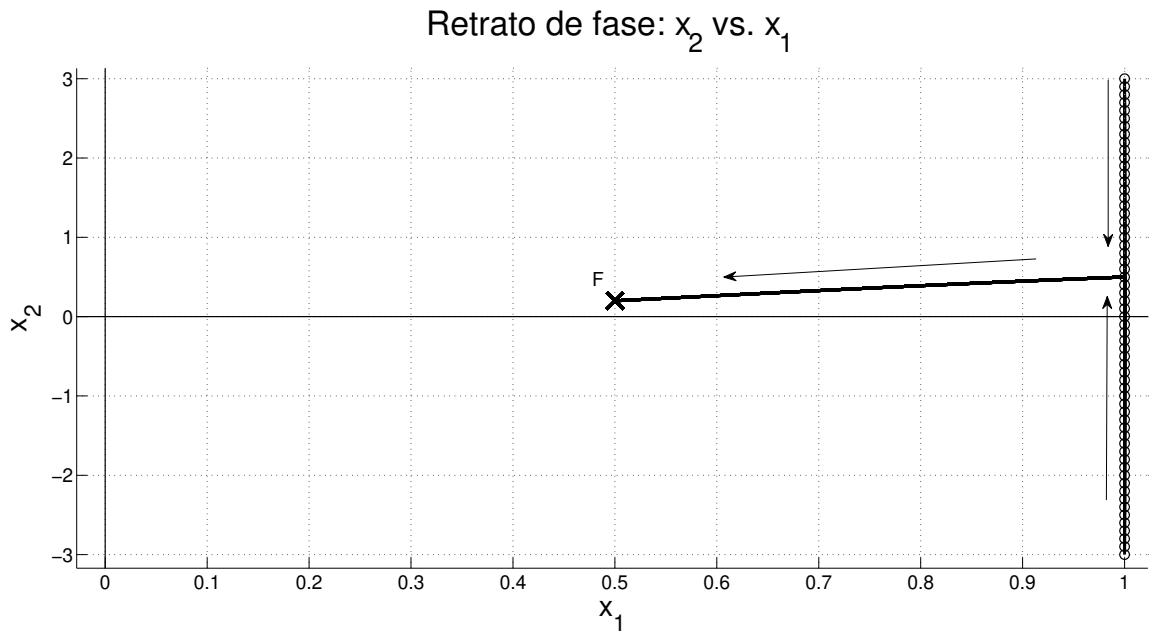


Figura 4.2: Retrato de fase con ETD

La figura 4.2 corresponde al retrato de fase x_2 *vs* x_1. Sin importar la condición inicial, las curvas hacen un salto en vertical, de tal forma que en adelante siguen la misma trayectoria. En la figura 4.3 se tiene la gráfica del número de pasos vs. condición inicial para x_2, en la cual se observa que sin importar el valor inicial de x_2, el número de pasos es el mismo para llegar a la solución.

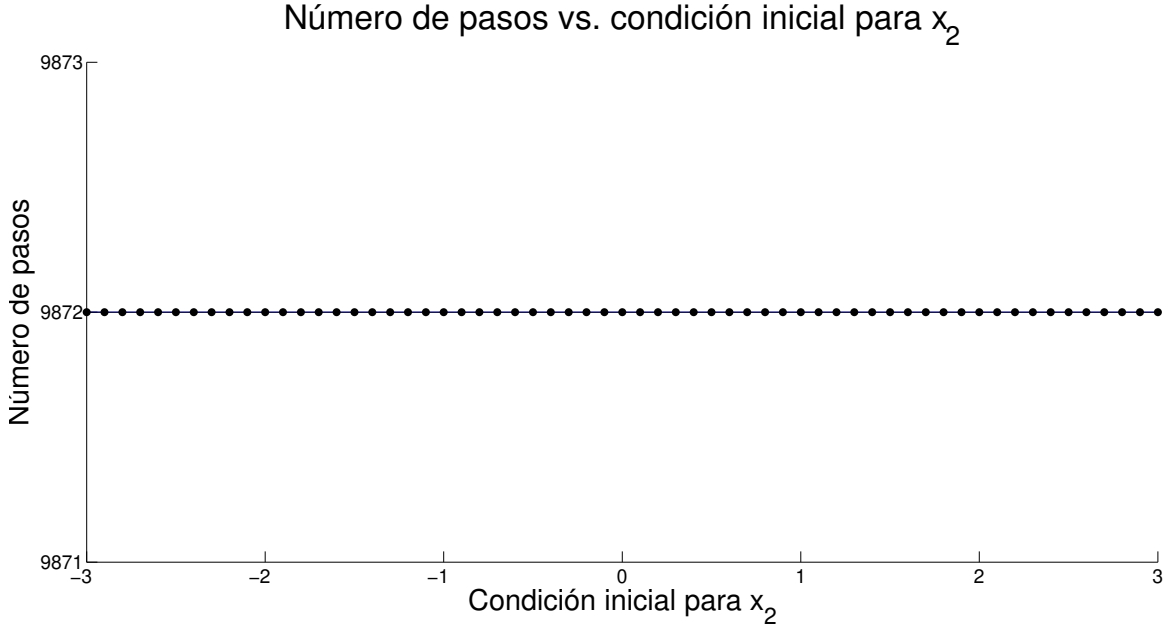


Figura 4.3: Número de pasos vs. condición inicial con ETD

Gráficas de ETDM

Utilizando los mismos parámetros usados para la ETD, se tiene un nuevo retrato de fase x_2 *vs.* x_1 (figura 4.4), en el que se observa que para cada condición inicial diferente corresponde otra curva. Con la gráfica de la figura 4.5 se ve que el número de pasos para alcanzar la solución es distinto, dependiendo de la condición inicial.

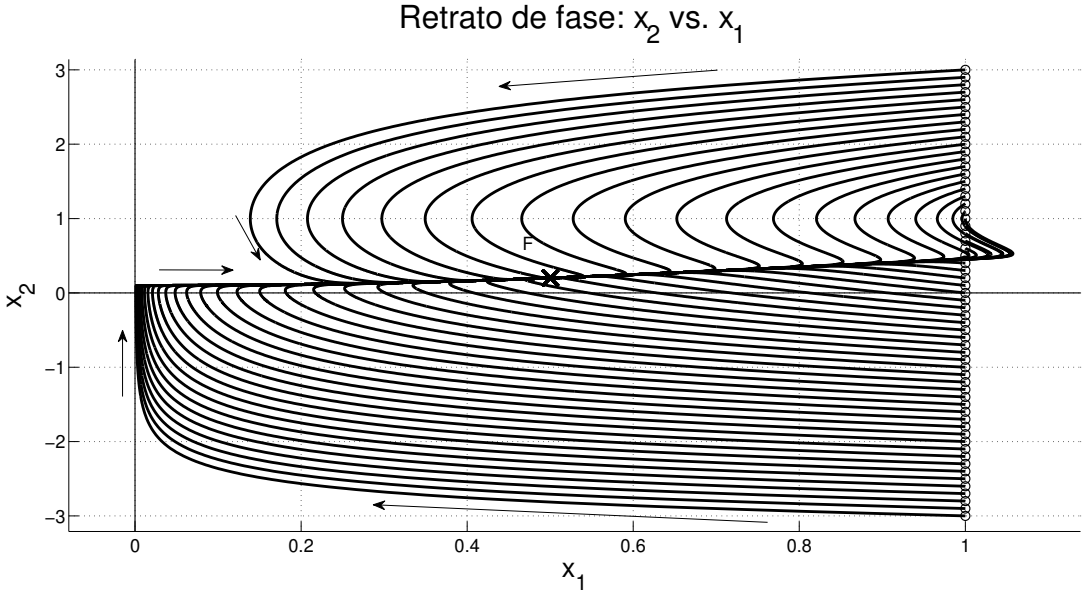


Figura 4.4: Retrato de fase con ETDM

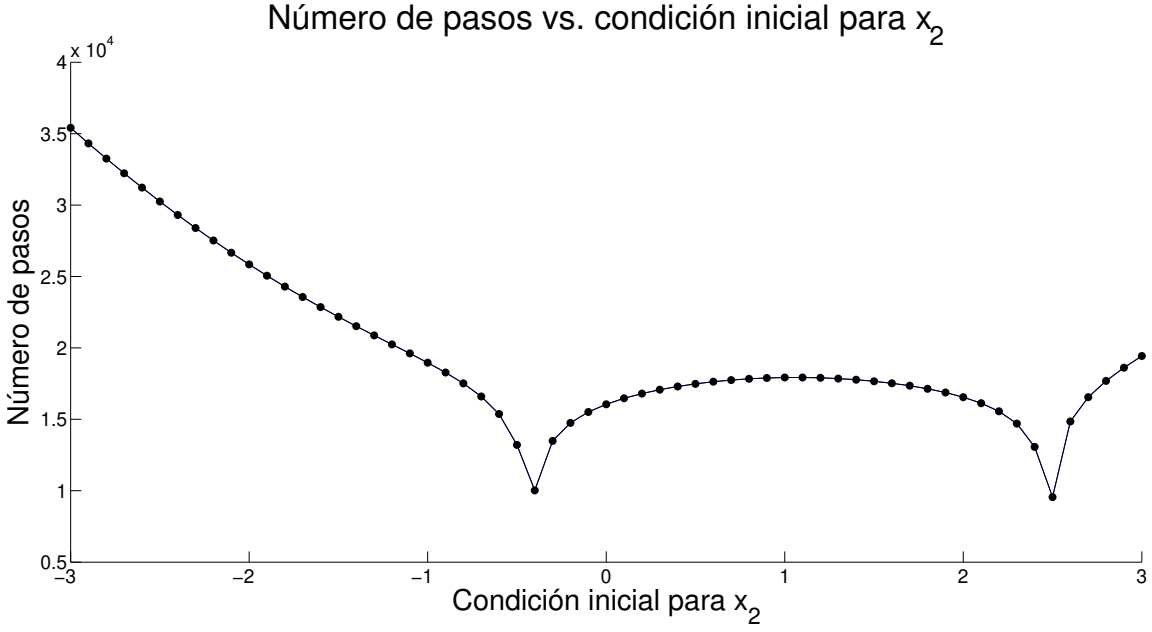


Figura 4.5: Número de pasos vs. condición inicial con ETDM

Cambiando únicamente uno de los parámetros de la resistencia no lineal de $b = 0$ a $b = 1$ (ver tabla 4.2), se genera otro retrato de fase mostrado en la figura 4.6 .

Cuadro 4.2: Segundo conjunto de parámetros utilizados en el diseño del circuito de 1 nodo

$a = 1$	Constante "a" de resistencia variable R_n
$b = 1$	Constante "b" de resistencia variable R_n
$paso = 10^{-3}$	Parámetro de escala
$eps = 10^{-8}$	Parámetro de comparación del criterio de convergencia de la MGD
$power = 10^{-8}$	Parámetro de comparación del criterio de convergencia del MNR
$dx_1 = 10^{-5}$	Diferencial de x_1
$dx_2 = 10^{-5}$	Diferencial de x_2
V_d =0.2	Valor deseado para x_2
$V_0 = 1$	Voltaje de entrada

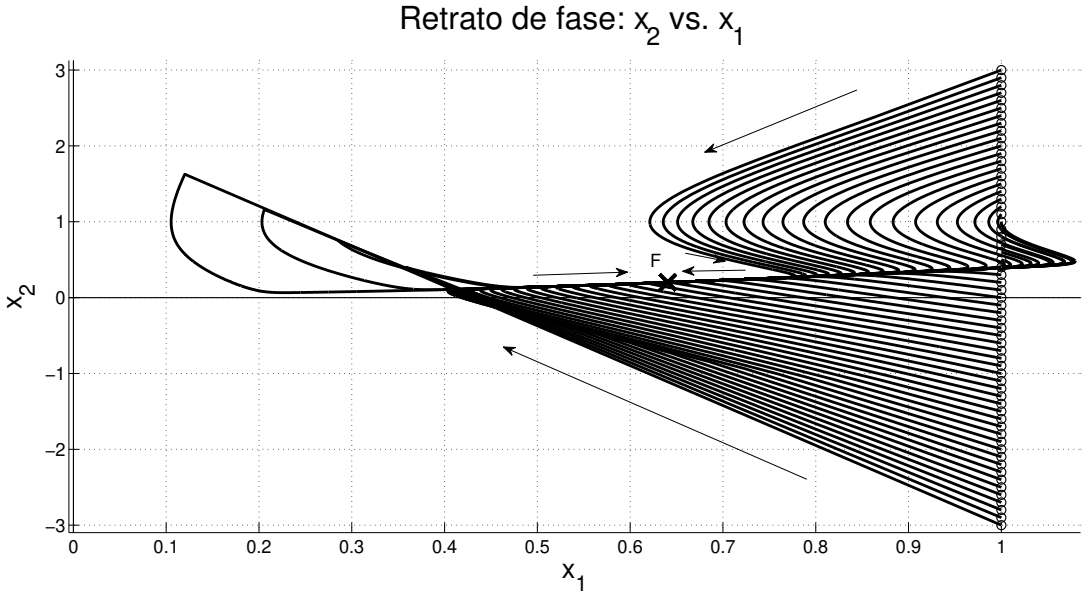


Figura 4.6: Retrato de fase con ETDM y b=1

El retrato de fase obtenido (ver figura 4.6) tiene curvas que sobresalen del comportamiento de la mayoría de las curvas de este retrato de fase, éstas corresponden a $x_2 = [-3, \ -2.3]$. En esa misma zona se formó una curva convexa en la figura 4.7. A pesar de llegar a la solución, esas curvas tienen cambios abruptos y se cruzan entre sí, por lo que para que sean adecuadas, debe modificarse el tamaño del parámetro de escala.

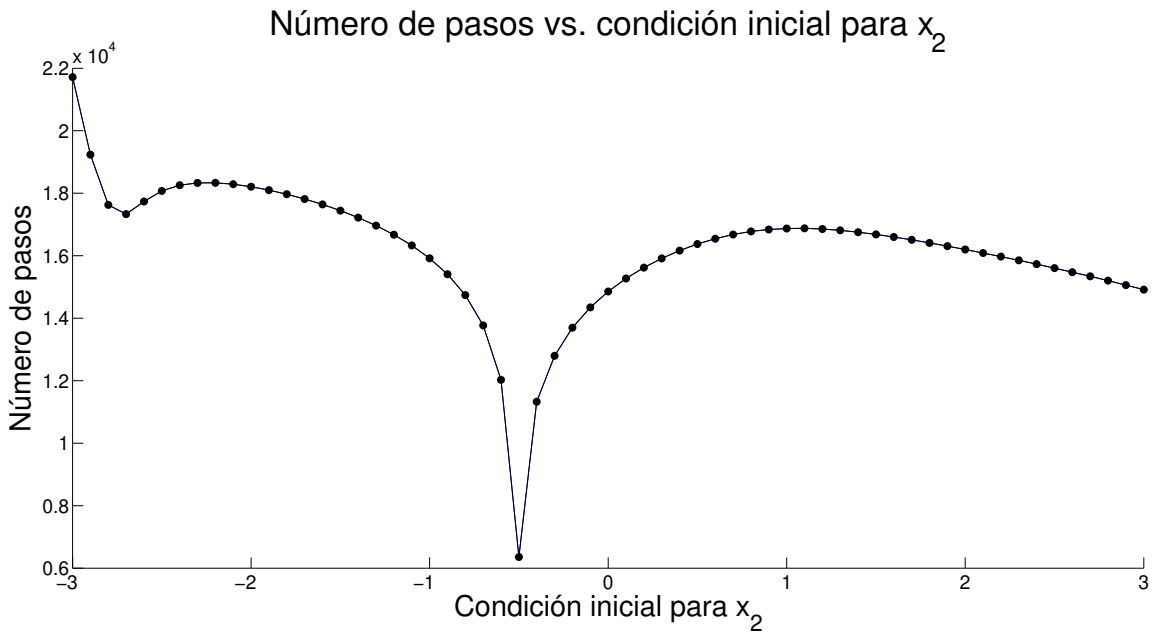


Figura 4.7: Número de pasos vs. condición inicial con ETDM y b=1

Para tal corrección, se cambia el paso de tiempo de 10^{-3} a 10^{-4} (ver tabla 4.3) y se obtiene un nuevo retrato de fase, mostrado en la figura 4.8. Con esos cambios, las curvas correspondientes a x_2 =-3 hasta x_2 =-2.3 tienen un comportamiento más suave y acorde al de todas las demás curvas. En la figura 4.9 se muestra la relación del número de pasos vs condición inicial para x_2.

Cuadro 4.3: Tercer conjunto de parámetros utilizados en el diseño del circuito de 1 nodo

$a = 1$	Constante "a" de resistencia variable R_n
$b = 1$	Constante "b" de resistencia variable R_n
$paso = 10^{-4}$	Parámetro de escala
$eps = 10^{-8}$	Parámetro de comparación del criterio de convergencia de la MGD
$power = 10^{-8}$	Parámetro de comparación del criterio de convergencia del MNR
$dx_1 = 10^{-5}$	Diferencial de x_1
$dx_2 = 10^{-5}$	Diferencial de x_2
V_d =0.2	Valor deseado para x_2
$V_0 = 1$	Voltaje de entrada

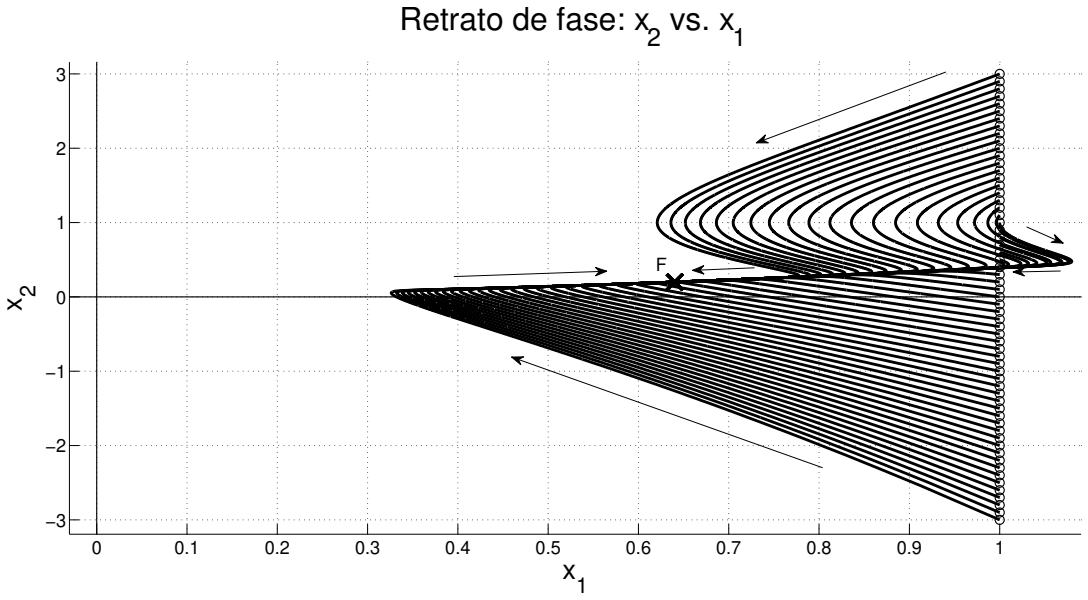


Figura 4.8: Retrato de fase con ETDM, b=1 y $paso = 10^{-4}$

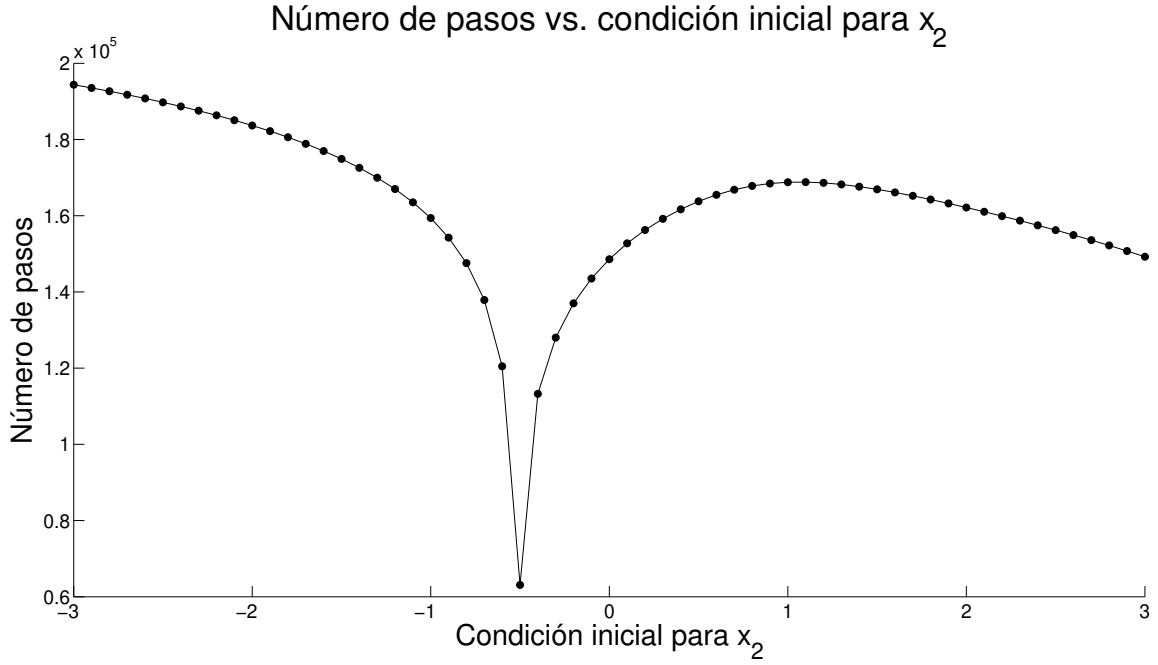


Figura 4.9: Número de pasos vs. condición inicial con ETDM, b=1 y $paso = 10^{-4}$

4.1.5.2. Análisis del efecto de aceleración

Se analizó el efecto de aceleración a través de conmutaciones entre las ETDM y la ETD. Se le indicó al programa una cantidad de iteraciones que debió realizar con la estrategia tradicional de diseño modificada, antes de cambiar a la estrategia tradicional. Al momento del cambio, la ETD toma como valor inicial al valor final generado por la ETDM. Entonces, a partir del valor de x_1 dado por la ETDM, se calcula el nuevo valor de x_2 con el MNR de la ETD; quedando un nuevo par ordenado formado por la anterior x_1 y el nuevo x_2. Gráficamente esto se ve como un salto, una línea en vertical que une el nuevo punto con el anterior. Al terminar de ejecutarse por primera vez la rutina de la ETD se generan nuevos valores de x_1 y x_2, y el proceso se repite, creándose la trayectoria correspondiente a la ETD, hasta alcanzar la solución de diseño.

Se cambió la cantidad de iteraciones permitidas en la ETDM, manteniendo el mismo conjunto de parámetros y condiciones iniciales, de tal manera que se generaran trayectorias con diferente paso de conmutación y un total de pasos distinto en cada ejecución del programa. De esa manera, se buscó reducir ese número con la finalidad de encontrar el número de pasos o iteraciones adecuado de conmutación para que al momento de conmutar se llegara inmediatamente a la solución de diseño. Lo que se obtuvo fue un gran efecto de aceleración, pues se usaron 0 pasos de la ETDM, 1 paso de conmutación (correspondiente al uso del MNR de la ETD sin terminar ni una iteración de la estrategia tradicional) y 0 pasos completos de la ETD.

Cuadro 4.4: Conjunto de parámetros utilizado en el diseño del circuito de 1 nodo con $b = 0$

$a = 1$	Constante "a" de resistencia variable R_n
$b = 0$	Constante "b" de resistencia variable R_n
$paso = 10^{-4}$	Parámetro de escala
$eps = 10^{-8}$	Parámetro de comparación del criterio de convergencia de la MGD
$power = 10^{-8}$	Parámetro de comparación del criterio de convergencia del MNR
$dx_1 = 10^{-5}$	Diferencial de x_1
$dx_2 = 10^{-5}$	Diferencial de x_2
V_d =0.2	Valor deseado para x_2
$V_0 = 1$	Voltaje de entrada

Usando los valores de la tabla 4.4 y el vector de estados $\mathbf{X} = [0.5, x_2]$ se obtiene la gráfica de la figura 4.10. Teniendo como condición inicial $x_{2_0} = -3$, se comienza con la ETDM y se conmuta a la ETD en el paso que el usuario indique. De este modo lo que se busca es encontrar el número de pasos con el cual la conmutación permita llegar a la solución con el menor número de éstos. En este caso, el menor número de pasos ocurre cuando la conmutación a la ETD es realizada después 8 pasos de la ETDM. Al cambiar a la ETD se ha dado el efecto de aceleración.

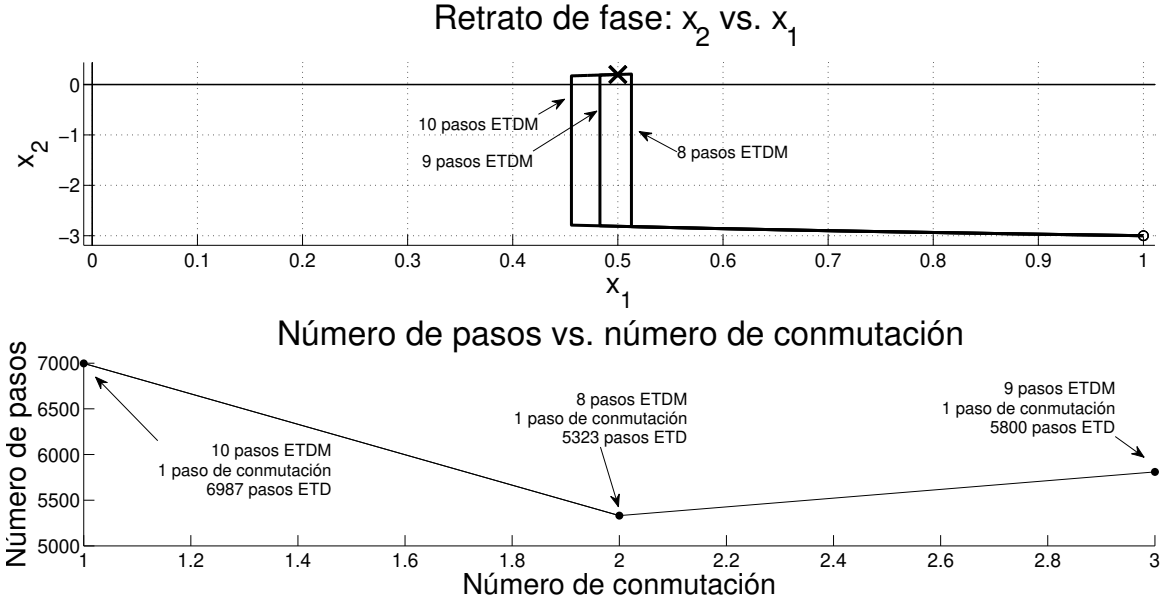


Figura 4.10: Conmutación ETDM a ETD usando los parámetros de la tabla 4.4

Con un acercamiento al punto final indicado con una cruz de la figura 4.10 es posible darse cuenta que en realidad no se tiene un único punto final, pues se observan 2 cruces en la figura 4.11. Incluso, con otro acercamiento como el de la figura 4.12 se puede ver que existen más puntos finales.

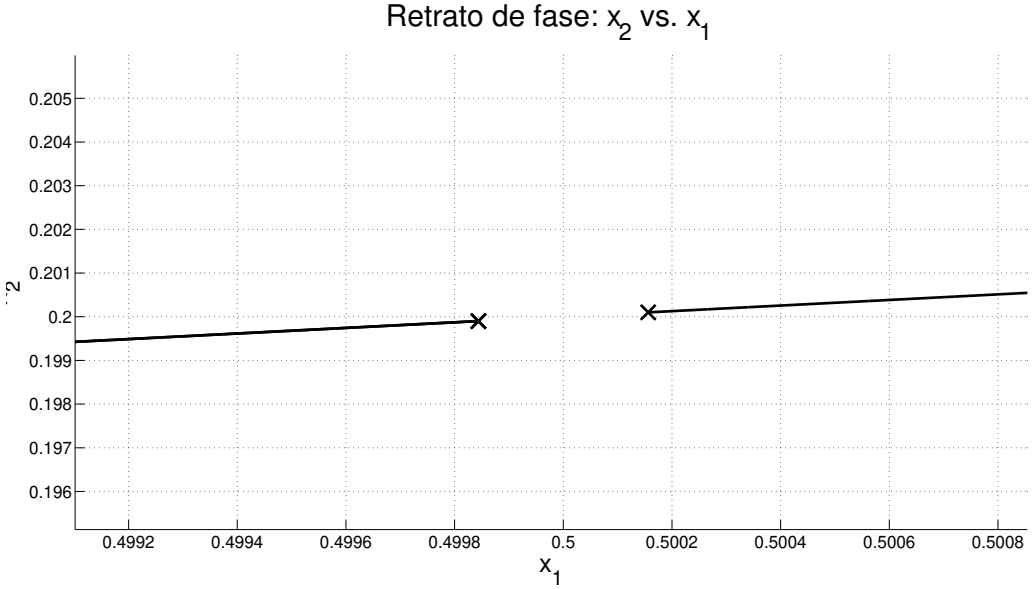


Figura 4.11: Acercamiento 1 de conmutación ETDM a ETD usando los parámetros de la tabla 4.1

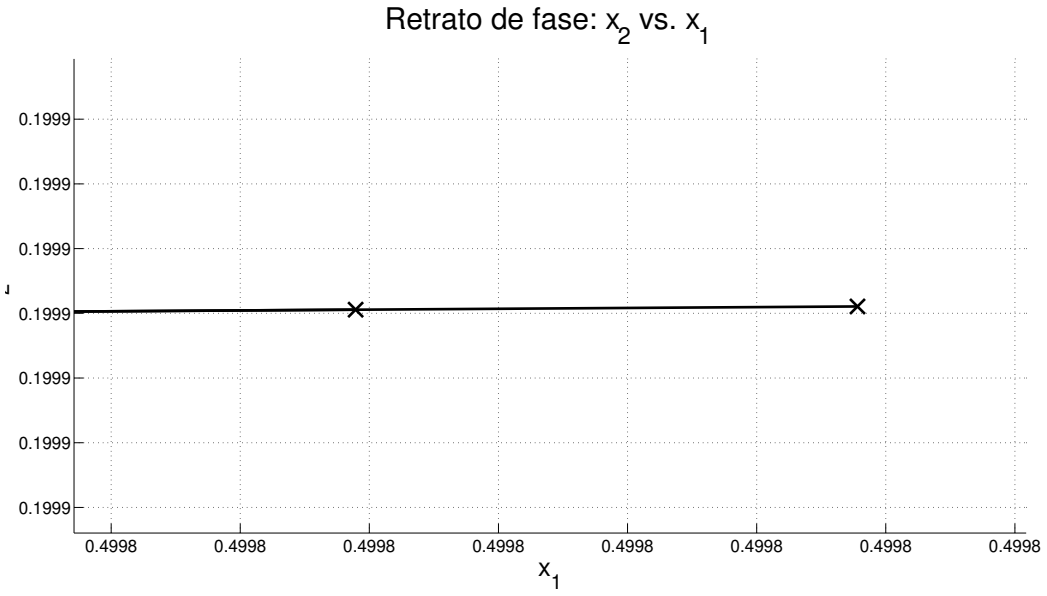


Figura 4.12: Acercamiento 2 de conmutación ETDM a ETD usando los parámetros de la tabla 4.1

Si el valor de épsilon es modificado de $\epsilon = 10^{-8}$ a $\epsilon = 10^{-4}$, se consigue la figura 4.13, en la que se obtiene un gran efecto de aceleración, al usarse únicamente 8 pasos de la ETDM y 0 de la ETD.

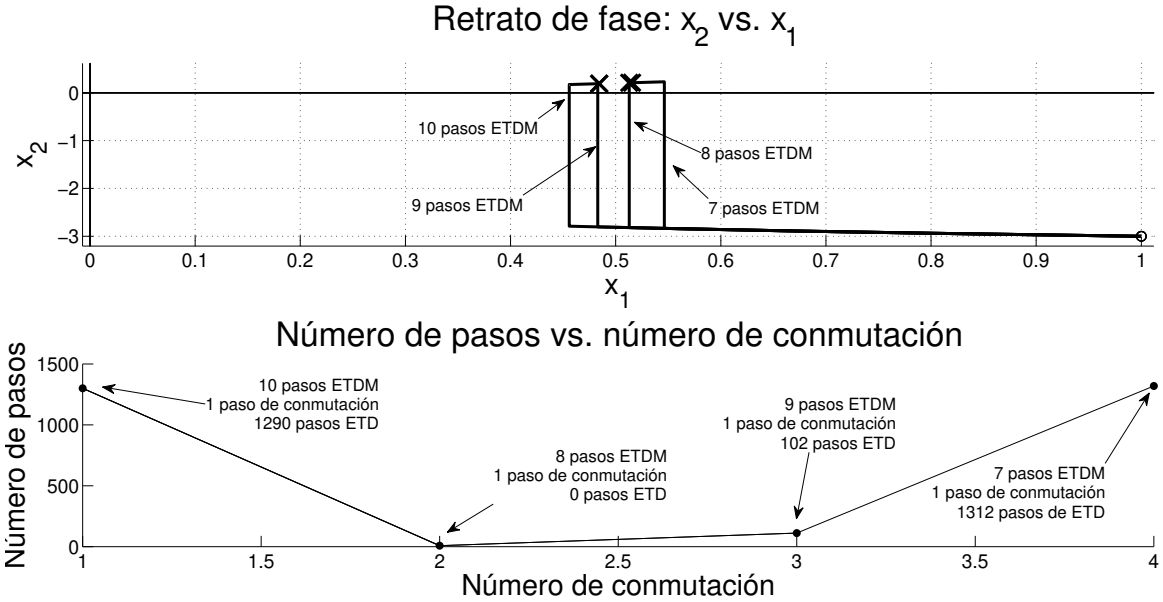


Figura 4.13: Conmutación ETDM a ETD usando los parámetros de la tabla 4.1 pero poniendo a $\epsilon = 10^{-4}$

En la figura 4.14 hay conmutaciones entre la ETDM y la ETD, usando los mismos paráme-

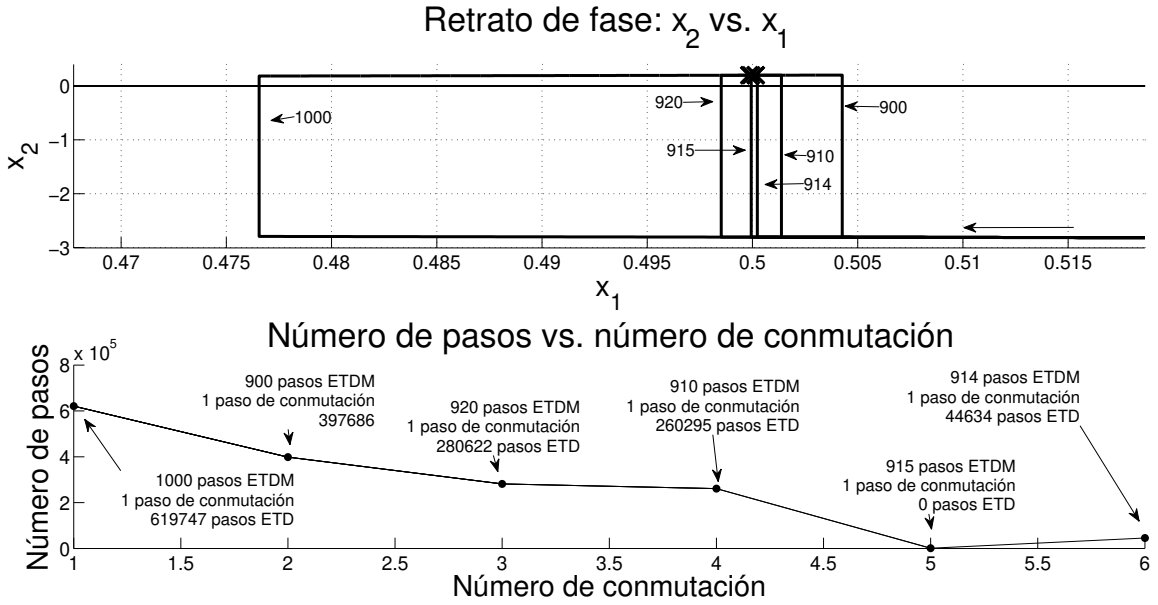


Figura 4.14: Conmutación ETDM a ETD usando los parámetros de la tabla 4.1 pero poniendo el $paso = 10^{-5}$

tros de la tabal 4.1 pero reduciendo el tamaño del parámetro de escala a 10^{-5}. Obsérvese que conmutando después del $paso = 915$ se llega al punto final señalado con una cruz, por lo que se alcanza la solución de diseño del sistema.

4.1.5.3. La separatriz: condiciones necesarias y suficientes para el efecto de aceleración

Los retratos de fase utilizados en esta sección, muestran en color negro las curvas que tienen posibilidad de tener el efecto de aceleración y en color gris las que no pueden tener ese efecto (ver figura 4.15). La curva gruesa en color negro es la curva separatriz que divide las zonas con efecto de aceleración y sin éste. La línea discontínua en vertical es la ordenada x_2 que atraviesa el punto final de la solución, F, marcado con una cruz.

El retrato de fase de la figura 4.15 se generó usando la tabla 4.4 y el vector de estados $\mathbf{X} = [0.5, x_2]$ con $x_2 = -3, -2.9, ..., 3$. Las curvas con posibilidad del efecto de aceleración deseado, son al menos las correspondientes a la condición inicial $x_2 \in \{-3, -2.9, ..., -0.5\} \cup \{2.3, 2.4, ..., 3\}$. Mientras que las trayectorias con la condición inicial $x_2 \in \{-0.6, -0.5, ..., 2.2\}$ son las que no pueden tener el efecto de aceleración que esperamos. Tómese en cuenta que no hay certeza de que las curvas no graficadas y que su condición inicial es $x_2 = (-0.5, 2.3)$ se encuentran o no en el conjunto de trayectorias con el efecto buscado. También, se debe de considerar que aunque el comportamiento de las curvas dentro de la región que se ha considerado con efecto de aceleración tiende a ser uniforme, no se puede asegurar que las trayectorias no mostradas dentro de esa región también cuenten con el efecto de aceleración.

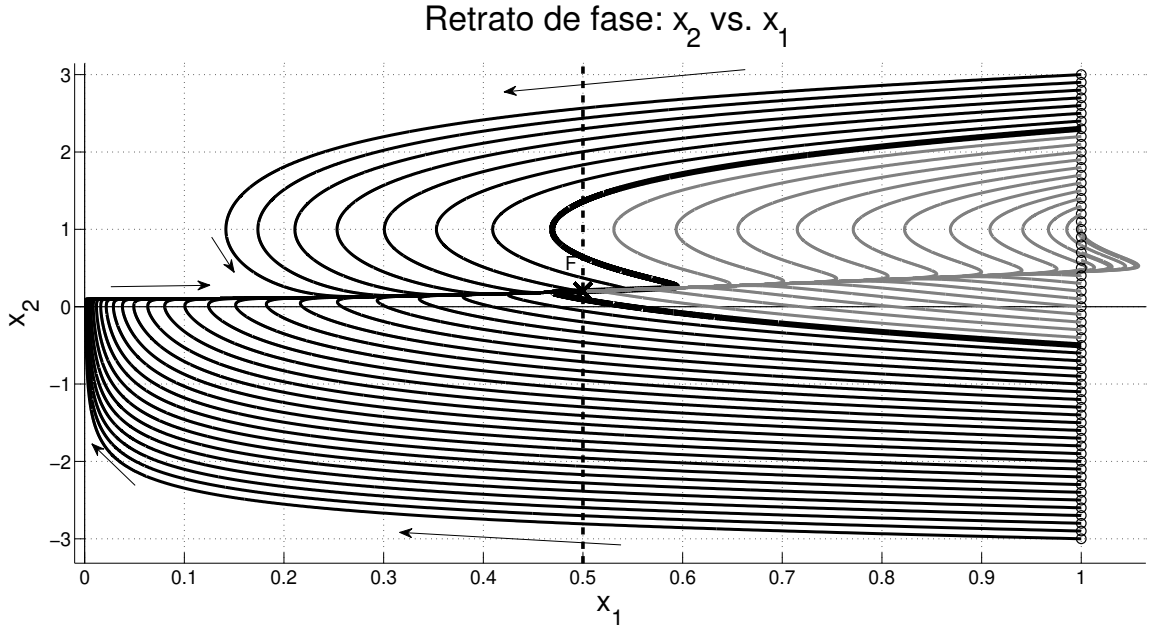


Figura 4.15: Retrato de fase con parámetros de la tabla 4.4 y su separatriz

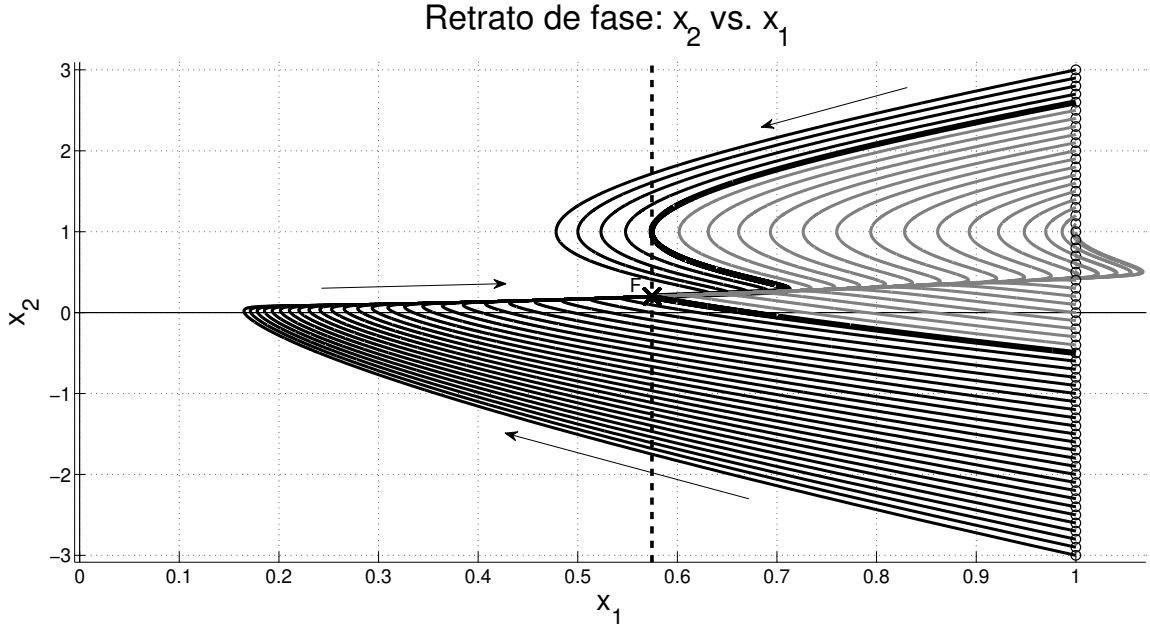


Figura 4.16: Retrato de fase con parámetros de la tabla 4.4, pero cambiando a($b = 0.5$)

Cambiando la constante $b = 0$ a $b = 0.5$ se obtiene el retrato de fase de la figura 4.16. Ahora, el vector de estado inicial desde donde es posible generar trayectorias con el efecto de aceleración que se busca es $\mathbf{X} = [0.5, x_2]$ con $x_2 \in \{-3, -2.9, ..., 0.5\} \cup \{2.6, 2.7, ..., 3\}$. Comparando con el retrato de fase de la figura 4.15 se obtuvieron menos trayectorias con el efecto de aceleración, pues la región con curvas sin tal efecto se hizo más ancha, alejando su

límite superior de $x_2 = 2.3$ a $x = 2.6$ (considerando que el cambio de x_2 es en 0.1, el límite superior pudiera ser $x_2 = 2.6 + \epsilon_L$, donde $\epsilon_L < 0.1$.

Ahora, se cambia el valor de b a $b = 1$, obteniéndose así la figura 4.17. De nuevo, el número de curvas con valores iniciales x_2 sin efecto de aceleración aumentó en 1, hasta llegar al menos a $x_2 = 2.8$.

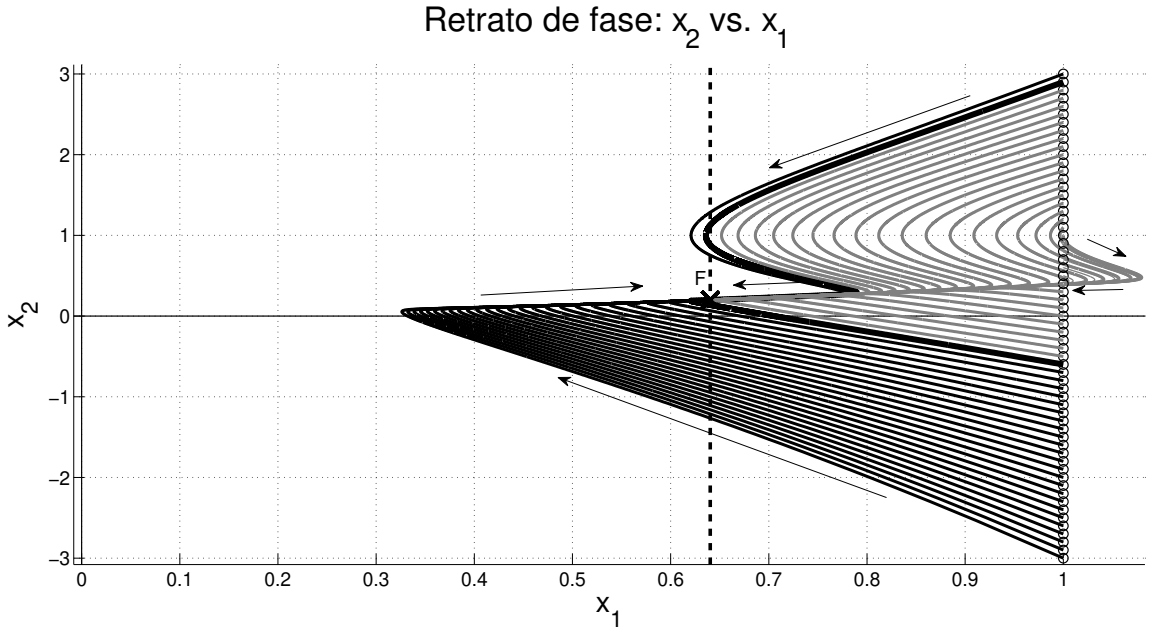


Figura 4.17: Retrato de fase con los parámetros de la tabla 4.4, pero cambiand a $b = 1.0$

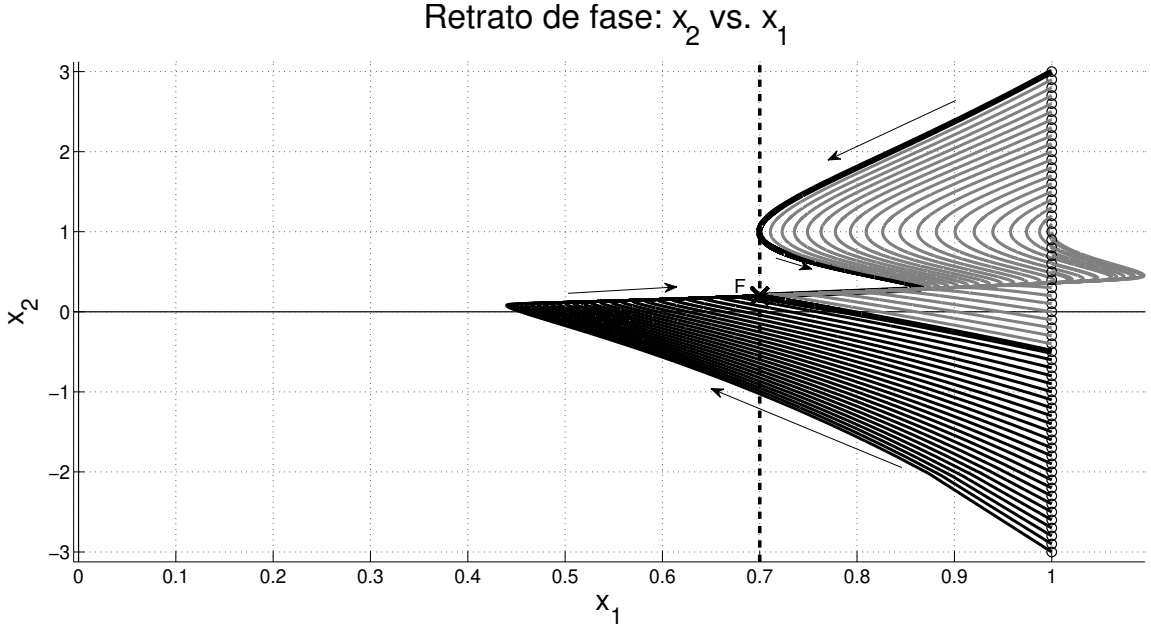
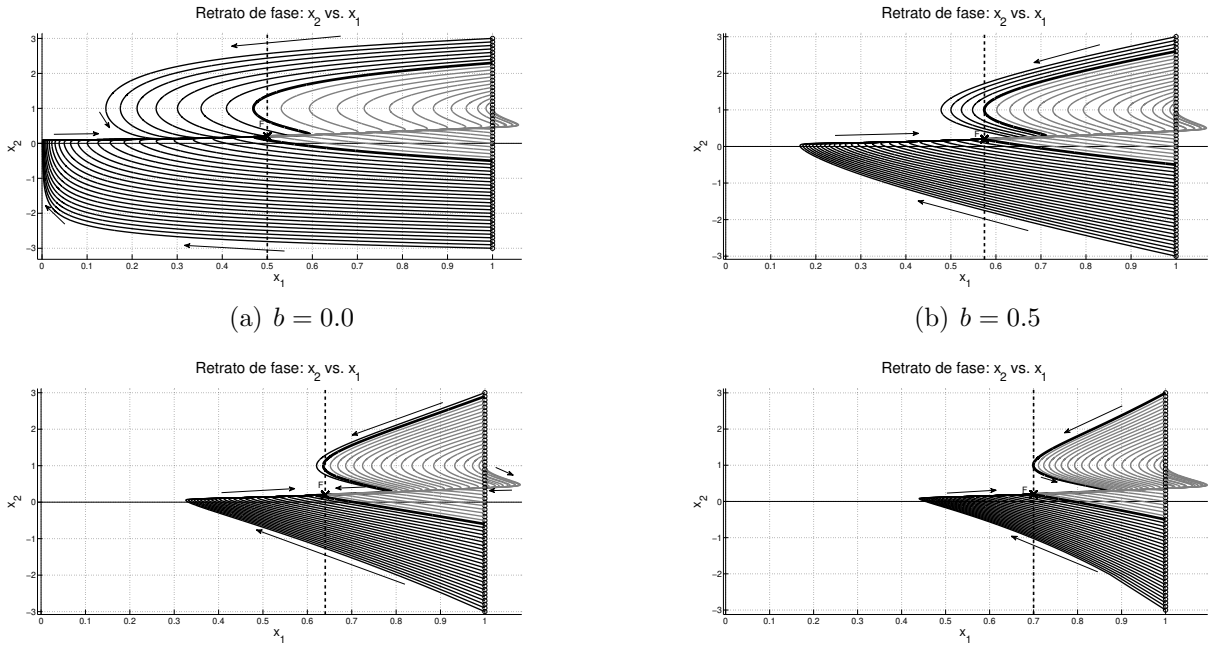


Figura 4.18: Retrato de fase con $b = 1.5$

En la figura 4.18 se presenta el retrato de fase donde se ha cambiado el valor b a b =1.5. Obsérvese que a comparación del retrato de fase de la figura 4.17, se agregó al menos una curva más a la zona de trayectorias sin efecto de aceleración, siendo ésta generada con la variable de estado inicial $x_2 = 2.9$.



(a) $b = 0.0$ (b) $b = 0.5$

(c) $b = 1.0$ (d) $b = 1.5$

Figura 4.19: Retratos de fase con diferentes valores de b

En la figura 4.19 se muestran las cuatro gráficas con los distintos valores del parámetro b utilizados: $b = 0,\ 0.5,\ 1,\ 1.5$, para poderlas comparar visualmente. Obsérvese que el retrato de fase se vuelve más estrecho conforme b crece. También, todas las curvas se alejaron del ordenada de origen. La familia de curvas que posibilitó un mayor número de trayectorias con el efecto de aceleración, usando los mismos parámetros pero cambiando el parámetro b de la resistencia no lineal, fue la correspondiente a $b = 0$ pues su separatriz $E - F - G$ envolvió menos trayectorias grises (sin posibilidad del efecto de aceleración).

Utilizando la tabla 4.4 usada para las últimas gráficas, recordando que contiene $b = 0$, y cambiando el primer valor del vector de estados $x_1 = 1$ a $x_1 = 0.4$, se genera el retrato de fase de la figura 4.20. En ella, se visualiza que ninguna curva puede adquirir el efecto de aceleración que se busca, pues ninguna interseca con la ordenada del punto final. Esto es así, debido a haber elegido mal el valor de x_1. En realidad, con valores superiores a $x_1 = 0.5$, se pueden encontrar conjuntos de curvas con posibilidad del efecto de aceleración. Entonces, al comparar la figura 4.20 con la 4.15, se nota la importancia de elegir correctamente el vector de estados inicial para poder hallar familias de curvas con posibilidad del efecto de aceleración.

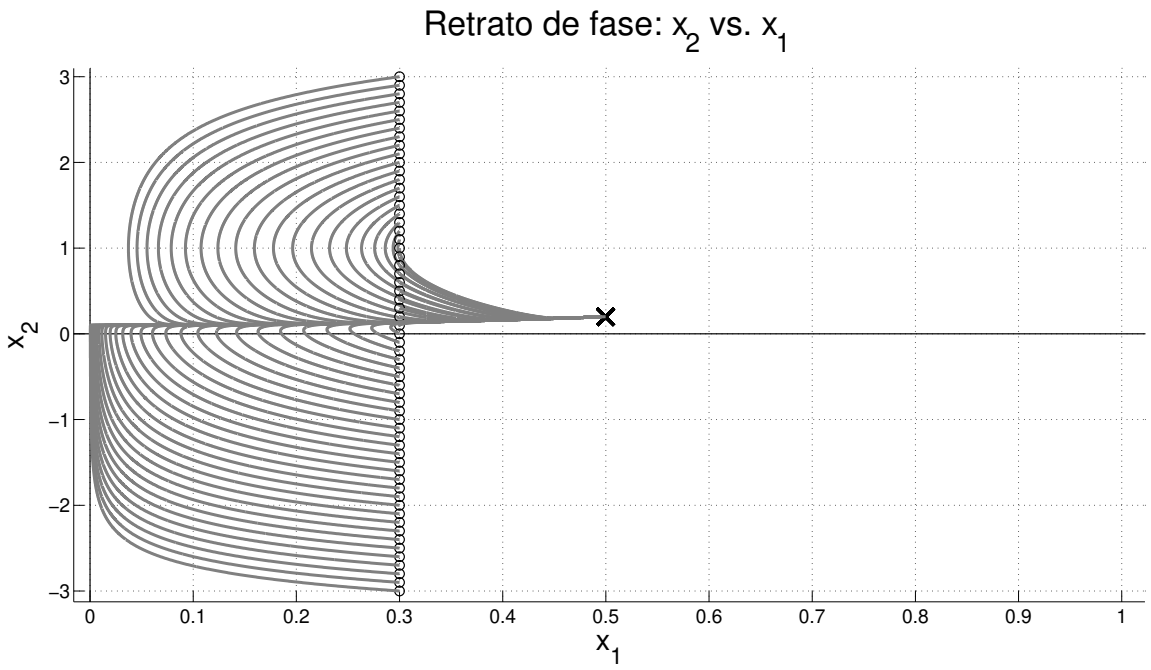


Figura 4.20: Retrato de fase con parámetros de la tabla 4.4, pero cambiando a $x_1 = 0.3$ inicial

Ganancia en tiempo

En este trabajo se le ha llamado ganancia en tiempo a la razón entre dos cantidades de tiempo para alcanzar la solución de diseño usando dos distintas estrategias. Es una manera de ver cuántas veces más rápida es una estrategia que otra para alcanzar la solución de diseño.

En las tablas 4.5, 4.6 y 4.7 se tiene para cada estrategia de diseño el número de pasos de tiempo requeridos para alcanzar la solución de diseño, el timepo total de procesamiento y la ganancia en tiempo.

Primero se presenterá la tabla 4.5, la cual muestra cómo el tiempo y los pasos se comportan con respecto al cambio del estado inicial x_1 del proceso de diseño. En el apartado "Valores iniciales"se pudo observar que para diferentes casos, el número de pasos usados por la ETD se mantuvo constante respecto al cambio de x_2, por lo tanto no hay ganancia de tiempo entre las trayectorias ETD con los distintos puntos iniciales usados. Para la tabla 4.5 se muestra la ganancia obtenida al dividir el tiempo de la trayectoria con mayor tiempo entre la de menor tiempo de la misma estrategia y usando los mismos parámetros. En el caso de las ETD la ganancia es 1 debido a que para cualquier valor inicial de x_2 el tiempo de procesamiento es el mismo. Esta tabla muestra que el valor inicial de estados influye en el tiempo del procesamiento del diseño.

Cuadro 4.5: Pasos, tiempo y ganancia de estrategias de diseño (tomando en cuenta el cambio de estado inicial)

Estrategia de diseño	Número de pasos de tiempo	Tiempo CPU (ms)	Ganancia en tiempo: $\frac{mayor\ tiempo\ de\ misma\ estrategia}{menor\ tiempo\ en\ misma\ estrategia}$
ETD (tabla 4.1)	9872 $\forall\ x_2 \in \mathcal{D}$	2.75	1
ETDM con menos pasos (tabla 4.1)	9556 en $x_2 = 2.5$	2.48	3.56
ETDM con más pasos (tabla 4.1)	35410 en $x_2 = -3$	8.82	—
ETD (tabla 4.2)	103987 $\forall\ x_2 \in \mathcal{D}$	29	1
ETDM con menos pasos (tabla 4.2)	63104 en $x_2 = -0.5$	9	5.44
ETDM con más pasos (tabla 4.2)	194379 en $x = -3$	49	—

En la tabla 4.6 se comparan las estrategias utilizando la mismos valores iniciales para el vector de estados $\mathbf{X} = [1, x_2]$, manteniendo los parámetros constantes de la tabla 4.3 a excepción del parámetro de no linealidad b de la resistencia R_n y cambiando a $b = 0.5$. Se tomó únicamente en cuenta los estados x_2 con el menor número de pasos entre los valores $x_2 \in \mathcal{D} = -3, -2.9, ..., 3$ en cada estrategia de diseño.

Cuadro 4.6: Pasos, tiempo y ganancia de estrategias de diseño, al modificar el valor de b

Estrategia de diseño	Número de pasos de tiempo	Tiempo CPU (ms)	Ganancia en tiempo: $\frac{tiempo\ EGD}{tiempo\ EGDM}$
ETD con $b = 0$	98751 $\forall\ x_2 \in \mathcal{D}$	20	1
ETD con $b = 0.5$	100419 $\forall\ x_2 \in \mathcal{D}$	19	1
ETD con $b = 1$	103987 $\forall\ x_2 \in \mathcal{D}$	29	1
ETD con $b = 1.5$	108474 $\forall\ x_2 \in \mathcal{D}$	29	1
ETDM con $b = 0$	76678 en $x_2 = 2.5$	19	1.05
ETDM con $b = 0.5$	71461 en $x_2 = -0.5$	19	1
ETDM con $b = 1$	63104 en $x_2 = -0.5$	10	2.9
ETDM con $b = 1.5$	77648 en $x_2 = -0.5$	19	1.53

Se puede observar en la tabla 4.6 que aumentando el parámetro b en 0.5 desde 0 hasta 1.5, el número de pasos utilizados en la ETD se incrementó. Su tiempo de CPU incrementó en 10 ms aunque no de manera monótona creciente respecto a b. Por otro lado, utilizando la ETDM, a medida que b creció, el número de pasos disminuyó y de nuevo creció. Un dato interesante es que con la ETDM y $b = 1$ se pudo alcanzar la solución 3 veces más rápido que con cualquiera de las otras trayectorias. Véase que el tiempo utilizado por la ETDM fue siempre menor o igual que el de la ETD.

Usando los parámetros de la tabla 4.1 pero aumentando el valor deseado *eps* de la función objetivo a $eps = 10^{-4}$ se generan los datos de la tabla 4.7. Ahí se muestran el mejor tiempo obtenido para la ETD y la ETDM. La estrategia compleja ETDM → ETD fue el mejor resultado en la región de $x_2 = -3, -2.9, ..., 3$ con una ganancia de 543.64. Sin embargo, buscando más allá de esa región se pudo hayar que con 1 único paso antes de la conmutación se obtiene la solución de diseño, con una ganancia de 3523.38.

Cuadro 4.7: Pasos, tiempo y ganancia de estrategias de diseño

Estrategia de diseño	Número de pasos de tiempo	Tiempo CPU (ms)	Ganancia en tiempo: $\frac{tiempo\ ETD}{tiempo\ estrategia\ actual}$
ETDM → ETD	8 pasos ETDM + 1 paso de conmutación	0.002061	543.64
ETDM → ETD	1 paso ETDM + 1 paso de conmutación	0.000318	3523.38
ETD	4326	1.120434	1.00
ETDM	738	0.183762	6.10

En la tabla 4.8 se muestra que la mejor estrategia compleja obtenida es 154088 veces más rápida que la trayectoria más lenta las tablas mostradas. Es importante darse cuenta de la importancia de saber elegir los parámetros y valores iniciales para el proceso de diseño, pues dependiendo de eso se pueden obtener los diseños en un tiempo muy corto o de lo contrario, consumir mucho tiempo computacional.

Cuadro 4.8: Ganancia entre la trayectoria más rápida con la más lenta

Trayectoria	Ganancia en tiempo
La más rápida (ver tabla 4.7) vs. la más lenta (tabla 4.5)	154088.0503

4.2. Efecto de aceleración en circuito pasivo de 2 nodos

Se presenta el circuito de 2 nodos, que con la programación adecuada se obtuvieron los resultados correspondientes, tales como el número de pasos y el tiempo que cada una de las 4 diferentes estrategias tardan para obtener la solución al sistema de ecuaciones que describen el circuito. Cada estrategia corresponde a distintos valores de vectores de control. Se muestran familias de curvas con capacidad de tener efecto de aceleración, además, ejemplos que evidencian su existencia.

El diagrama del circuito (figura 4.21) está formado por una resistencia no lineal y tres resistencias lineales. Se ha fijado un voltaje de entrada $V_0 = 1\ volt$ y la salida del circuito es el nodo del voltaje V_2. El otro nodo del circuito está en V_1.

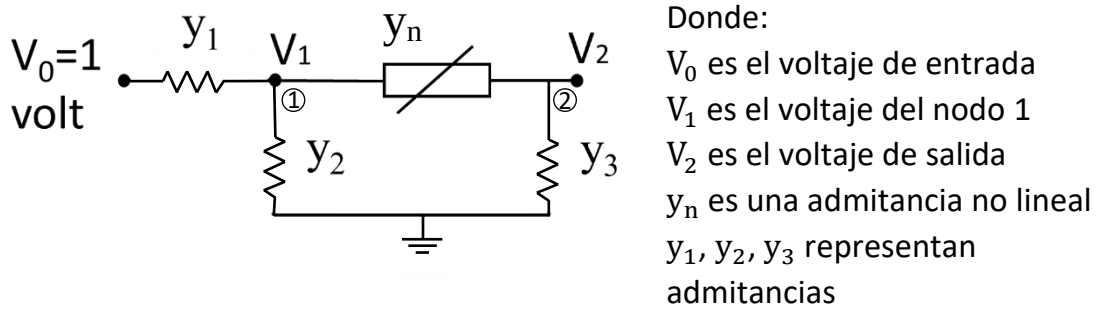


Figura 4.21: Circuito de 2 nodos

4.2.1. Ecuaciones que describen al circuito

El vector de estados es $\mathbf{X} = [x_1, \ x_2, \ x_3, \ x_4, \ x_5]$, donde cada variable es descrita por la ecuación 4.15.

$$\begin{aligned} x_1^2 &= y_1 \\ x_2^2 &= y_2 \\ x_3^2 &= y_3 \\ x_4 &= V_1 \\ x_5 &= V_2 \end{aligned} \tag{4.15}$$

La admitancia correspondiente a la resistencia no lineal que es descrita por la ecuación 4.16, donde y_0 y a_n son parámetros.

$$y_n = y_0 + a_n(x_4 - x_5)^2 \tag{4.16}$$

Las ecuaciones que describen matemáticamente al sistema son la 4.17 y la 4.18.

$$g_1(\mathbf{X}) = \left(x_1^2 + x_2^2 + y_n\right) x_4 - x_5 y_n - x_1^2 = 0 \tag{4.17}$$

$$g_2(\mathbf{X}) = x_3^2 x_5 + y_n(x_5 - x_4) = 0 \tag{4.18}$$

4.2.2. Ecuaciones de diseño con MGD

Para resolver el sistema por medio de la MGD se utilizan la función objetivo tradicional (ecuación 4.19) y la función de compensación (ecuación 4.20), para obtener la función objetivo (ver ecuación 4.21).

$$C(\mathbf{X}) = (x_5 - V_d)^2 \tag{4.19}$$

$$\varphi(\mathbf{X}) = u_1 g_1^2 + u_2 g_2^2 \tag{4.20}$$

$$F(\mathbf{X}) = C(\mathbf{X}) + \varphi(\mathbf{X}) \tag{4.21}$$

Con el vector de control $U = u_1, \ u_2$ se decide la forma de la función de penalización $\varphi(\mathbf{X})$, dependiendo del tipo de estrategia a utilizar.

Cuadro 4.9: Parámetros utilizados en el diseño del circuito de 2 nodos

$a = 1$	Constante "a" de resistencia variable R_n
b =0.7	Constante "b" de resistencia variable R_n
$paso = 10^{-4}$	Parámetro de escala
$eps = 10^{-6}$	Parámetro de comparación del criterio de convergencia de la MGD
$power = 10^{-6}$	Parámetro de comparación del criterio de convergencia del MNR
$dx_1 = 10^{-5}$	Diferencial de x_1
$dx_2 = 10^{-5}$	Diferencial de x_2
$dx_3 = 10^{-5}$	Diferencial de x_3
$dx_4 = 10^{-5}$	Diferencial de x_4
$dx_5 = 10^{-5}$	Diferencial de x_5
V_d =0.2	Valor deseado para x_2
$V_0 = 1$	Voltaje de entrada

4.2.3. Resultados

Los resultados que se obtienen son para distintas estrategias, cada una definida con distintos valores de vectores de control: la ETD, con $U = 0, 0$; la ETDM, con $U = 1, 1$; y las otras 2 estrategias correspondientes a $U = 0, 1$ y a $U = 1, 0$. Se presentan resultados para distintos valores de los parámetros del circuito que, con diferentes condiciones iniciales para algunas de las variables independientes, se generan familias de curvas distintas para cada estrategia. Luego, se muestran algunas condiciones para que exista el efecto de aceleración. Además, se tiene la comparación entre las distintas estrategias y se muestra la ventaja del efecto de aceleración.

4.2.3.1. Análisis de las proyecciones de la trayectoria

Gráficas de ETD: U=[0,0]

Con los siguientes parámetros, dados por la tabla 4.9, y con las condiciones iniciales del vector de estados X = [1.2, 1.2, 1.2, 1.2, x_5], con x_5= -4, -3.8, ..., 4, se generan las gráficas de las figuras 4.22, 4.23, 4.24 y 4.25.

Obsérvese que la gráfica de la figura 4.22, que corresponde a la estrategia tradicional de diseño, tiene un comportamiento similar a la gráfica de la ETD para resolver el circuito de 1 nodo (ver 4.2).

Gráficas de la estrategia (0,1)

La figura 4.23 es la gráfica de la estrategia $(0, 1)$, la cual tiene un comportamiento casi simétrico respecto al eje de abscisas $x_5 = 0.2$.

Gráficas de la estrategia (1,0)

La gráfica de la estrategia $(1, 0)$, que se muestra en la figura 4.24, también tiene un comportamiento similar a la de la ETD (figura 4.22).

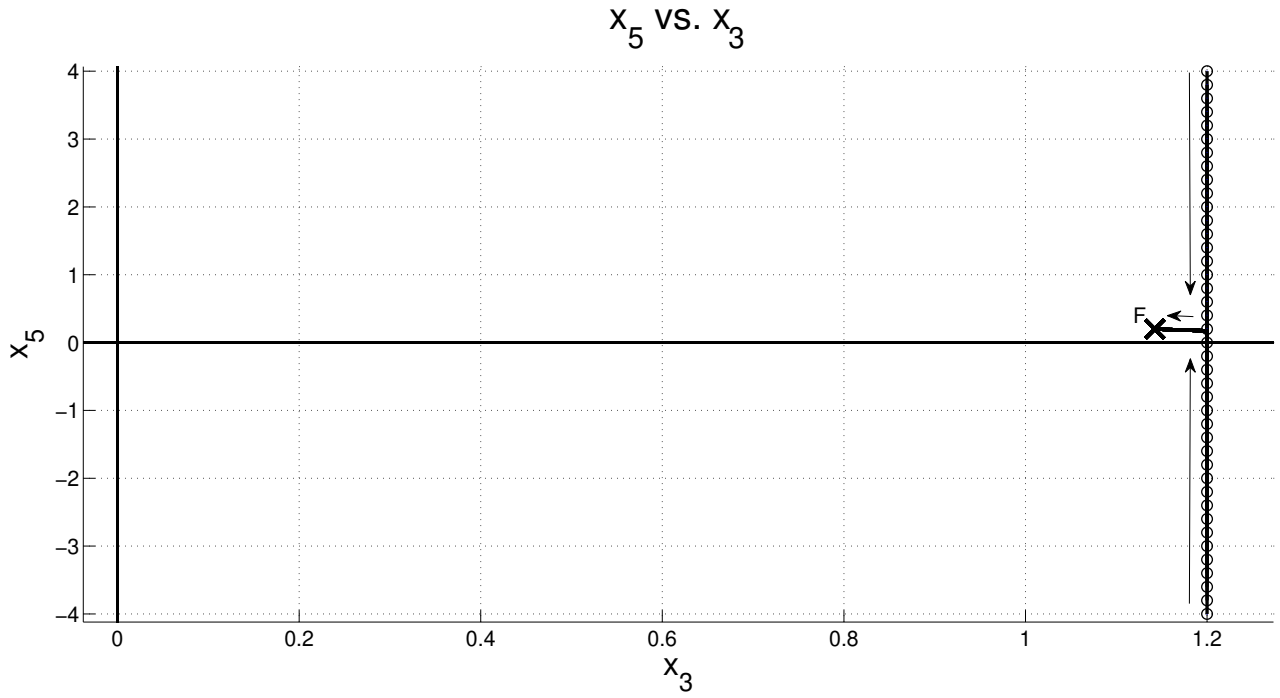


Figura 4.22: Retrato de fase con ETD para el circuito de 2 nodos

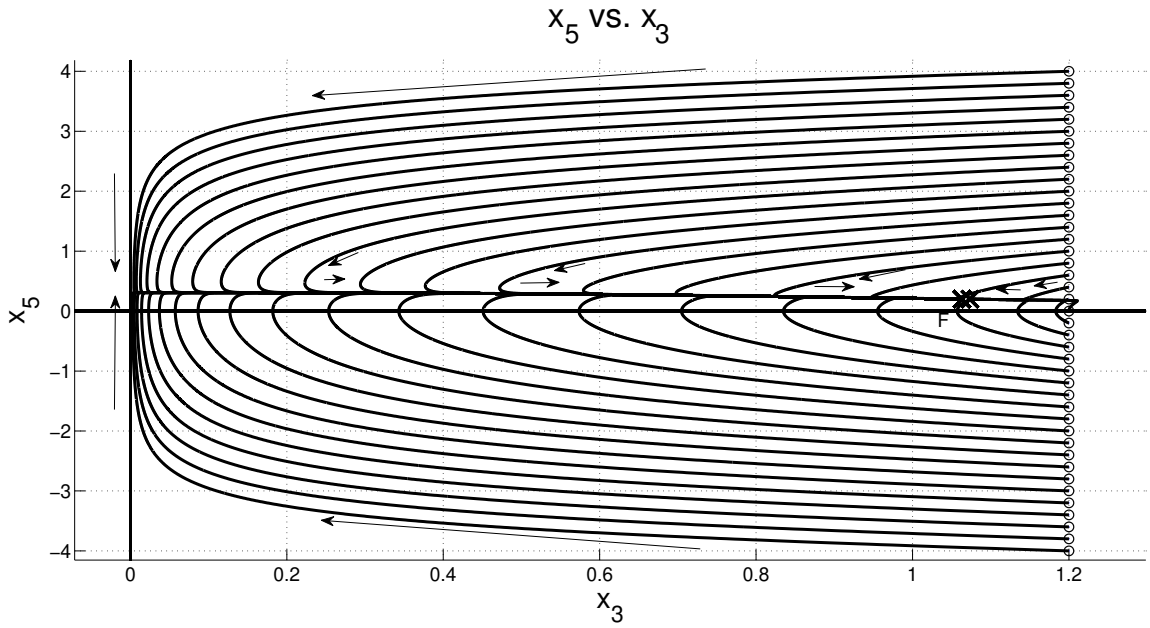


Figura 4.23: Retrato de fase con la estrategia 01

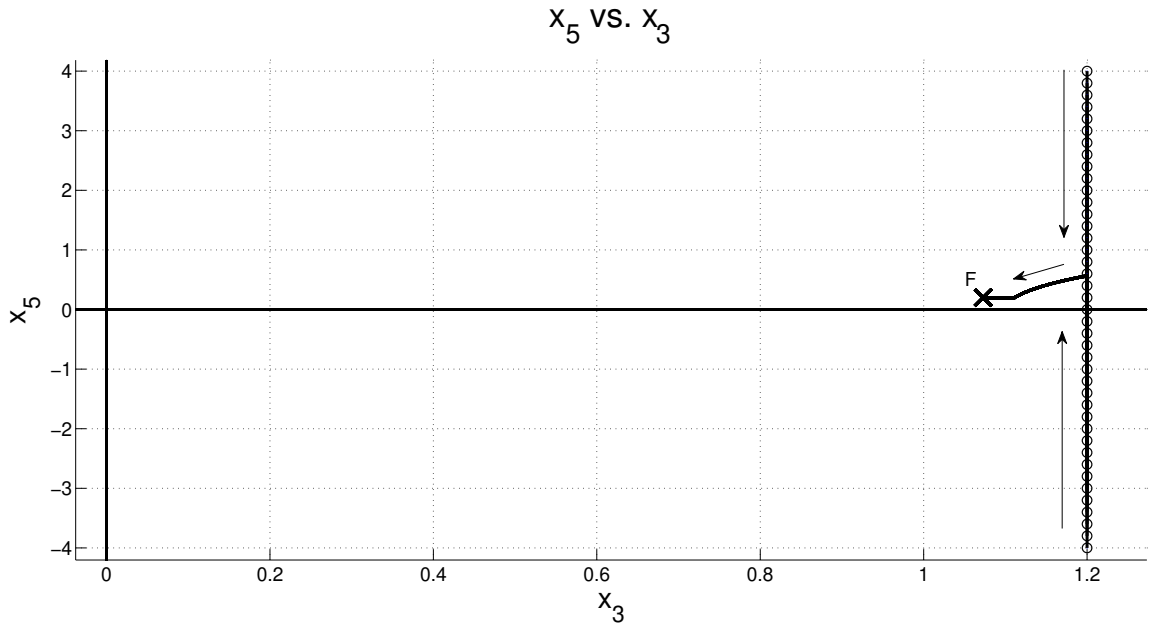


Figura 4.24: Retrato de fase con la estrategia 10

Gráficas de la ETDM

La figura 4.25 muestra la gráfica de la estrategia tradicional de diseño modificada.

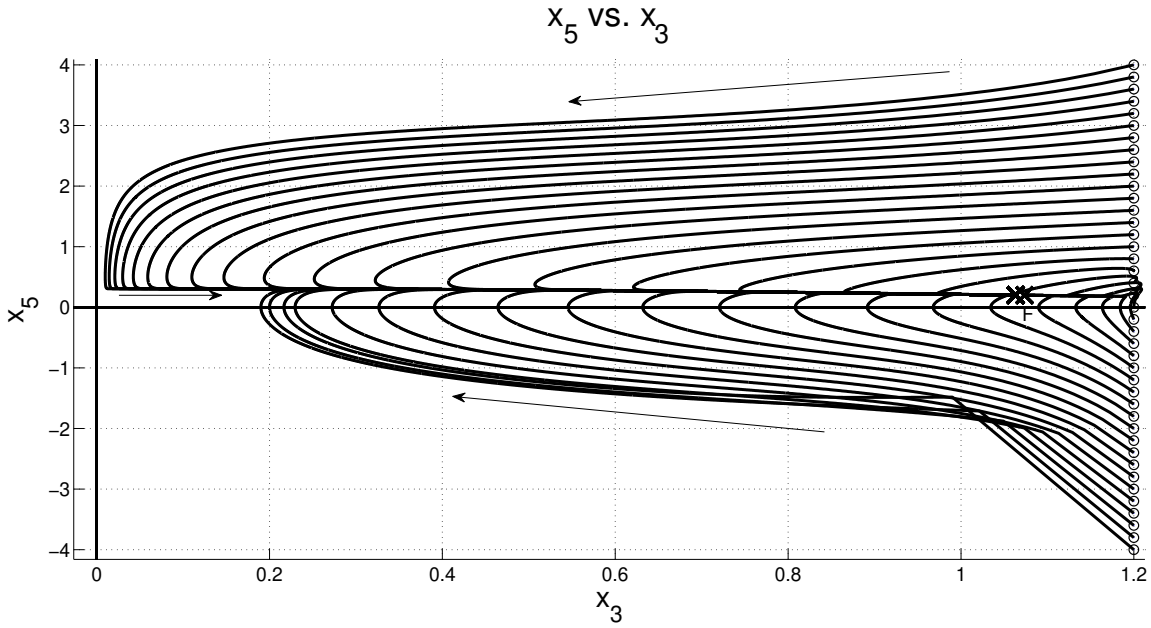


Figura 4.25: Retrato de fase con ETDM

Evidencias del efecto de aceleración

Obsérvese que en la gráfica de la figura 4.26 hay distintas conmutaciones desde la ETDM a la ETD. Las líneas verticales indican los saltos entre esas estrategias.

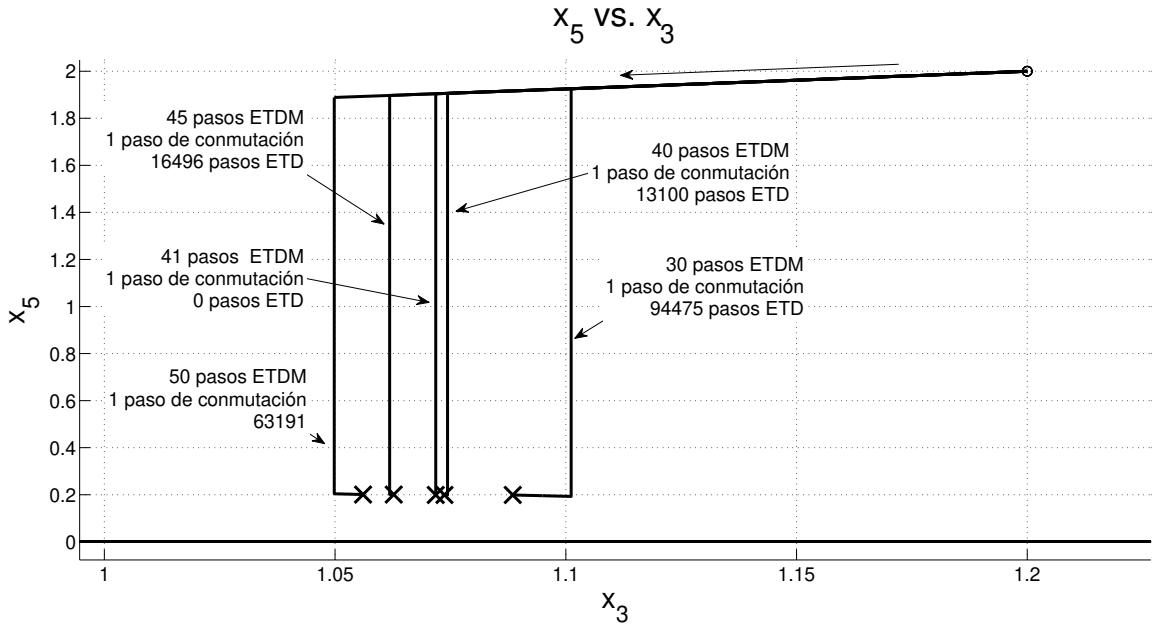


Figura 4.26: Conmutación ETDM a ETD

En la figura 4.27 se diferencian las zonas con las curvas con posibilidad del efecto de aceleración: las curvas de color negro están en las zonas en las que sí existe y de gris en las que no. La curva más gruesa de color negro es la proyección de la separatriz de aquellas regiones. Las intersecciones realizadas con la línea vertical discontinua y las curvas, indican los puntos en los cuales se ejecuta la conmutación hacia la estrategia tradicional de diseño.

En la figura 4.28 se tienen conmutaciones entre la estrategia [0,1] y la estrategia [0,0]. Todas las líneas verticales indican ese cambio. Las verticales de mayor importancia son las que cruzan con alguna cruz negra (soluciones del diseño) pues son las que tienen el mayor efecto de aceleración entre todas las conmutaciones. Que exista más de una cruz es debido a la existencia de más de una solución dentro del margen de error dado al programa. Mientras menor sea el rango del error, las soluciones serán más parecidas y por tanto las cruces más cercanas entre sí.

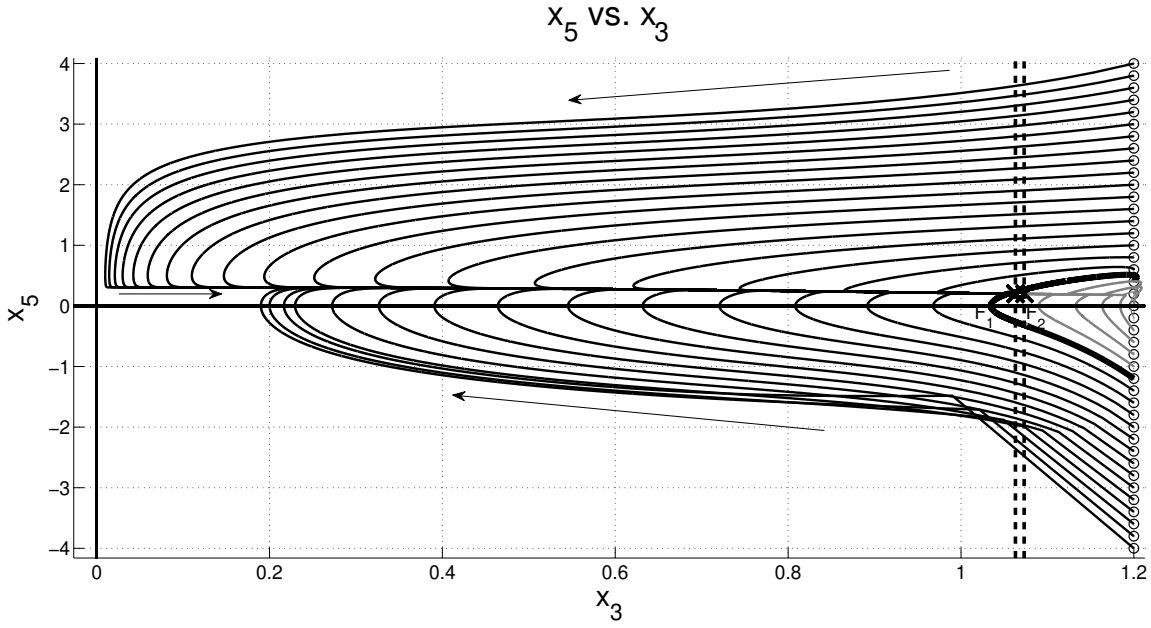


Figura 4.27: Separatrices de la ETDM

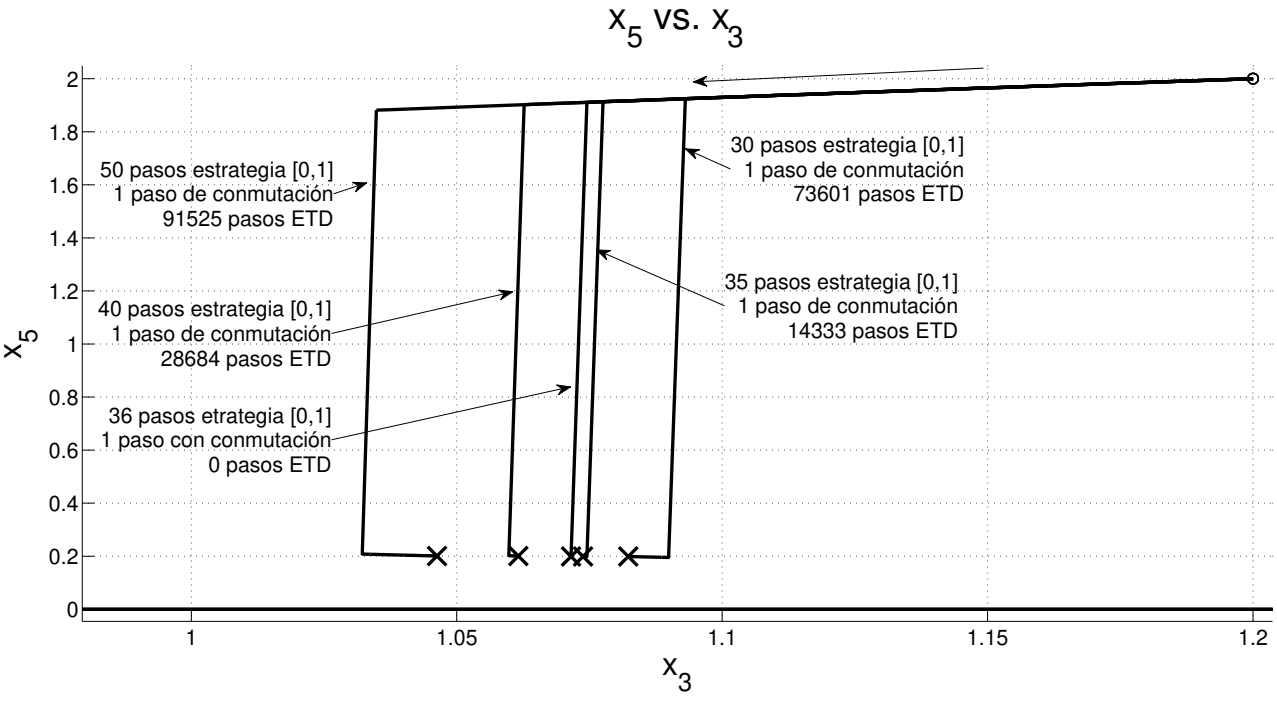


Figura 4.28: Conmutación desde la estrategia [0,1] hacia la ETD

La gráfica de la figura 4.29 muestra la separatriz (curva gruesa) diferenciando las curvas del método $(0, 1)$, entre negras y grises, con posibilidad de efecto de aceleración y sin éste,

respectivamente.

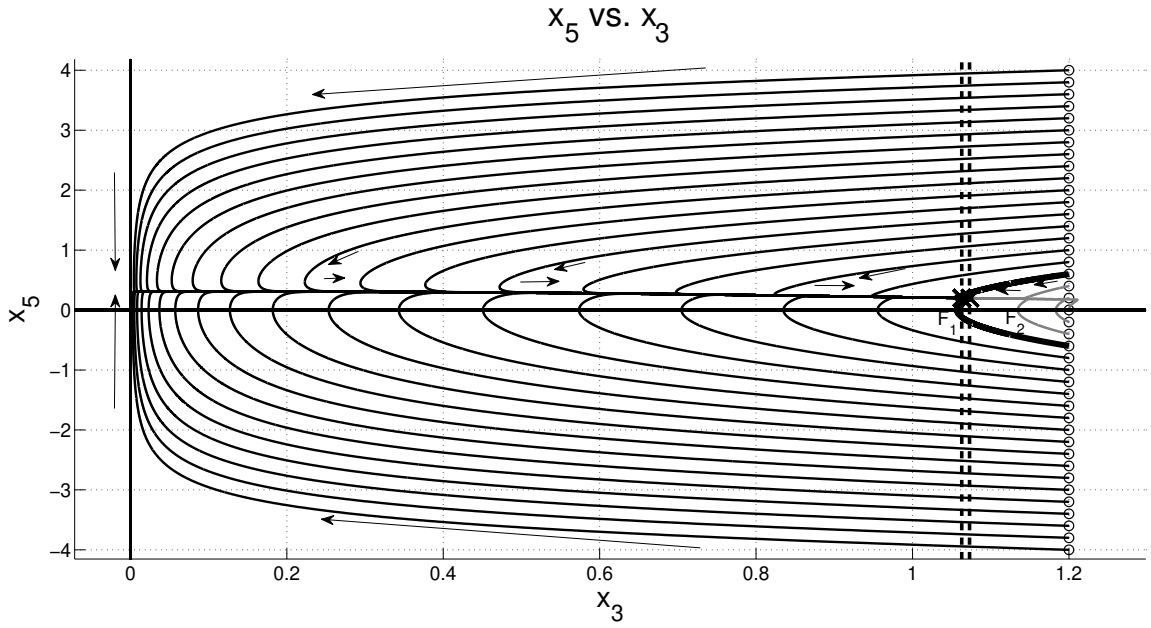


Figura 4.29: Separatrices de la estrategia [0,1]

4.2.3.2. Ganancia en el tiempo de ejecución

La tabla 4.10 indica el número de pasos y el tiempo necesarios que le toma a cada estrategia llegar a la solución del diseño del sistema. También se tiene la ganancia de tiempo con respecto a la ETD, la cual se obtiene al dividir el tiempo de la respectiva estrategia entre el de la estrategia tradicional de diseño.

Estrategia	Pasos	Tiempo en μs	Ganancia
$[0,0]$	176837	381679.68	1
$[0,1]$	23149	54208.48	7.04
$[1,0]$	245297	549690.95	0.69
$[1,1]$	99253	103503.01	3.69
$[1,1] \rightarrow [0,0]$	41	43.87	8700.01
$[0,1] \rightarrow [0,0]$	37	86.46	4414.51

Cuadro 4.10: Tabla comparativa de iteraciones y tiempo

La figura 4.30 contiene la gráfica comparativa entre las distintas estrategias. Las barras de color azul indican el número de pasos para alcanzar la solución, y en anaranjado se indica el tiempo en μs. Se observa que la estrategia que requiere de un mayor lapso para alcanzar la solución es la $[0,1]$. La información correspondiente a las combinaciones de estrategias no es posible verse a menos que se le aplique un zoom, el cual permite observar que la estrategia $[0,1] \rightarrow [0,0]$ es la que requiere del menor tiempo para solucionar el sistema de 2 nodos.

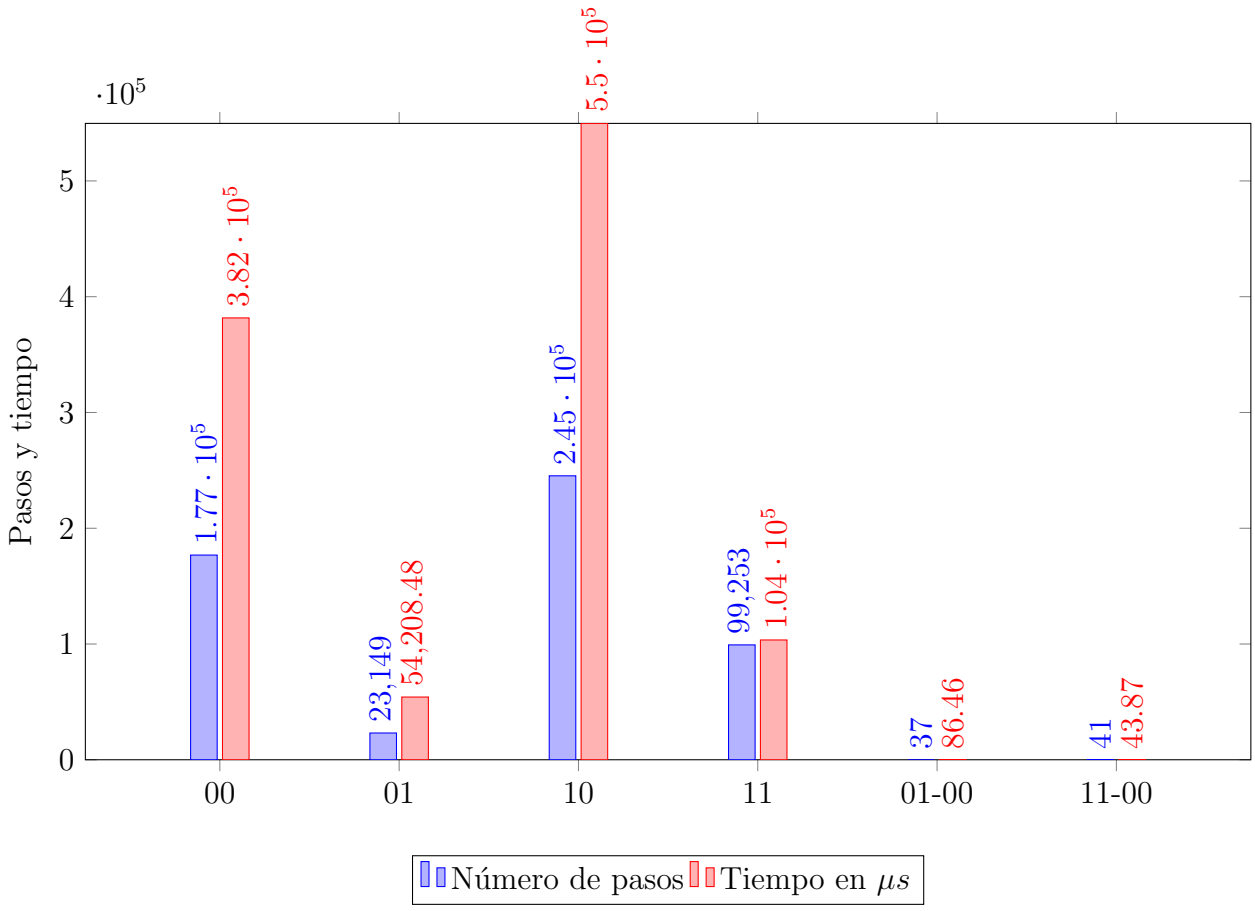


Figura 4.30: Pasos y tiempo vs. estrategia de diseño utilizada

De la figura 4.30 es posible obtener con facilidad la tabla 4.11 para colocar las estrategias en orden ascendente en función del tiempo (primera columna) y en función del número de pasos (segunda columna). Obsérvese que el orden de las estrategias cambia dependiendo del criterio utilizado. Se observa que a pesar de que la estrategia $[0,1] \rightarrow [0,0]$ tiene un menor número de pasos frente a la estrategia $[1,1] \rightarrow [0,0]$, no es argumento suficiente para creer que es la mejor opción a utilizar para resolver el problema de 2 nodos, pues requiere de mayor tiempo computacional que el requerido para la ETDM.

La figura 4.31 representa la ganancia en tiempo que cada estrategia tiene en comparación con la ETD. Es de llamar la atención que la estrategia $(1,1) \rightarrow (0,0)$ tiene una ganancia superior a todas, de hasta casi 8700.01 veces, lo cual demuestra la existencia de un efecto de aceleración grande en este ejemplo. También se observa que las estrategias sin conmutación tienen ganancias muy pequeñas.

Cuadro 4.11: Estrategias ordenadas de forma ascendente por número de pasos y tiempo de procesamiento

Estrategias ordenadas por número de pasos (de menor a mayor)	Estrategias ordenadas por tiempo (de menor a mayor)
01 → 00	11 → 00
11 → 00	01 → 00
01	01
11	11
00	00
10	10

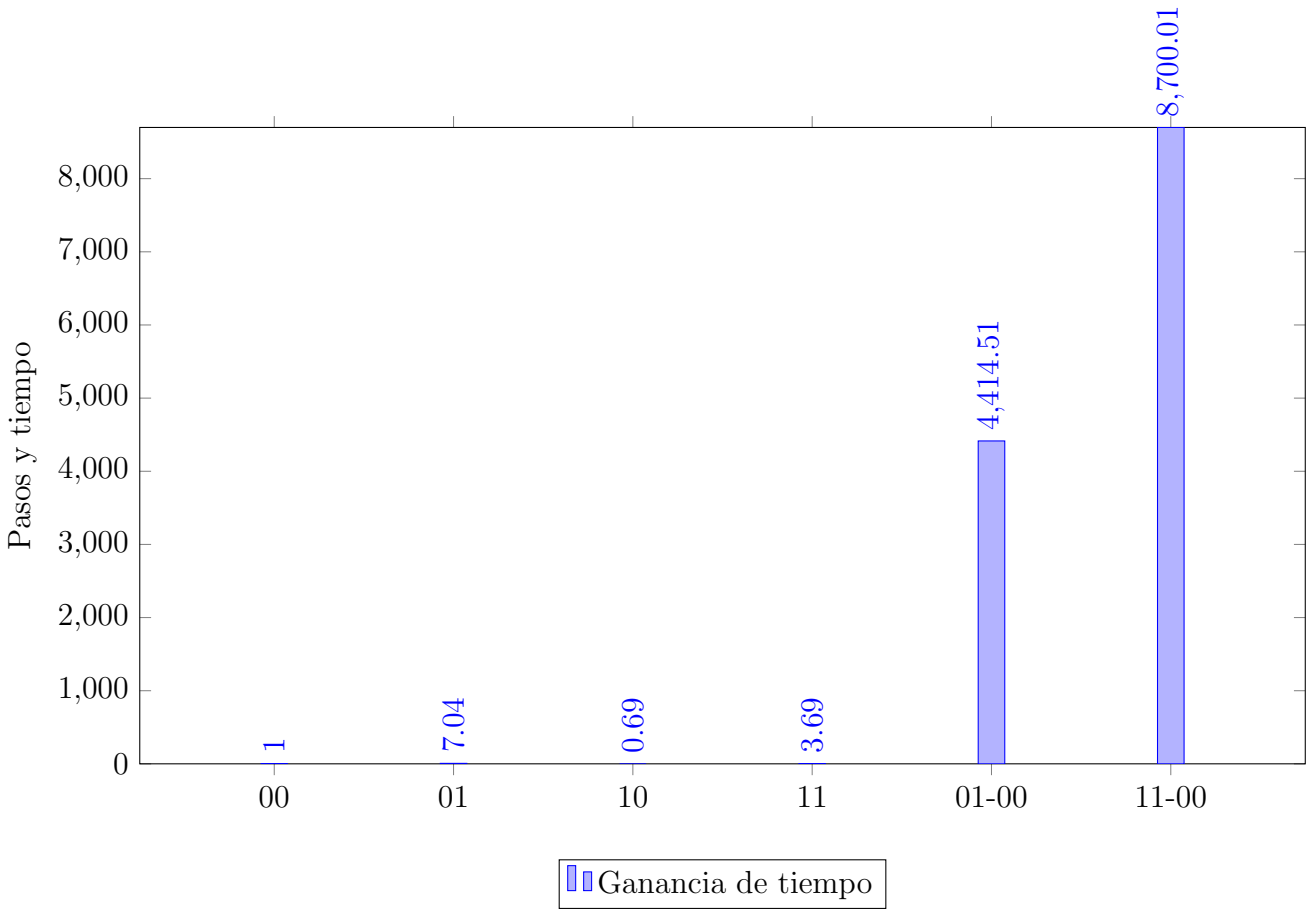


Figura 4.31: Ganancia vs. estrategia de diseño utilizada

4.3. Efecto de aceleración en circuitos activos

4.3.1. Amplificador con transistor

Diferentes modelos del transistor (cuyo diagrama se muestra en figura 4.32) se han propuesto para describir su comportamiento, entre ellos el modelo de Ebers-Moll, el cual describe ecuaciones para 4 regiones: la región normal activa, la región inversa, la región de saturación y la región de corte. Computacionalmente este método es utilizado por el software SPICE2 (Massobrio & Antognetti, 1993) con algunas modificaciones.

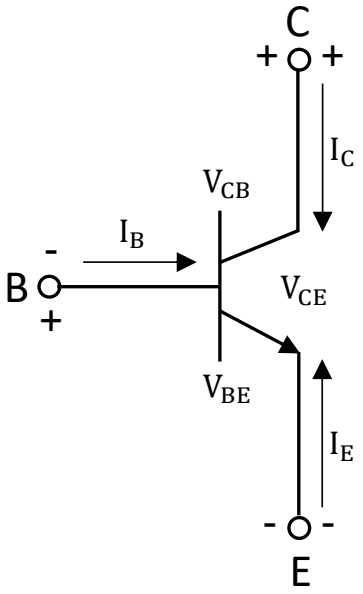


Figura 4.32: Transistor

Región normal activa

En la situación en la que el transistor tenga a $V_{BE} > -5\frac{kT}{q}$ y a $V_{BC} <= -5\frac{kT}{q}$, el transistor está en la región normal activa y se utilizan las ecuaciones 4.22 y 4.23.

$$I_c = I_s(e^{q\frac{V_{BE}}{kT}} + \frac{1}{\beta_R}) + [V_{BE} - (1 + \frac{1}{\beta_R})V_{BC}]GMIN \tag{4.22}$$

$$I_B = I_s[\frac{1}{\beta_F} + (e^{q\frac{V_{BE}}{kT}} - 1) - \frac{1}{\beta_R}] + (\frac{V_{BE}}{\beta_F} + \frac{V_{BC}}{\beta_R})GMIN \tag{4.23}$$

Región inversa

Cuando $V_{BE} <= -5\frac{kT}{q}$ y $V_{BC} > -5\frac{kT}{q}$ se dice que el transistor está en la región inversa, por lo que se usan las ecuaciones 4.24 y 4.25.

$$I_c = -I_s[e^{q\frac{V_{BC}}{kT}} + \frac{1}{\beta_R}(e^{q\frac{V_{BC}}{kT}} - 1)] + [V_{BE} - (1 + \frac{1}{\beta_R})V_{BC}]GMIN \tag{4.24}$$

$$I_B = -I_s[\frac{1}{\beta_F} - \frac{1}{\beta_R}(e^{q\frac{V_{BC}}{kT}} - 1)] + (\frac{V_{BE}}{\beta_F} + \frac{V_{BC}}{\beta_R})GMIN \tag{4.25}$$

Región de saturación

La región de saturación ocurre cuando $V_{BE} > -5\frac{kT}{q}$ y $V_{BC} > -5\frac{kT}{q}$. Las ecuaciones 4.26 y 4.27 se utilizan para este caso.

$$I_c = I_s[(e^{q\frac{V_{BE}}{kT}} - e^{q\frac{V_{BC}}{kT}}) - \frac{1}{\beta_R}(e^{q\frac{V_{BC}}{kT}} - 1)] + [V_{BE} - (1 + \frac{1}{\beta_R})V_{BC}]GMIN \tag{4.26}$$

$$I_B = I_s[\frac{1}{\beta_F}(e^{q\frac{V_{BE}}{kT}} - 1) + \frac{1}{\beta_R}(e^{q\frac{V_{BC}}{kT}} - 1)] + (\frac{V_{BE}}{\beta_F} + \frac{V_{BC}}{\beta_R})GMIN \tag{4.27}$$

Región de corte

Cuando $V_{BE} <= -5\frac{kT}{q}$ y $V_{BC} <= -5\frac{kT}{q}$, entonces el transistor se haya en la región de corte, en donde las ecuaciones 4.28 y 4.29 son las que describen matemáticamente al transistor.

$$I_c = \frac{I_s}{\beta_R} + [V_{BE} - (1 + \frac{1}{\beta_R})V_{BC}]GMIN \tag{4.28}$$

$$I_B = -I_s(\frac{\beta_F + \beta_R}{\beta_F \beta_R}) + (\frac{V_{BE}}{\beta_F} + \frac{V_{BC}}{\beta_R})GMIN \tag{4.29}$$

Sin embargo, a causa de las componentes exponenciales, su aplicación en programación puede provocar problemas ya que pueden aparecer números demasiado grandes. Tomando en cuenta esto, se ha decidido que el máximo exponente del número de Euler de las ecuaciones de Ebers-Moll sea 34. Así que cada vez que el exponente sea mayor a 34, se fijará su valor en este último número.

Diagrama del circuito con transistor

En la figura 4.33 se tiene el circuito utilizado para mostrar el efecto de aceleración. Es un circuito que involucra un transistor con una resistencia conectada en cada una de sus tres terminales.

Ecuaciones del circuito con transistor

Las ecuaciones que describen al circuito de la figura 4.33 forman un sistema de ecuaciones constituido por las ecuaciones 4.30, 4.31 y 4.32.

$$g_1(\mathbf{X}) \equiv I_B + (V_B - E_1)\, y_1 = 0 \tag{4.30}$$

$$g_2(\mathbf{X}) \equiv I_E - V_E y_2 = 0 \tag{4.31}$$

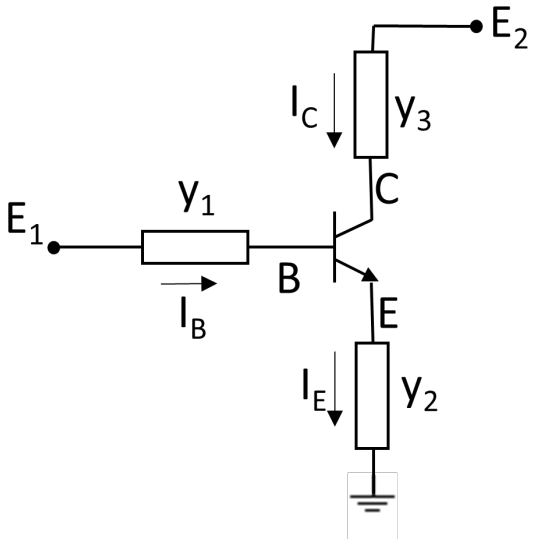


Figura 4.33: Diagrama del circuito con transistor

$$g_3(\mathbf{X}) \equiv I_c + (V_c - E_2)\, y_3 = 0 \tag{4.32}$$

El vector de estados del circuito está conformado por 6 variables (ver ecuación 4.33). Cada elemento del mismo está relacionado con alguna de las variables del circuito con transistor de la manera mostrada en la ecuación 4.34.

$$\mathbf{X} = (x_1,\ x_2, x_3, x_4,\ x_5,\ x_6) \tag{4.33}$$

$$\begin{aligned} x_1^2 &= y_1 \\ x_2^2 &= y_2 \\ x_3^2 &= y_3 \\ x_4 &= V_1 = V_B \\ x_5 &= V_2 = V_E \\ x_6 &= V_3 = V_C \end{aligned} \tag{4.34}$$

En la tabla 4.12 se muestran los valores de los parámetros utilizados para generar los resultados de diseño obtenidos con la MGD.

Cuadro 4.12: Conjunto de parámetros utilizados en el diseño del circuito con transistor

$paso = 10^{-4}$	Parámetro de escala
$eps = 10^{-6}$	Parámetro de comparación del criterio de convergencia de la MGD
$power = 10^{-6}$	Parámetro de comparación del criterio de convergencia del MNR
$Vced = 5.0$	Voltaje colector emisor deseado
$Vbed = 1.2$	Voltaje base emisor deseado
E_1 =4	Valor de entrada
$E_2 = 20$	Voltaje de entrada
$dx_1 = 10^{-5}$	Diferencial de x_1
$dx_2 = 10^{-5}$	Diferencial de x_2
$dx_3 = 10^{-5}$	Diferencial de x_3
$dx_4 = 10^{-5}$	Diferencial de x_4
$dx_5 = 10^{-5}$	Diferencial de x_5
$dx_6 = 10^{-5}$	Diferencial de x_6
$T = 27°C$	Temperatura

Para la estrategia [0,0,0] se generó la proyección del retrato de fase mostrada en la figura 4.34.

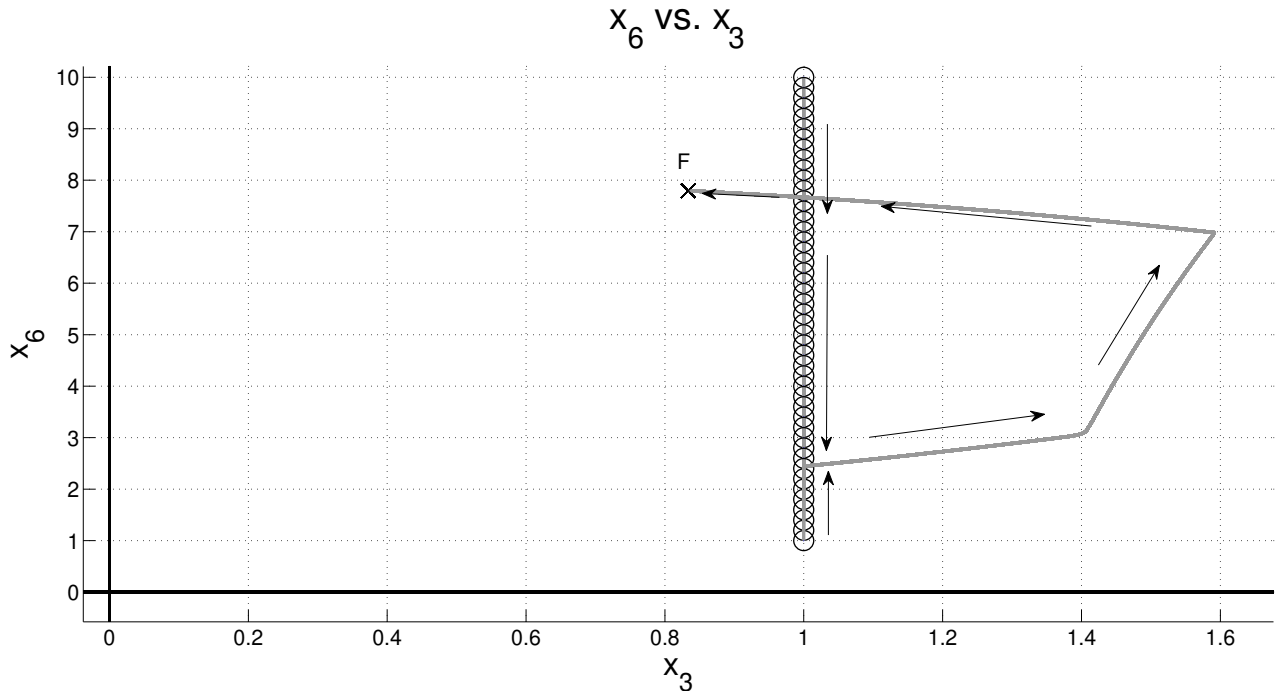


Figura 4.34: Proyección del retrato de fase con con la ETD

Con la estrategia [0,0,1] se obtuvo la proyección del retrato de fase mostrada en la figura 4.35.

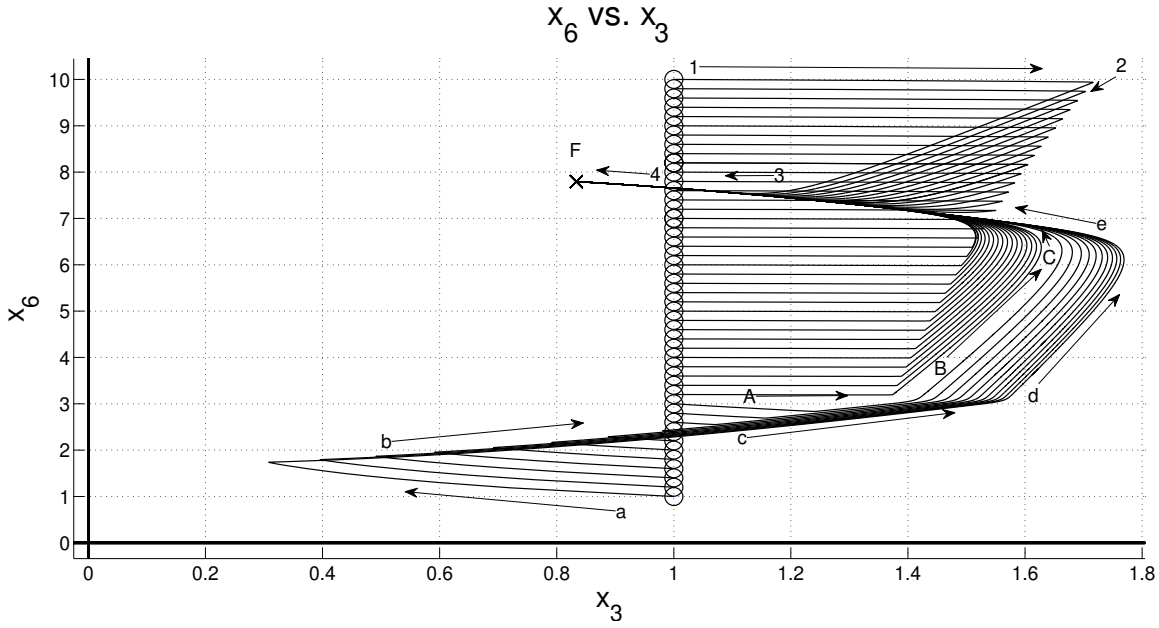


Figura 4.35: Proyección del retrato de fase con la estrategia [0,0,1]

Usando la estrategia [0,1,0] se graficó la proyección del retrato de fase mostrada en la figura 4.36.

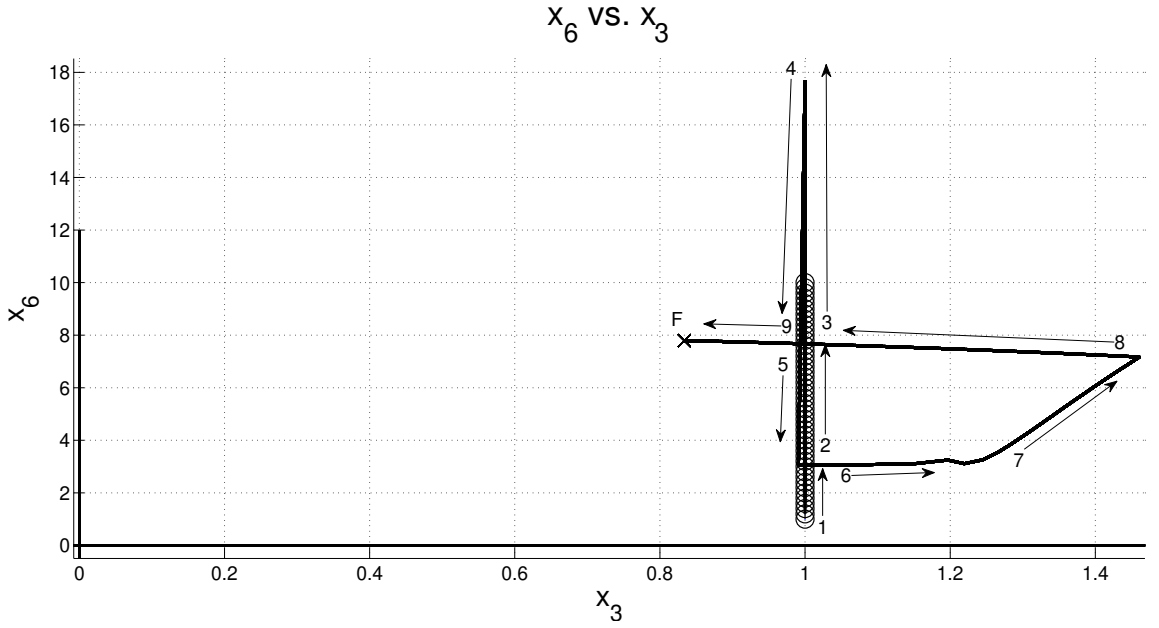


Figura 4.36: Proyección del retrato de fase con la estrategia [0,1,0]

Utilizando la estrategia [0,1,1] se generó la proyección del retrato de fase mostrada en la figura 4.37.

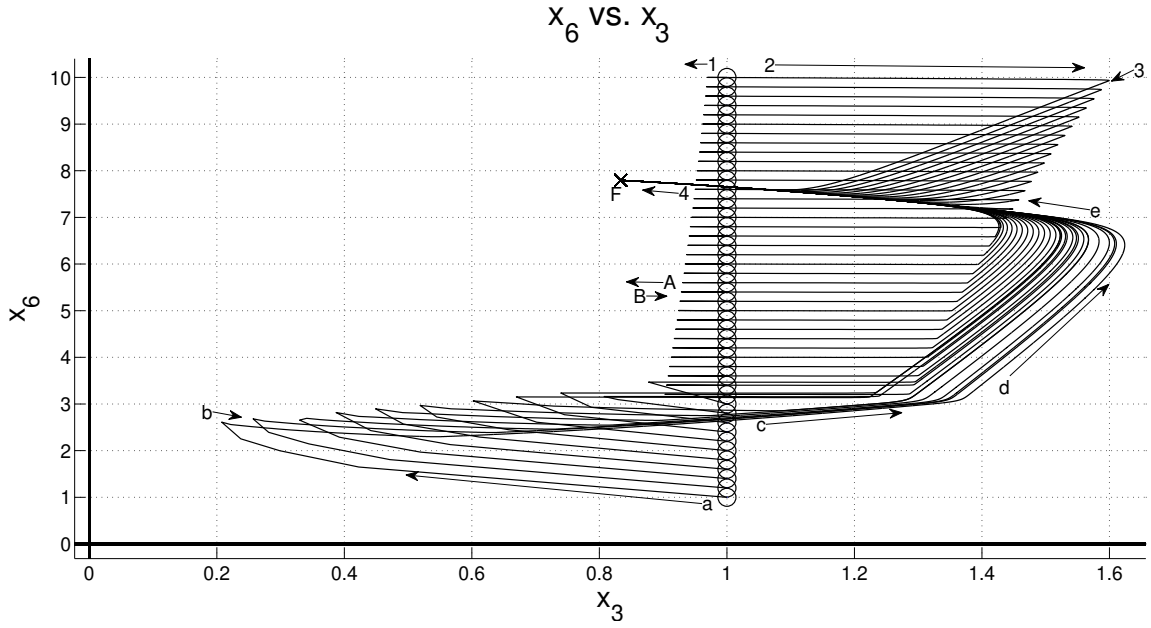


Figura 4.37: Proyección del retrato de fase con la estrategia [0,1,1]

Para la estrategia [1,0,0] se graficó la proyección del retrato de fase mostrada en la figura 4.38.

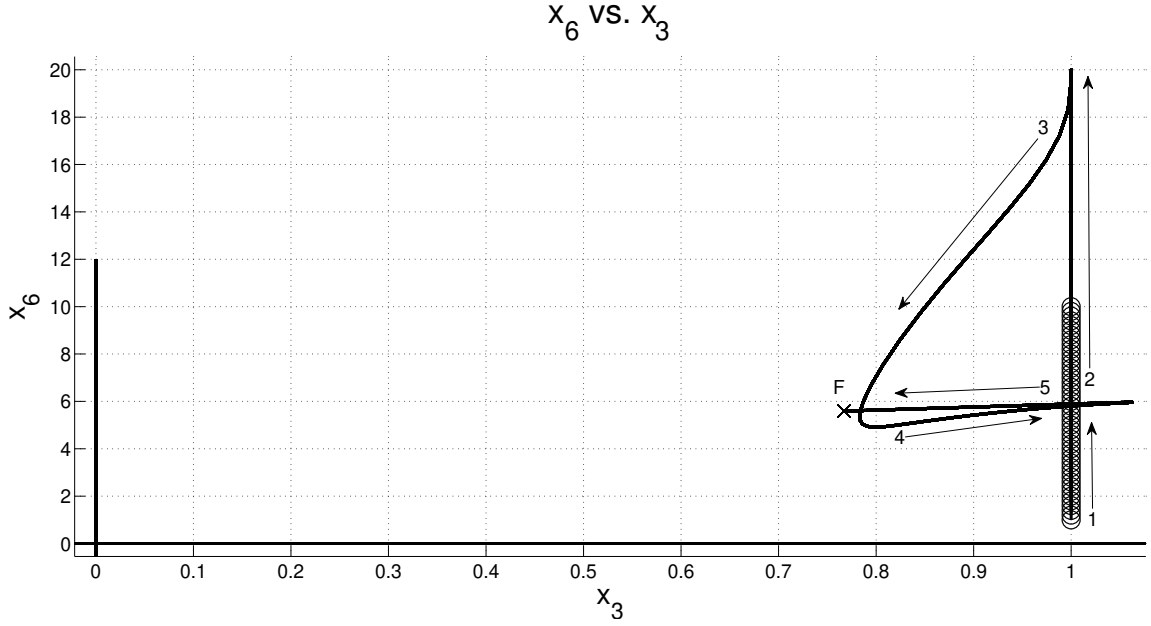


Figura 4.38: Proyección del retrato de fase con la estrategia [1,0,0]

Con la estrategia [1,0,1] se obtuvo la proyección del retrato de fase mostrada en la figura 4.39.

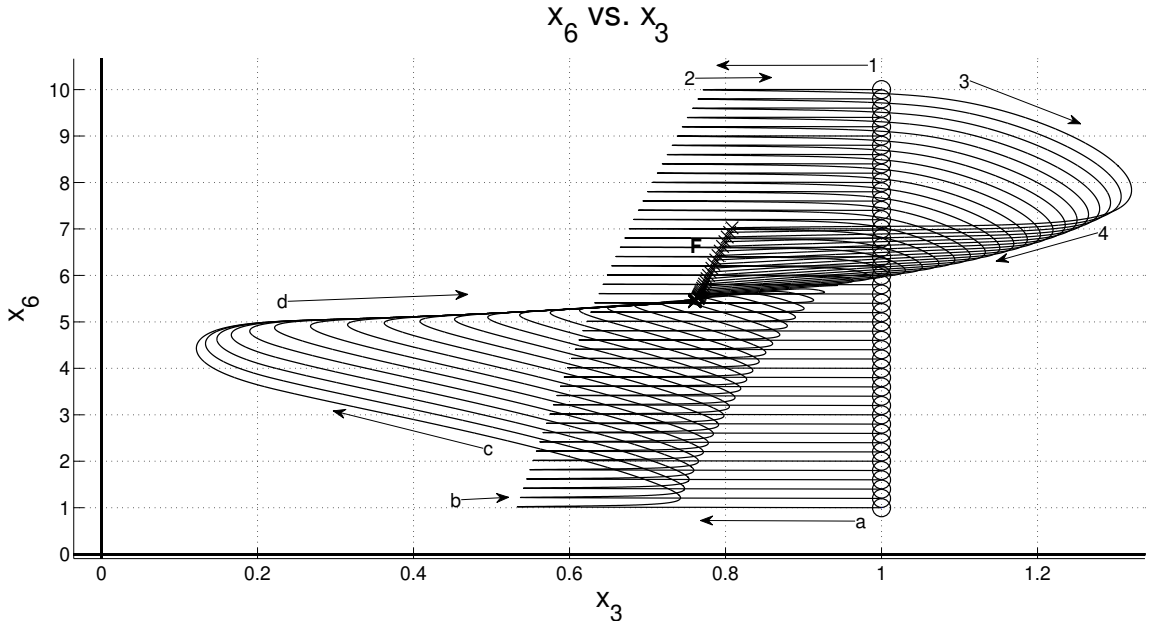


Figura 4.39: Proyección del retrato de fase con la estrategia [1,0,1]

Utilizando la estrategia [1,1,0] se trazó la proyección del retrato de fase que se tiene en la figura 4.40. Cada curva se completó después de $1,163,465,124$ pasos.

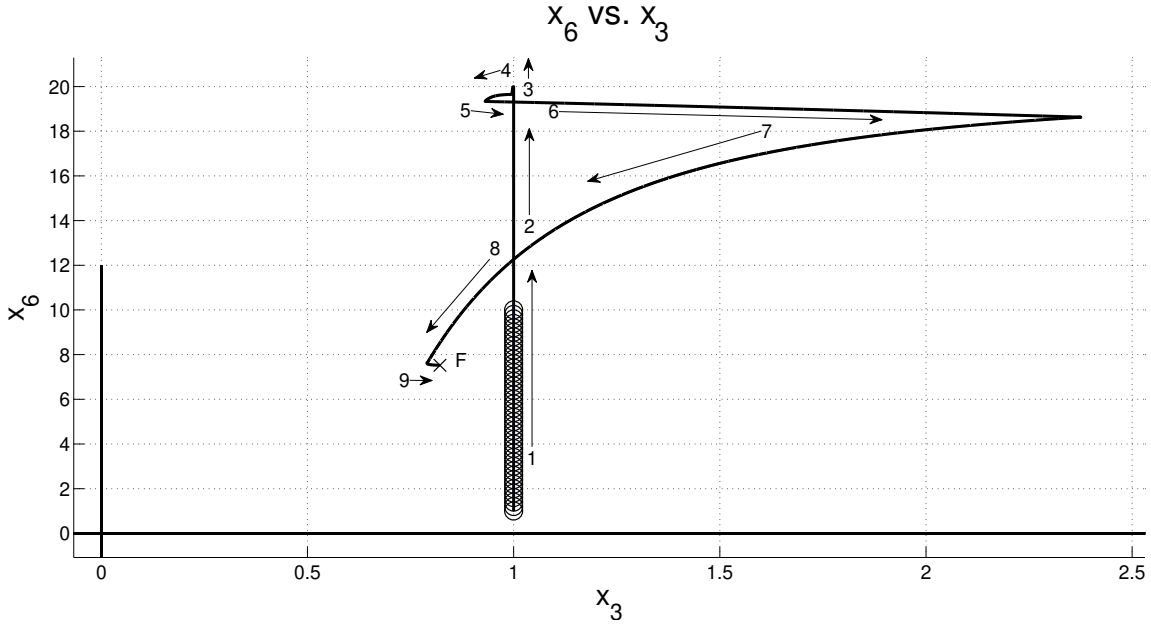


Figura 4.40: Proyección del retrato de fase con la estrategia [1,1,0]

Con la estrategia [1,1,1] se tiene la proyección del retrato de fase mostrada en la figura 4.41.

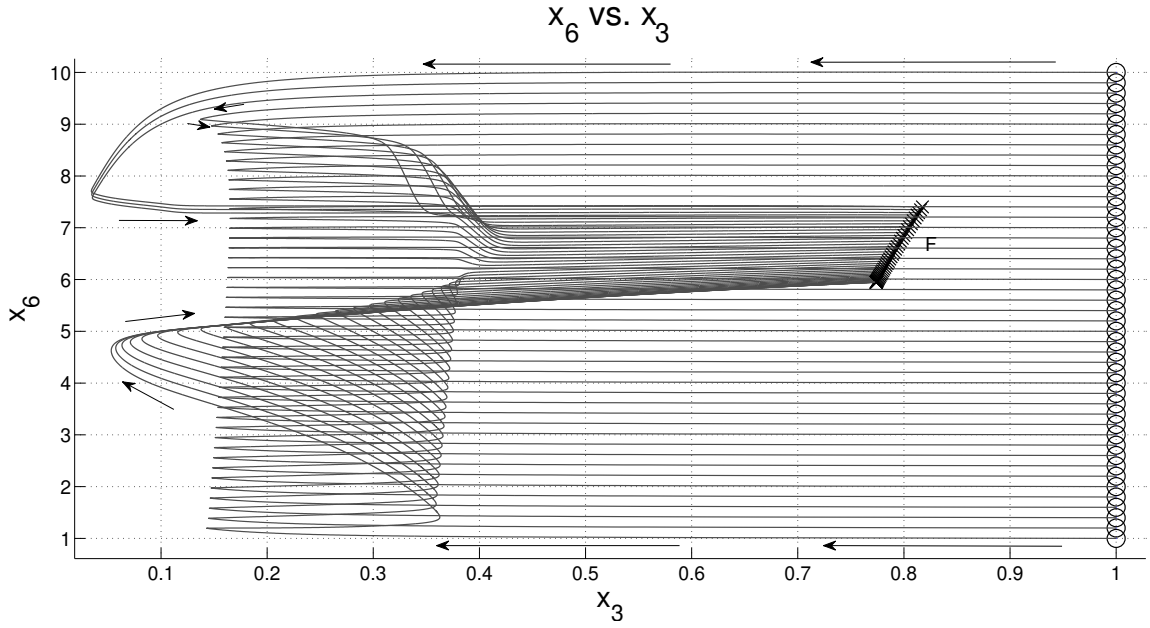


Figura 4.41: Proyección del retrato de fase con la ETDM

Se obtuvo una estrategia compleja mostrada en la figura 4.42.

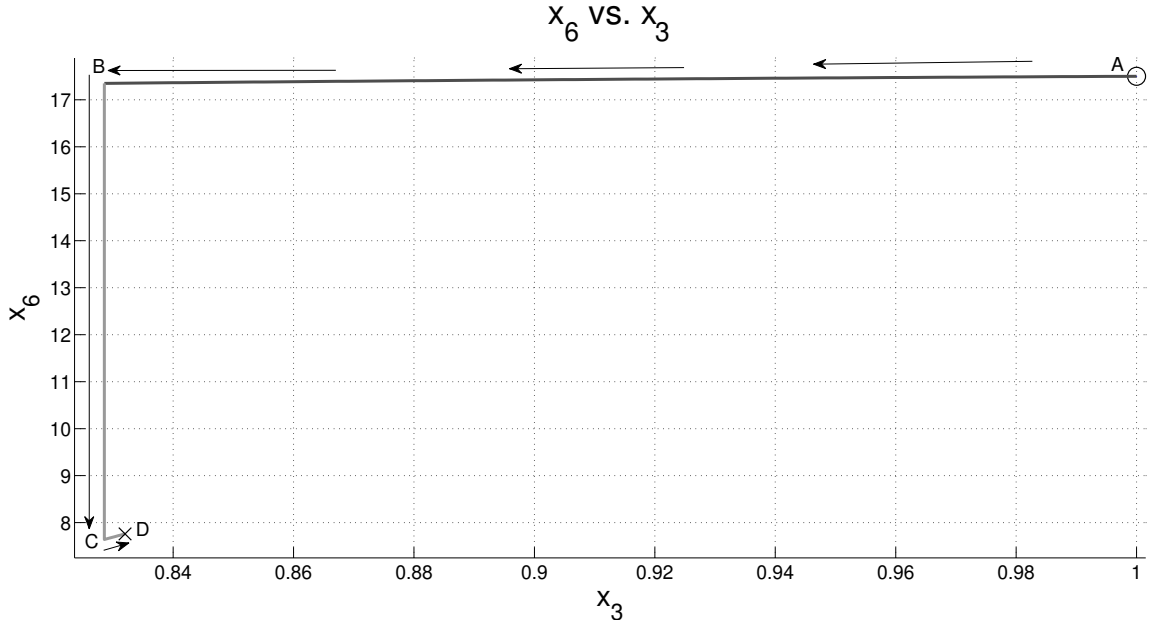


Figura 4.42: Conmutación de ETDM → a ETD

4.3.2. Ganancia en el tiempo

Como resultado del análisis de varias estrategias y sus proyecciones, se obtuvieron los siguientes resultados:

1. ETD [0,0,0] incluye 407,465 pasos y un tiempo de CPU T=4,514.712 ms.
2. ETMD [1,1,1] incluye 589,945 pasos y timepo computacional de T=2,524.965 ms.
3. La estrategia compleja [1,1,1]-[0,0,0] incluye 86 pasos con la estrategia [1,1,1], 1 paso de conmutación y 27 pasos con la ETD, y tiene un tiempo de CPU de T=0.645 ms. La línea AB corresponde a la estrategia [1,1,1], después hay una conmutación a la ETD, entonces la curva BCD es generada.

La ganancia en tiempo en el caso de la estrategia [1,1,1] es 1.788 y con la estrategia compleja la ganancia de tiempo es de 6,998.686.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

La estrategia tradicional para el diseño de circuitos y sistemas electrónicos no es óptima en tiempo. Se ha propuesto y analizado una metodología generalizada, que resuelve el problema de diseño óptimo en tiempo computacional como un problema de tiempo mínimo de la teoría de control. De la aplicación de esta metodología, se puede concluir lo siguiente.

- La aplicación de la idea general del diseño de circuitos electrónicos muestra el surgimiento de un gran conjunto de diferentes estrategias de diseño. Cada estrategia tiene su propio número de operaciones y tiempo de ejecución en la computadora. Entre este conjunto de diferentes estrategias existen algunas estrategias óptimas que minimizan el tiempo computacional para el diseño.

- La idea de introducir un vector de control en el sistema de ecuaciones básicas permite generalizar matemáticamente el problema de diseño y obtener la capacidad de controlar el movimiento del procedimiento de diseño y comparar diferentes trayectorias en el retrato de fase.

- La comparación de la estrategia de diseño tradicional contra otras estrategias que aparecen dentro de la metodología generalizada muestra la ventaja en tiempo computacional para algunas estrategias nuevas.

- Basado en el efecto de aceleración, se ha demostrado que las estrategias complejas, que incluyen estrategias tradicional y otras, reducen significativamente el tiempo de cálculo. Se encontraron condiciones suficientes para dicho efecto.

- Este efecto existe debido al muy diferente comportamiento de las trayectorias de diseño que tienen distinto vector de control y diferentes puntos iniciales del espacio de diseño.

- Se ha analizado la influencia de los valores iniciales para que el efecto de aceleración se presente. Se ha estudiado la existencia de la separatriz que divide al espacio de diseño en

dos zonas: una con curvas que tienen la posibilidad de tener un efecto de aceleración y otra zona en la que no.

- Una estrategia de diseño óptima o casi óptima permite reducir el tiempo de cálculo en varios órdenes de magnitud.
- La metodología general de diseño permite obtener grandes ganancias en tiempo del proceso de diseño, en comparación de la obtenida con las estrategias sin conmutación, tanto en circuitos pasivos como en activos.
- El análisis del efecto de aceleración y la elección del punto de partida aportan información necesaria sobre las propiedades y estructura de la estrategia óptima. Esta información sirve como un paso importante hacia la construcción de un algoritmo de diseño óptimo
- Los valores numéricos de los parámetros del sistema influyen en el comportamiento de las trayectorias del proceso de diseño.
- El tiempo del proceso de diseño y el comportamiento de su trayectoria dependen del tamaño del parámatro de escala.
- En el caso de los circuitos pasivos, para tener el efecto de aceleración en vertical, es necesario que algún punto de la trayectoria del proceso de diseño se encuentre dentro de una región delimitada por las 2 ordenadas más alejadas entre sí que atraviesan a una misma región de puntos de solución.

Apéndice A

Teorema del punto fijo de Banach y definiciones al respecto

El teorema de punto fijo de Banach, también llamado el principio de contracción, es desrito en Clapp, 2015, pág. 102 de la siguiente manera:

> Sea X un espacio métrico completo, no vacío, y sea ψ: $X \to X$ una contracción. Entonces se cumple lo siguiente:
>
> - ψ tiene un único punto fijo $x*$
> - Para cualquier $x_0 \in X$ la sucesión $(\psi^k(x_0))$ converge a $x*$ en X,
>
> y se cumple que:
>
> $$d(\psi^k(x_0), x*) \leq \frac{\alpha^k}{1-\alpha} d(\psi(x_0), x_0)), \tag{A.1}$$
>
> donde $\alpha \in (0, 1)$ satisface la ecuación A.5.

Particularmente, usando únicamente números reales, el teorema anterior se puede expresar al generalizar lo descrito en Henrici, 1964, de la manera presentada a continuación:

Sea R una región tal que $a_i \leq x_i \leq b_i \ \forall a_i, x_i, b_i \in \mathbb{R}$ con el subíndice $i = 1, 2, ...$ Suponga que las funciones g_i satisfacen las condiciones siguientes:

- g_i están definidas y son continuas en R.
- Para cada $\mathbf{X} \in R$, el punto $\mathbf{G}(\mathbf{X})$ también cae en R.
- Existe una constante $L < 1$ tal que para cualesquiera dos puntos $\mathbf{X}_1$ y $\mathbf{X}_2$ en R la desigualdad A.2 (condición Lipschitz) se cumple:

$$||\mathbf{G}(\mathbf{X}_1) - \mathbf{G}(\mathbf{X}_2)|| \leq L||\mathbf{X}_1 - \mathbf{X}_2||. \tag{A.2}$$

Entonces las siguientes afirmaciones son verdaderas:

- La ecuación 2.3 tiene precisamente una solución $\mathbf{S}$ en R.

- Para cualquier elección de $\mathbf{X}^0$ en R, la secuencia $\mathbf{X^{k+1}}$ dada por la ecuación 2.4 está definida y converge a $\mathbf{X}$.
- Para cualquier $k = 1, 2, ...$ la inecuación A.3 se cumple.

$$||\mathbf{X}^k - \mathbf{s}|| \leq \frac{L_n}{1-L}||\mathbf{X}^1 - \mathbf{X}^0|| \tag{A.3}$$

Espacio métrico: Conjunto *provisto de una métrica* Clapp, 2015.

Métrica: También llamada distancia, es una función $d : X \times X \to \mathbb{R}$ (donde X es un conjunto) que, dados $x, y \in X$, *tiene las siguientes 3 propiedades:*

- $d(x, y) = 0$ si y sólo si $x = y$.
- $d(x, y) = d(y, x)\ \forall x, y \in X$.
- La desigualdad del triángulo: $d(x, z) \leq d(x, y) + d(y, z)\ \forall x, y, z \in X$.

Clapp, 2015, pág. 8.

Espacio métrico completo: En Clapp, 2015 se define que un espacio métrico X es completo, *si toda sucesión de Cauchy en X converge en X*. Ejemplo: el espacio $\mathbb{R}$.

Sucesión de Cauchy: Es aquella definida en un espacio métrico X de modo que dada $\epsilon > 0$, existe $k_0 \in \mathbb{N}$ tal que

$$d_X(x_k, x_j) < \epsilon \quad \forall k, j \geq k_0 \tag{A.4}$$

(d_X denota una métrica en X) Clapp, 2015.

Contracción: Es una función $\phi : X \to X$ (donde X es un espacio métrico) tal que:

$$d(\phi(x), \phi(y)) \leq \alpha d(x, y)\ \forall x, y \in X \tag{A.5}$$

con $\alpha \in (0, 1)$ Clapp, 2015.

Apéndice B

Otros métodos de integración de ecuaciones diferenciales no lineales

B.1. Métodos de Runge-Kutta

Los métodos de Runge-Kutta son una familia de algoritmos de orden s con la forma de las ecuaciones B.1 y B.2 (Mastorakis & Sakellaris, 2009).

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i \tag{B.1}$$

donde

$$k_i = f\Big(x_n + c_i h, y_n + h \sum_{j=1}^{s} a_{ij} k_j\Big), \qquad i = 1, 2, ..., s \tag{B.2}$$

Estos métodos son utilizados para resolver problemas de valor inicial para un sistema de ecuaciones diferenciales ordinarias de primer orden, representado por la ecuación B.3 (Neumann, 2001):

$$\mathbf{X}' = \mathbf{f}(t, \mathbf{X}) \tag{B.3}$$

donde $\mathbf{f}$ representa las n funciones que describen al vector de derivadas $\mathbf{X}$, t denota la variable independiente respecto a la cual se ha derivado y $\mathbf{X}$ es un vector con n variables. La ecuación B.4 contiene n elementos, donde cualquier *j-ésimo* elemento tiene la forma:

$$x'_j = f_j(t, x_j) \tag{B.4}$$

Estos métodos están pensados "en imitar las expansiones de Taylor pero usando solo evaluaciones de la función" $f_j(t, x_j)$ (Flores, 2010)).

$$x_{j,n+1} = x_{j,n} + hf(t, x_{j,n}) + \frac{h^2}{2!} f'(t, x_{j,n}) + \frac{h^3}{3!} f''(t, x_{j,n}) \tag{B.5}$$

Entre esta famila se tienen los métodos de Runge-Kutta de segundo orden (método de Runge-Kutta simplificado) (Nieves & Domínguez, 2014). En la ecuación B.6 se tiene una versión multivariable de tal método.

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{2}(\mathbf{k}_0 + \mathbf{k}_1) \tag{B.6}$$

donde

$$\begin{aligned} \mathbf{k}_0 &= \mathbf{f}(\mathbf{x}_n) \\ \mathbf{k}_1 &= \mathbf{f}(\mathbf{x}_n + h\mathbf{k}_0) \end{aligned}$$

También se tienen los Método de Runge-Kutta de cuarto orden, entre los que destaca el método de Runge-Kutta clásico (Nieves y Domínguez, 2014 y Neumann, 2001). Su versión multivariable se muestra en la ecuación B.7.

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \tag{B.7}$$

donde

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{f}(\mathbf{x}_n) \\ \mathbf{k}_2 &= \mathbf{f}(\mathbf{x}_n + \frac{h}{2}\mathbf{k}_1) \\ \mathbf{k}_3 &= \mathbf{f}(\mathbf{x}_n + \frac{h}{2}\mathbf{k}_2) \\ \mathbf{k}_4 &= \mathbf{f}(\mathbf{x}_n + h\mathbf{k}_3) \end{aligned}$$

Apéndice C

Métodos de optimización de búsqueda directa

En los métodos de búsquda directa, sus restricciones son manejadas de una manera explícita y no requieren de la información del gradiente de la función objetivo (Rabat y col., 1985 y Baeyens y col., 2016).

C.1. Programación Lineal

Dentro de los modelos de investigación de operaciones se tiene el método de programación lineal, el cual es un método de optimización basado en la resolución de ecuaciones lineales, donde el óptimo a buscar es una combinación lineal de variables independientes sujeto a restricciones definidas por relaciones lineales de igualdad o desigualdad (Gerald y Wheatley, 2004 y Castillo y col., 2002).

Taha (2012) afirma que este método, así como todos los los modelos de investigación de operaciones, "consta de tres componentes básicos:

1. Las **variables** de decisión que pretendemos determinar.
2. El **objetivo** (la meta) que necesitamos optimizar (maximizar o minimizar).
3. Las **restricciones** que la solución debe satisfacer.

Para aplicar el método de programación lineal se define correctamente las variables de decisión, luego se construye la función objetivo (ecuación C.1), aquella que se quiere optimizar (minimizar o maximizar lo más posible), la cual debe ser lineal, tener n variables y estar sujeta a un número finito de restricciones lineales: k restricciones de igualdad y l restricciones de desigualdad (Taha, 2012 Y Karloff, 1961). De acuerdo a Taha (2012) deben ser conocidos con certeza todos los parámetros del modelo: *los coeficientes de las funciones objetivo y de restricción.*

Matemáticamente, el problema se plantea como la minimización de la función lineal:

$$F(\mathbf{X}) = \mathbf{c}^T\mathbf{X} \tag{C.1}$$

sujeta a las restricciones

$$A\mathbf{X} = \mathbf{b} \tag{C.2}$$
$$A'\mathbf{X} \geq \mathbf{b}' \tag{C.3}$$

donde $\mathbf{X}, \mathbf{c} \in \mathbb{R}^n$, A es una matriz de $k \times n$ elementos, $b \in \mathbf{R}^k$, A' es una matriz de $l \times n$ celdas y $b' \in \mathbb{R}^l$ [Karloff, 1961]. En la descripción encontrada en Karloff, 1961, se tiene que cada elemento del vector $\mathbf{X}$ es representado con x_k para algún valor de $k \leq n$ donde $k, n \in \mathbb{N}$. Los elementos son $x_k \geq 0$ si $k = 1, 2, ..., r$; los elementos con $k = r+1, r+2, ..., n$ obtienen cualquier valor de $\mathbb{R}$ (Karloff, 1961).

Se tienen dos formas especiales de programación lineal descritas en Karloff (1961): la forma estándar y la canónica. La forma estándar consiste en minimizar la ecuación C.1 sujeta a

$$A\mathbf{X} = \mathbf{b} \tag{C.4}$$

con $\mathbf{X} \geq \mathbf{0}$. La forma canónica también consiste en minimizar a la ecuación C.1 pero sujeto a la desigualdad

$$A\mathbf{X} \geq \mathbf{b} \tag{C.5}$$

con $\mathbf{X} \geq \mathbf{0}$.

Las 3 maneras de plantear un problema de programación lineal son equivalantes, por lo que cualquier problema representado en alguna de estas formas se puede configurar en cualquiera de las otras representaciones (Karloff, 1961) a través de manipulación algebraica, como el uso de variables de holgura (Belaire, 2009).

C.2. Simplex

Éste es un método para resolver problemas de programación lineal utilizando el algoritmo simplex sobre dos fases: con la primera se obtiene una solución factible si es que existe; en la segunda se consigue una solución óptima, si es que hay alguna, o se alcanza una clase de soluciones en la que los valores objetivos tienden a infinito (Dantzig & Thapa, 1997). Primero se escoge una esquina y se inspecciona, luego se elige la esquina contigua que incremente más la función objetivo, entonces se repite el proceso hasta que no se encuentre una mejor solución al problema de programación lineal (James & Dyke, 2018). Este método solicita que las ecuaciones estén escritas en un modo estándar, por lo que el primer paso es introducir variables de holgura o de exceso para transformar las restricciones de desigualdad en restricciones de igualdad.

Programa lineal en su forma estándar:

$$\begin{array}{ccccccccc} c_1x_1 & + & c_2x_2 & + & \dots & + & c_nx_n & = & z(Min) \\ a_{11}x_1 & + & a_{12}x_2 & + & \dots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \dots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & \vdots & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \dots & + & a_{mn}x_n & = & b_m \end{array} \tag{C.6}$$

con $x_1, x_2, \ldots, x_n \geq 0$.

La notación matricial de un programa lineal en su forma estándar es:

$$\begin{array}{rcl} \mathbf{c}^T\mathbf{X} & = & z \\ A\mathbf{X} & = & \mathbf{b} \end{array} \tag{C.7}$$

con $A : m \times n$, $\mathbf{X} \geq \mathbf{0}$

El primer paso del método es introducir variables artificiales a la forma estándar del problema lineal de tal manera que el problema se pueda llevar a la forma canónica. Las variables artificiales son variables no negativas que se deben de quitar de la base factible inicial de la fase 1 del método simplex. La función objetivo z es reemplazada por una nueva, w, la cual se conforma con la suma de las variables artificiales.

Luego, el problema debe cambiarse a la forma canónica factible. Una manera de lograrlo es restarle a la función objetivo w las restricciones del problema. Ahora es posible inicializar la Fase I del método, en la que es utilizado el algortimo simplex para que a través de consecutivas opereaciones de pivote se produzca una sucesión de diferentes formas canónicas, de tal manera que por cada iteración la función objetivo w decrezca hasta ser 0. Entonces se obtiene una solución factible básica (si es que existe) del problema original. Si $w > 0$ al final de esta fase, entonces no existe solución factible para el problema original.

En el algortimo simplex se requiere elegir la variable pivote, a partir de la cual se realizarán las operaciones de pivote (operaciones tipo Gauss-Jordan). El elemento pivote corresponde a la intersección de la columna de la variable de entrada con la fila de la variable de salida. La variable de entrada es la que tiene el coeficiente $c_s < 0$ donde $c_s = \min c_j$. La variable de salida tiene el índice i tal que se cumpla con la inecuación C.8.

$$\min_{\{i \ | a_{is} > 0\}} \ \frac{b_i}{a_{is}} \geq 0 \tag{C.8}$$

Para aplicar la Fase II es necesario retirar las variables artificiales del problema lineal. Esta fase corresponde a aplicar el alogritmo simplex a la forma canónica factible (obtenida con la Fase I) para minimizar el valor de la función objetivo z o generar un conjunto de soluciones en las que el valor de z tienda a $-\infty$.

La introducción de las variables artificiales llevan el PL (sistema de ecuaciones C.6) a la forma mostrada en la ecuación C.9 :

$$\begin{array}{ccccccccccc} x_{n+1} & + & x_{n+2} & + & \ldots & + & x_{n+m-1} & x_{n+m} & = & w \\ a_{11}x_1 & + & a_{12}x_2 & + & \ldots & + & a_{1n}x_n & x_{n+1} & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \ldots & + & a_{2n}x_n & x_{n+2} & = & b_2 \\ \vdots & & \vdots & & \ldots & & \vdots & \vdots & = & \vdots \\ a_{j1}x_1 & + & a_{j2}x_2 & + & \ldots & + & a_{jn}x_n & x_{n+j} & = & b_j \\ \vdots & & \vdots & & \ldots & & \vdots & \vdots & = & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \ldots & + & a_{mn}x_n & x_{n+m} & = & b_m \end{array} \tag{C.9}$$

Restando w en ambos lados de la primera igualdad del sistema de ecuaciones C.9, el PL se

puede expresar de manera matricial con el sistema de ecuaciones C.10.

$$\begin{aligned} \mathbf{e}^T\mathbf{X}_a &= w \\ A\mathbf{X} + I\mathbf{X}_a &= \mathbf{b} \end{aligned} \tag{C.10}$$

con $\mathbf{X}_a = (x_{n+1}, x_{n+2}, ..., x_{n+m}) \geq \mathbf{0}$, además $\mathbf{e} = (1, 1, ..., 1)^T$.

C.2.1. Tabulación

El método tabular es otra versión del método simplex, en el cual el sistema de ecuaciones es escrito en orden de manera tabular (James & Dyke, 2018). Las variables básicas y la variable de la función objetivo son colocadas en la columna izquierda de una tabla; en la primera fila se ponen las etiquetas de las variables no básicas, de las básicas y al final la etiqueta "solución"; debajo de cada etiqueta se tiene su correspondiente columna con los coeficientes correspondientes a las variables de la etiqueta. Las operaciones que son utilizadas en el método tabular son similares a las usadas en el método simplex.

Apéndice D

Otros métodos de búsqueda indirecta

D.1. Métodos de 2do. orden

D.1.1. Newton

El método de Newton-Raphson para solucionar sistemas de ecuaciones no lineales, visto en la sección previa, también se puede aplicar al problema de hallar un mínimo o un máximo, en lugar de buscar los valores de $\mathbf{X}$ tal que el sistema de ecuaciones sea $\mathbf{0}$, lo que se hace es buscar los valores de $\mathbf{X}$ que hagan 0 a la derivada de la función a optimizar (Gerald & Wheatley, 2004). Este método de optimización utiliza iterativamente la ecuación D.1 para aproximarse a la función escalar $f(\mathbf{X} = 0)$, donde el vector de longitudes de paso $\mathbf{h}$, el vector gradiente $\mathbf{G}$ y la matriz Hessiana $\mathbf{J}$ son variables descritas por las ecuaciones D.2, D.3 y D.4, respectivamente (James & Dyke, 2018).

$$f(a_1 + h_1, ..., a_n + h_n) \approx f(a_1, ..., a_n) + \mathbf{h}^T\mathbf{G} + \frac{1}{2}\mathbf{h}^T\mathbf{J}\mathbf{h} \tag{D.1}$$

$$\mathbf{h} = \begin{bmatrix} h_1 \\ \vdots \\ h_n \end{bmatrix} \tag{D.2}$$

$$\mathbf{G} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} \tag{D.3}$$

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1^2} \end{bmatrix} \tag{D.4}$$

A pesar de que los métodos de Newton tienen la ventaja de que convergen rápidamente, en James y Dyke (2018) se afirma que el método de Newton es muy poco confiable, en particular

para problemas de dimensiones superiores, a menos que el valor inicial se encuentre cerca del mínimo. El problema es que la condición que utiliza este método para encontrar el mínimo, también sirve para encontrar máximos y puntos de silla, encontrándose que el procedimiento termine dando un punto máximo o acercándose a un punto de silla para luego diverger, y en el caso multidimensional, los puntos de silla son abundantes [James y Dyke, 2018].

D.1.2. Cuasi-Newton

Los métodos cuasi-Newton son métodos de optimización que buscan la minimización de la función $f = (x_1, ..., x_n)$ que tiene el gradiente dado por el vector $\mathbf{G}$ iterando una matriz Hessiana H_i la cual es actualizada en cada iteración cumpliendo con la ecuación D.5 (James & Dyke, 2018).

$$\mathbf{X_{i+1}} = \mathbf{X_i} - \lambda H_i \mathbf{G}_i \tag{D.5}$$

En Cottle y Thapa (2017) se da la observación de que estos métodos comienzan típicamente con una aproximación de la matriz Hessiana inicial H_0 igual a la matriz identidad. Una vez que el nuevo vector $\mathbf{X}$ ha sido calculado, la matriz Hessiana H es actualizada, obteniéndose H_1 (Cottle & Thapa, 2017), y así sucesivamente se obtienen actualizaciones de H_i. La matriz Hessiana debe tener una forma tal que satisfaga a la ecuación D.6, donde $\mathbf{h}_i = \mathbf{X}_{i+1} - \mathbf{X}_i$ y $\mathbf{y}_i = \mathbf{G}_{i+1} - \mathbf{G}_i$ James y Dyke, 2018.

$$H_{i+1}\mathbf{y}_i = \mathbf{h}_i \tag{D.6}$$

Entre los métodos pertenecientes a esta familia se menciona en Cottle y Thapa (2017) el método DFP (Davidon, Fletcher y Powell) y el método de Broyden, descrito en Nieves y Domínguez (2014). La ecuación D.7 muestra la forma que tiene la matríz Hessiana correspondiente al método DFP (James & Dyke, 2018).

$$H_{i+1} = H_i - \frac{H_i \mathbf{y}_i \mathbf{y}_i^T H_i}{\mathbf{y}_i^T H_i \mathbf{y}_i} + \frac{\mathbf{h}_i \mathbf{h}_i^T}{\mathbf{h}_i^T \mathbf{y}_i} \tag{D.7}$$

James y Dyke (2018) afirma que la familia de métodos cuasi-Newton es aún una de las más utilizadas y fiables aunque tiene la desventaja de ser programas muy largos y tediosos de escribir ya que se requiere mucho trabajo en verificación y correcciones para que el programa no se detenga.

Apéndice E

Códigos

Los códigos de los programas para el proceso de diseño se muestran a continuación. Los códigos tienen ciclos de instrucciones, pero algunas secciones de instrucciones fueron escritas de manera explícita en vez de estar dentro de ciclos (bucles), esto debido a la intención de tener mayor claridad al momento de analizar la estructura matemática usada y estar seguro de obtener los resultados correctos. También es importante aclarar que para hacer las mediciones de tiempo se tienen que excluir del código las líneas que tienen que ver con la generación de figuras y con la recolección de datos, incluyendo los vectores de datos.

E.1. Código para diseño de circuito pasivo de 1 nodo

Se muestran el código utilizado para generar las figuras e información utilizadas para el análisis del circuito de 1 nodo.

De la línea 1 a la 7 se les llama a las librerías requeridas para el programa. En la 9 y 10 se colocan los valores máximo y mínimo a adquirir la condición inicial de la variable de estado x_2. La línea 11 indica la separación entre cada condición inicial de x_2. En las líneas 12 a la 16 se definen los valores para el parámetro de escala, el parámetro del criterio de convergencia de la MGD, el parámetro de comparación del criterio de convergencia del MNR y los diferenciales dx_1 y dx_2. En la línea 18 se define el valor deseado de diseño para x_2; en 19, el voltaje de entrada; en 20 y 21, los coeficientes de la función que define a la resistencia no lineal R_n. En 22 se define una variable que indica el valor máximo que pueden adquirir los vectores para almacenar datos. Luego se declaran un conjunto de variables y funciones que serán utilizadas. La línea 44 es utilizada para abrir la ventana de comandos de MATLAB. Del renglón 46 al 124 se tiene el programa principal, en el cual primero se declara el valor de la variable de control en la línea 48: si $u = 0$ entonces se inicia a trabajar con la ETD y en caso contrario se comienza con la ETDM. En 49 se tiene la variable *conpco* que si le es asignado el valor de 1, entonces el programa trabajará con pasos de conmutación, mientras que con 0 trabajará únicamente con una de las 2^M estrategias básicas de diseño. Con Nci (línea 51), se determina el número de condiciones iniciales con las que se trabajará, o sea, se generarán Nci trayectorias de diseño. La variable K contiene el número de variables de estado independientes. La función *engEvalString*

permite escribir código para que MATLAB lo procese, así en la línea 58, se está limpiando el espacio de trabajo de MATLAB e inicianizándose la función g que es declarada y utilizada más adelante para graficar las trayectorias. De los renglones 58 al 61 se tiene que si no hay paso de conmutación, entonces el programa hará únicamente 1 trayectoria, de lo contrario, podrá generarse al menos 100 distintas trayectorias con una sola puesta en marcha del programa; esta parte es útil cuando se quiere analizar distintos valores del punto de conmutación. En caso de haber habilitado la conmutación, los renglones 61 al 66 abrirán una interfaz de usuario, solicitándole el número de pasos usados por la primera estrategia de diseño antes de conmutar hacia la otra. En las líneas 71 y 72 se definen las variables de estado. Las variables *datos0[]* y *datos1[]*, en distintos puntos del código, guardan todos los valores de las variables de estado; éstas son usadas para que al completarse un solución de diseño, MATLAB pueda tomar esa información y graficar la trayectoria de ese proceso de diseño. De la línea 86 a la 90, se tiene indicado que se itera la función de la metodología general de diseño *MGD()*. En 115 y 117 se tienen funciones para generar las gráficas requeridas. La función para generar derivadas numéricas está desarrollada en las líneas 126-131. La función objetivo tradicional se define en 132-136. La ecuación que describe al circuito de 1 nodo se encuentra en 138-142. La derivada de la ecuación del circuito está en 144-149. El método de Newton-Raphson se desarrolla en 151-164. La función de penalidad está en 166-170. La función objetivo con la función de penalidad está en 172-176. El vector de dirección del movimiento **H** se encuentra definido desde el renglón 178 al 195. La metodología general de diseño se encuentra en 197-224.

```
1  #include <iostream>
2  #include <time.h>
3  #include <cmath>
4  #include<stdlib.h>
5  #include<fstream>
6  #include <stdio.h>
7  #include "engine.h"
8
9  #define     cimax 3
10 #define    cimin -3
11 #define    distanciaci 0.1
12 #define    paso 1*pow(10,-4)  //Parmetro de escala
13 #define    eps pow(10,-8)     //psilon, valor de comparacin
14 #define    power  pow(10.0,-8)
15 #define    dx1 pow(10,-5)       //delta x1
16 #define    dx2 pow(10,-5)        //delta x2
17
18 #define     Vd 0.2             //Valor deseado para X2
19 #define    V0 1             //Voltaje de entrada
20 #define    a 1              //Constante "a" de resistencia variable Rn
21 #define    b 0              //Constante "b" de resistencia variable Rn
22 #define    numdatos 20000000
23
24 using namespace std;
25
26 int it, Nci, ci; //Iteracin, //Nmero de condiciones iniciales,
27 //condicin inicial
28 int K, tiempo, u,pco, conpco, iteraciones, contador, mcontador,
29 estados, pasos;
30 double Dg;
31 double datos0[numdatos], datos1[numdatos], deltaX2, Xn, Xna, h1, h2;
32 clock_t T;
33 double X[2];
34 mxArray *x_array, *y_array, *nci_array, *it_array, *conmutacion_array,
35  *cimax_array, *cimin_array, *distanciaci_array;
36 double *px, *py, *pne, *pit, *pconmutacion, *pcimax, *pcimin,
37 *pdistanciaci;
38
39 double derivada(double, double, double), C(double), g(double, double),
40 dgdx(double, double, double);
41 double MN(double, double), Phi(double, double), F(double, double);
42 void H(double, double), MGD(), graficar(), graficar_2(),
43 graficar_1a1();
44 Engine *ep = engOpen(NULL);
45
46 int main()
47 {
```

```
    u=1;                //Elige estrategia
    conpco=0*u;         //Si conpco=1, entonces hay conmutacin. Si conpco=0, entonces no hay conmutacin

    Nci=1+(cimax-cimin)/distanciaci;//Nmero de condiciones iniciales
    K=u+1;
    if (!(ep = engOpen(""))) {
        fprintf(stderr, "\nCan't start MATLAB engine\n");
        return EXIT_FAILURE;
    }

    engEvalString(ep, "clear all; g=0;");

    if (conpco==0) mcontador= 1;
    else mcontador=100;

    for (contador = 0; contador < mcontador; contador=contador+1)
    {
         if (conpco==1)
        {
            cout << "PASO DE CONMUTACION"<< endl;
            cin >> pco;             //Paso de conmutacin
            u=1;
        }

        for (ci=0; ci<Nci; ci=ci+1)
        {
            engEvalString(ep, "clear x, y;");
            X[0]=0.3;               //Valores iniciales del vector X
             X[1]=cimax-(ci)*distanciaci;

               printf("X[1]=%f",X[1]);
            if(Nci<10)
            {
                printf("NUMERO DE CONDICION INICIAL : %d\n",ci+1);
                printf("Valor inicial X1: %f\n", X[0]);
                printf("Valor inicial X2: %f\n", X[1]);
                printf("# PASO DE CONMUTACION: %d\n", contador+1);
            }

             datos0[0]=X[0];      datos1[0]=X[1];

             T=clock();

            for (it = 0; it <= 1000000; it=it+1)
            {
                  if(pco<=it && conpco==1) u=0;
               MGD();
            }

            tiempo=clock()-T;

            if(it==1000001){
                printf("Error, no se ha alcanzado el valor "
                 "deseado\n");
                break;
            }

            if(u==1) it=iteraciones-1;
            else it=iteraciones;

            estados=it+2;
            pasos=it+1;

            {
                printf("Resultado: %f ---------- %f\n",
               X[0], X[1]);
                printf("Iteraciones: %d\n",it);
                printf("Tiempo: %f\n\n", (double)tiempo);
            }
            engEvalString(ep, "line(x(2,1), y(2,1), 'Marker', "
            "'x', 'LineWidth',2.5, 'MarkerSize',20, 'MarkerEdgeColor',"
            "[0,0,0]);");
            graficar();
        }
        if (conpco==0) graficar_2();
    }
}

double derivada(double f2, double f1, double dx)
{
      double Df = f2-f1;
      double dfdx= Df/dx;
      return dfdx;
}
double C(double x2)
{
    double c=pow(x2-Vd,2);
    return c;
```

```
}

double g(double x1, double x2)
{
    double A= (a+b*pow(x2-V0,2)+pow(x1,2))*x2-pow(x1,2)*V0;
   return A;
}

double dgdx(double X0, double X1, double dx)
{
   Dg=g(X0,X1+dx)-g(X0,X1);     //Delta g(X)
    double dgdx=Dg/dx;           //Derivada de la funcin g(X)
   return dgdx;
}

double MN(double x1, double x2)
{
   for(int j=0; j<100; j++)
    {
        double x2a=x2;
        double g1= x1*x1 + a + b*(x2-1)*(3*x2-1);
        x2=x2-g(x1,x2)/g1;
      double absx2_x2a=fabs(x2-x2a);
        if(absx2_x2a<= power)break;
        if (j==99) printf("REVISA el MN \n!!!!!!");
    }
    X[1]=x2;
    return x2;
}

double Phi(double x1, double x2)
{
    double B= pow(g(x1, x2),2);
    return B;
}

 double F(double x1, double x2)
{
   double FF = C(x2) + u*Phi(x1, x2);
   return FF;
}

void H(double x1, double x2)
{
    if(u==0)
    {
      double x2a=x2;
      x2=MN(x1+dx1,x2);

      deltaX2=(x2-x2a);
       h1= -derivada(F(x1, x2a+dx2),
      F(x1,x2a), dx2)*derivada(x2, x2a, dx1);
   }

   else if (u==1)
   {
      h1= -derivada(F(x1+dx1,x2), F(x1,x2), dx1);
      h2=   -derivada(F(x1,x2+dx2), F(x1,x2), dx2);
   }
}

void MGD()
{
   if (u==0)
   {
      MN(X[0], X[1]);
      datos0[it+1]=X[0];
      datos1[it+1]=X[1];
   }

   if(eps<fabs(F(X[0],X[1])))

    {
       H(X[0],X[1]);
       double h[2]={h1,h2};
         for (int i=0; i<K; i++)
         {
            X[i]=X[i] + paso*h[i];
       }
         if (u==0) datos0[it+2]=X[0];
         if (u==1) {datos0[it+1]=X[0];    datos1[it+1]=X[1];}

    }
    else
   {
      iteraciones=it; //iteraciones=it-1;
      it=6000000;
   }
```

```
224 }
```

La función *graficar()*, que va de la línea 225 a la 321, manda la información desde DevC++ a MATLAB y genera las gráficas utilizadas en el análisis. Desde el renglón 227 al 234 se crean las matrices tipo mxArray; previamente, en las líneas 34-37 fueron declarados punteros tipo *mxArray* y otros punteros tipo *double* que son utilizados para transferir la información entre ambos softwares. En las líneas 258-265 se crean variables en el espacio de trabajo de MATLAB y se guarda en ellas la información proveniente del programa ejecutado en DevC++. Las líneas 267-268 contienen instrucciones de decisión a cerca de crear la figura del retrato de fase en caso de no haber puntos de conmutación; en caso contrario se genera una figura con dos gráficas, la primera con el retrato de fase y la segunda con la comparación del número de iteración vs. el punto de inicio. De la línea 270 a la 312 se realiza lo necesario para generar las figuras de manera correcta. Desde el renglón 314 al 320 se libera memoria. Con la función $grafica r\_2()$ se genera una figura con una gráfica correspondiente al número de pasos vs. la condición inicial. La función $grafica r\_1a1$, traza las curvas punto a punto en vez de esperar a terminar el proceso de diseño; esto hace más lenta la ejecución del programa pero en algunos casos es útil.

```
225 void graficar()
226 {
227    x_array = mxCreateDoubleMatrix(it+2,1,mxREAL);
228    y_array = mxCreateDoubleMatrix(it+2,1,mxREAL);
229    nci_array = mxCreateDoubleMatrix(1,1,mxREAL);
230    it_array = mxCreateDoubleMatrix(1,1,mxREAL);
231    conmutacion_array = mxCreateDoubleMatrix(1,1,mxREAL);
232    cimax_array = mxCreateDoubleMatrix(1,1,mxREAL);
233    cimin_array = mxCreateDoubleMatrix(1,1,mxREAL);
234    distanciaci_array = mxCreateDoubleMatrix(1,1,mxREAL);
235
236    px = mxGetPr(x_array);
237    py = mxGetPr(y_array);
238    pne = mxGetPr(nci_array);
239    pit = mxGetPr(it_array);
240    pconmutacion = mxGetPr(conmutacion_array);
241    pcimax= mxGetPr(cimax_array);
242    pcimin= mxGetPr(cimin_array);
243    pdistanciaci= mxGetPr(distanciaci_array);
244
245    for(int k=0;k<=it+1;k++)
246    {
247       px[k]= datos0[k];
248       py[k]= datos1[k];
249    }
250
251    pne[0]= ci;
252    pit[0]=it;
253    pcimax[0]= cimax;
254    pcimin[0]= cimin;
255    pdistanciaci[0]= distanciaci;
256    pconmutacion[0]=contador+1;
257
258     engPutVariable(ep,"x",x_array);
259    engPutVariable(ep,"y",y_array);
260     engPutVariable(ep,"nci",nci_array);
261    engPutVariable(ep,"it",it_array);
262    engPutVariable(ep,"conmutacion",conmutacion_array);
263    engPutVariable(ep,"cimin",cimin_array);
264    engPutVariable(ep,"cimax",cimax_array);
265    engPutVariable(ep,"distanciaci",distanciaci_array);
266
267    if (conpco==0) engEvalString(ep, "figure (1);");
268    else engEvalString(ep, "subplot(2,1,1);;");
269
270    if(1+ci==1)    //LO hace ms rpido o lento?
271    {
272       engEvalString(ep, "line([-5,5],[0,0],'LineWidth',1,"
273       "'color','black');");
274       engEvalString(ep, "line([0,0],[-5,5],'LineWidth',1,"
275       "'color','black');");
276       engEvalString(ep, "title('Retrato de fase: x_2 vs. x_1',"
277       "'FontSize',28);");
278       engEvalString(ep, "xlabel('x_1','FontSize',24);");
279       engEvalString(ep, "ylabel('x_2','FontSize',24);");
280       engEvalString(ep, "grid on;");
```

```
281     }
282
283      engEvalString(ep, "line(x(it+2,1),y(it+2,1),'Marker','x',"
284     "'LineWidth',2.5,"
285     "'MarkerSize',20,'MarkerEdgeColor',[0,0,0]);");
286      engEvalString(ep, "line(x(1,1),y(1,1),'color',[1,1,1],"
287     "'Marker','o',"
288     "'LineWidth',1,'MarkerSize',8,'MarkerEdgeColor',[0,0,0]);");
289     engEvalString(ep, "g=line(x,y,'LineWidth',2.5,'Color',"
290     "[0.5,0.5,0.5],'LineStyle',"
291     "'-');");
292     engEvalString(ep, "set(gca,'fontsize',16)");
293
294     if (conpco==0) engEvalString(ep, "itera(nci+1)=it;"
295     "nconmu(conmutacion)=conmutacion;");
296
297     else if  (conpco==1)
298     {
299        engEvalString(ep, "nconmu(conmutacion)=conmutacion;"
300        "itera(conmutacion)=it;");
301        engEvalString(ep, "subplot(2,1,2);;");
302        engEvalString(ep, "line(nconmu,itera,'color','black');");
303        engEvalString(ep, "line(conmutacion,it,'color','black',"
304        "'Marker','.',"
305        "'MarkerSize',20,'MarkerEdgeColor','black');");
306        engEvalString(ep, "title('Nmero de pasos vs. nmero de"
307        "conmutacin','FontSize',28);");
308        engEvalString(ep, "xlabel('Nmero de conmutacin',"
309        "'FontSize',24);");
310        engEvalString(ep, "ylabel('Nmero de pasos','FontSize',24)");
311         engEvalString(ep, "set(gca,'fontsize',16)");
312     }
313
314     mxDestroyArray(x_array);
315     mxDestroyArray(y_array);
316     mxDestroyArray(nci_array);
317     mxDestroyArray(it_array);
318     mxDestroyArray(conmutacion_array);
319     mxDestroyArray(cimin_array);
320     mxDestroyArray(cimax_array);
321 }
322
323 void graficar_2()
324 {
325     engEvalString(ep, "figure(2);");
326     engEvalString(ep, "line(cimax:-distanciaci:cimin,itera,"
327     "'FontSize',28);");
328     engEvalString(ep, "line(cimax:-distanciaci:cimin,itera,"
329     "'color','black',"
330     "'Marker','.','MarkerSize',20,'MarkerEdgeColor','black');");
331     engEvalString(ep, "title('Nmero de pasos vs. condicin"
332     "inicial para x_2','FontSize',28);");
333     engEvalString(ep, "xlabel('Condicin inicial para x_2',"
334     "'FontSize',24);");
335     engEvalString(ep, "ylabel('Nmero de pasos','FontSize',24);"
336     "set(gca,'fontsize',16)");
337 }
338
339 void graficar_1a1()
340 {
341     x_array = mxCreateDoubleMatrix(2,1,mxREAL);
342     y_array = mxCreateDoubleMatrix(2,1,mxREAL);
343     nci_array = mxCreateDoubleMatrix(1,1,mxREAL);
344     it_array = mxCreateDoubleMatrix(2,1,mxREAL);
345     conmutacion_array = mxCreateDoubleMatrix(1,1,mxREAL);
346     cimax_array = mxCreateDoubleMatrix(1,1,mxREAL);
347     cimin_array = mxCreateDoubleMatrix(1,1,mxREAL);
348     distanciaci_array = mxCreateDoubleMatrix(1,1,mxREAL);
349
350     px = mxGetPr(x_array);
351     py = mxGetPr(y_array);
352     pne = mxGetPr(nci_array);
353     pit = mxGetPr(it_array);
354     pconmutacion = mxGetPr(conmutacion_array);
355     pcimax= mxGetPr(cimax_array);
356     pcimin= mxGetPr(cimin_array);
357     pdistanciaci= mxGetPr(distanciaci_array);
358
359     for(int k=0;k<=1;k++)
360     {
361        px[k]= datos0[it+k];
362        py[k]= datos1[it+k];
363        pit[k]= it+k;
364     }
365
366     pne[0]= ci;
367     pcimax[0]= cimax;
368     pcimin[0]= cimin;
```

```
369     pdistanciaci[0]= distanciaci;
370     pconmutacion[0]=contador+1;
371
372      engPutVariable(ep,"x",x_array);
373     engPutVariable(ep,"y",y_array);
374      engPutVariable(ep,"nci",nci_array);
375     engPutVariable(ep,"it",it_array);
376     engPutVariable(ep,"conmutacion",conmutacion_array);
377     engPutVariable(ep,"cimin",cimin_array);
378     engPutVariable(ep,"cimax",cimax_array);
379     engPutVariable(ep,"distanciaci",distanciaci_array);
380
381
382     if(1+ci==1)
383     {
384        engEvalString(ep, "line([-5,5],[0,0],'LineWidth',1,"
385        "'color','black');");
386        engEvalString(ep, "line([0,0],[-5,5],'LineWidth',1,"
387        "'color','black');");
388        engEvalString(ep, "title('Retrato de fase: x_2 vs. x_1',"
389        "'FontSize',28);");
390        engEvalString(ep, "xlabel('x_1','FontSize',28);");
391        engEvalString(ep, "ylabel('x_2','FontSize',28);");
392        engEvalString(ep, "grid on;");
393     }
394
395     engEvalString(ep, "figure(1);");
396      if (it==0) engEvalString(ep, "line(x(1,1),y(1,1),'Marker','o','LineWidth',2.5,'MarkerSize',6,'
           MarkerEdgeColor',[0,0,0]);");
397     engEvalString(ep, "g=line(x,y,'LineWidth',3,'Color',[0,0,0],'LineStyle','-');");
398     engEvalString(ep, "hold on;");
399
400     engEvalString(ep, "figure(2);");
401     engEvalString(ep, "g=line(it,y,'LineWidth',1,'Color',[0,0,0],'LineStyle','-');");
402     engEvalString(ep, "hold on;");
403
404     mxDestroyArray(x_array);
405     mxDestroyArray(y_array);
406     mxDestroyArray(nci_array);
407     mxDestroyArray(it_array);
408     mxDestroyArray(conmutacion_array);
409     mxDestroyArray(cimin_array);
410     mxDestroyArray(cimax_array);
411 }
```

E.2. Código para diseño de circuito pasivo de 2 nodos

En esta sección se presenta el código utilizado para la generación de las figuras e información utilizadas en el análisis del circuito de 2 nodos. La descripción es similar a la realizada para el código del circuito de 1 nodo, por lo que se realizará de manera más resumida, excepto en algunas líneas claves del código.

De la línea 1 a la 7 se incluyen las librerías necesarias. Desde la línea 9 hasta la 57 se declaran variables, funciones y se tiene la instrucción de abrir la ventana de comandos de MATLAB.

Este código tiene la ventaja de poder reducir el número de datos sin sacrificar información importante, esto con la finalidad de evitar en lo posible sobrepasar los límites de la memoria. Se inicializan variables de iteración *id[ ]* en la línea 62, con la finalidad de ser capaz de tener diferente número de datos grabados para cada variable. En 63 y 64 se define la estrategia de diseño a utilizarse en un inicio. A la variable *menos_puntos* de la línea 67, se le asigna un valor de *true* cuando se desea tener menos información guardada. En el renglón 74 se le asigna un 1 a *juntas* si lo que se desea es juntar en una misma figura, la proyección del retrato de fase correspondiente a las 4 estrategias de diseño básicas. En 78, se le asigna a la í-esima *G[ ]* el valor de 1 si lo que se desea es conseguir la i-ésima gráfica, cada una representando una proyección del retrato de fase; en caso contrario se debe asignar un 0. El vector de estados es definido en las líneas 95-99. En 105 se le llama a una función que crea el vector donde se

grabarán los datos necesarios de las variables de estado. La metodología general de diseño es utilizada iterativamente en el renglón 118. La sentencia de decisión de la línea 120 ejecuta las instrucciones de los renglones 122-162 en tanto no se haya llegado al paso de conmutación dado en pco (esto se revisa mediante *(it+1) %gato*) y mientras el número de iteraciones siga siendo menor o igual al límite permitido por la variable *itmax*; tales instrucciones corresponden a uno de los pasos necesarios para poder mandar la información correctamente a MATLAB y entonces poder graficar los datos que se tengan hasta el momento; también se solicita el ingreso por parte del usuario de un nuevo tamaño del parámetro de escala y un nuevo número de iteraciones siguientes, si es que aún no se ha encontrado la solución al proceso de diseño. En la línea 161 se llama a la función que crea un nuevo vector para grabar datos de las variables de estado. En 162 se asignan los valores iniciales del vector de estados.

Después, con las líneas 191-194 se llama a las funciones encargadas de terminar de graficar las proyecciones del retrato de fase. Luego, con las líneas 200-206 se envía información a la pantalla. Si se ha seleccionado la opción de tener punto de conmutación, entonces en el renglón 213 se llama a a otra gráfica.

De 219-224 se define la función que permite derivar. En 226-230 se tiene a la función objetivo tradicional. La función que describe a la admitancia no lineal, se encuentra de 232-236. Luego, desde 238-248 se describe el sistema de ecuaciones del circuito de 2 nodos. En 250-271, la función de penalidad está escrita. Por otro lado, la función objetivo total, se encuentra en las líneas 273-277. El método de Gauss-Jordan es descrito en los renglones 279-314. El método de Newton-Raphson, desde la línea 316 hasta la 379. El vector **H** es creado desde el renglón 381 hasta el 474, teniendo un comportamiento diferente para cada uno de las 4 estrategias de diseño básicas. La metodología general de diseño, se encuentra en las líneas 476-565.

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <stdlib.h>
4  #include <conio.h>
5  #include <iostream>
6  #include <time.h>
7  #include "engine.h"
8
9  #define    eps 1.0e-6      //psilon, valor de comparacin contra funcin objetivo
10 #define    power 1.0e-6
11 #define    dx1 1.0e-5      //delta x1
12 #define    dx2 1.0e-5      //delta x2
13 #define    dx3 1.0e-5      //delta x3
14 #define    dx4 1.0e-5      //delta x4
15 #define    dx5 1.0e-5      //delta x5
16
17 #define     Vd     0.2      //Valor deseado para X2
18 #define    V0     1.0      //Voltaje de entrada
19 #define    y_0    1.0      //Constante "a" de resistencia variable Rn
20 #define    an     0.7      //Constante "b" de resistencia variable Rn
21
22 #define    itmax  25000000
23 #define    N      5
24
25 using namespace std;
26
27 Engine *ep = engOpen(NULL);
28 int M, K;
29 int it;
30 int Nci;
31 int tiempo, u, u1, u2, pco, conpco, iteraciones, contador, U;
32 int estados, pasos, n;
33 int E[100];
34
35 double a[2][3], efe;
36 double b, x, dx, R, v1, v2;
37 double Dg, gp, deltaX2, h, Phin1, Phin2, h1, h2, h3, h4, h5;
38 double derivadaFx1, derivadaFx2, ts, dPhi1, dPhi2, deltaX1, cimax, cimin, distanciaci, VC0;
39 double *datos0, *datos1, *datos2, *datos3, *datos4;
40 double X[N];
```

```
41
42 double dx5x1, dx5x2, dx5x3, dx5x4, dx4x1, dx4x2, dx4x3, dx4x5;
43 double ynx, yny, fx, fy, gx, gy;
44 clock_t T;
45 int ci, mcontador, juntas;
46 int sumE;
47 int G[9], gato, SizeArray[5], corridas, id[5], base;
48 double paso, x_anterior[5];
49 bool menos_puntos;
50
51 double derivada(double, double, double), C(double), yn(double, double), g1(double, double, double, double, double
       );
52 double g2(double, double, double), Phi(double, double, double, double, double), F(double, double, double, double,
        double);
53 void GJ(), MNR(double, double, double, double, double), MGD();
54 void ejes(), graficar(), graficar_2(), cpptomathlab(), crear_vectores_datos(), asignar_datos(double [],int []),
       puntosiniciales(),
55    puntosfinales();
56 double distancia(double,double,double,double);
57 int data0, data1, data2, data3, data4, data5;
58
59 int main()
60 {
61     base=0;
62     id[0]=id[1]=id[2]=id[3]=id[4]=0;
63    double U[2]={0,0};                    //    <<<<<----------Elige estrategia
64    u1=U[0]; u2=U[1];
65
66     conpco=0;
67     menos_puntos = false;
68    paso= 1.0e-4;
69    if (conpco==0) gato=  10000000;    //corridas
70    cimax=4;
71    cimin=4;
72    distanciaci=0.2;
73    Nci=int(1.1+(cimax-cimin)/distanciaci);   //Nmero de condiciones iniciales
74    juntas=0;                                 //Para juntar grficas
75    sumE=0;
76    K= 3+u1+u2;     M= N-K;      n=M;
77    engEvalString(ep, "clear all");
78    G[0]=0; G[1]=0; G[2]=0; G[3]=0; G[4]=0; G[5]=1;
79    if (conpco==0) mcontador= 1;
80    else mcontador=100;
81     mcontador=1;
82
83    for (contador = 0; contador < mcontador; contador=contador+1)
84    {
85       if (conpco>=1)
86       {
87       cout << "PASO DE CONMUTACION"<< endl;
88       cin >> pco;  //Paso de conmutacin
89       gato=pco;    //corridas
90       u1=U[0]; u2=U[1]; n=M;
91       }
92       int ei=0; int ef;
93       for (ci=0; ci<Nci; ci=ci+1)
94       {
95          X[0]=1.2;  //fijo      //Valores iniciales del vector X      <-----------------------------------------
96           X[1]=1.2;  //fijo
97           X[2]=1.2;
98           X[3]=1.2;
99          X[4]=cimax-(ci)*distanciaci;
100
101         printf("Valores iniciales: (%.12f, %.12f, %.12f, %.12f, %.12f)\n",X[0],X[1],X[2],X[3],X[4]);
102         printf("# CONMUTACION: %d\n", contador+1);
103
104            corridas=0;
105         crear_vectores_datos();
106         it=-1;
107         int coo_inicial[]={0,0,0,0,0};
108         asignar_datos(X,coo_inicial);
109
110            for (int i=0; i<5; i++)     SizeArray[i]=1;
111
112         cpptomathlab();
113         puntosiniciales();
114
115         T=clock();
116         for (it = 0; it <=itmax; it=it+1)
117          {
118            MGD();
119            }
120            if ((it+1)%gato==0 && it<=itmax)
121            {
122               int it_id[5];
123               efe= F(X[0], X[1], X[2], X[3], X[4]);
124               printf("efe = %.10f, X0= %.10f, X1= %.10f, X2= %.10f, X3= %.10f, X4= %.10f, X5= %.10f\n",efe, X[0], X
                     [1], X[2], X[3], X[4]);  //Los datos no son los puntos grficos
```

```
                if (menos_puntos == true)
                {
                    for (int i=0; i<5; i++)
                    {
                        SizeArray[i]=id[i]+1;
                        it_id[i] = id[i];
                    }
                }
                else
                {
                    SizeArray[0]=gato+1;
                    it_id[0] = it_id[1] = it_id[2] = it_id[3]= it_id[4] = it+1;
                 }
                cpptomathlab();
                graficar();
                corridas= corridas + it + 1;
                printf("corridas = %d\n",corridas);

                double dd[5];
                if  (G[0]==1 || G[3]==1)    dd[0]=datos0[it_id[0]];
                if (G[1]==1 || G[4]==1) dd[1]=datos1[it_id[1]];
                if (G[2]==1 || G[5]==1) dd[2]=datos2[it_id[2]];
                if (G[0]==1 || G[1]==1 || G[2]==1)  dd[3]=datos3[it_id[3]];
                if (G[3]==1 || G[4]==1 || G[5]==1)  dd[4]=datos4[it_id[4]];

                if(pco<=it+1 && conpco>0) {u1=0; u2=0; n= 2; conpco=0;}
                it=-1;
                for (int i=0; i<5; i++) id[i]=0;

                cout << "Ingresar tamao de parmetro de escala"<< endl;
                cin >> paso;
                cout << "Ingresar nmero de iteraciones siguientes"<< endl;
                cin >> gato;
                engEvalString(ep, "clear all");

                crear_vectores_datos();
                asignar_datos(dd,coo_inicial);
            }
        }
        tiempo=clock()-T;

        it=iteraciones;
        if(u1+u2==2)
        {
            if (menos_puntos == true)
            {
                for (int i=0; i<5; i++)
                    SizeArray[i] = id[i] + 1;
            }
            else SizeArray[0]=iteraciones+1;
        }
        else
        {
            if (menos_puntos == true)
            {
                for (int i=0; i<5; i++)
                    SizeArray[i] = id[i] + 1;
            }
            else SizeArray[0]=iteraciones+2;
        }
        corridas= corridas + it + 1;
        pasos = corridas;
        estados = corridas + 1;
        it = corridas - 1;

        cpptomathlab();
        graficar();
        puntosfinales();

        if(it==itmax + 1){
            printf("Error, no se ha alcanzado el valor deseado\n");
            iteraciones=it-1;
        }

        printf("Funcin objetivo F: %f\n",F(X[0], X[1], X[2], X[3], X[4]));
        printf("Resultado: %f, %f, %f, %f, %f\n",X[0],X[1],X[2],X[3],X[4]);
        printf("Iteraciones: %d\n",it);
        printf("Estados: %d\n", estados);
        printf("Pasos o corridas: %d\n", pasos);
        printf("Tiempo: %f\n", (double)tiempo);
          printf("g1: %f, g2: %f\n\n", g1(X[0], X[1], X[2],X[3],X[4]), g2(X[2],X[3],X[4]));

    }
    for (int i=0; i<Nci; i++)
    {
        sumE=sumE+E[i];
    }
```

```
213         if (conpco==0) graficar_2();
214     }
215     system("PAUSE()");
216     return 0;
217 }
218
219 double derivada(double f2, double f1, double dx)
220 {
221         double Df = f2-f1;
222         double dfdx= Df/dx;
223         return dfdx;
224 }
225
226 double C(double x5c)
227     {
228         double C= pow(x5c-Vd,2);
229           return C;
230     }
231
232 double yn(double x4yn, double x5yn)
233     {
234         double Yn= y_0 + an*(x4yn-x5yn)*(x4yn-x5yn);
235           return Yn;
236     }
237
238 double g1(double x11, double x12, double x13, double x14, double x15)
239     {
240         double A= (x11*x11 + x12*x12 + yn(x14,x15))*x14 - x15*yn(x14,x15) - x11*x11;
241         return A;
242     }
243
244 double g2(double x23, double x24, double x25)
245     {
246         double A= yn(x24,x25)*(x25-x24) + pow(x23,2)*x25;
247         return A;
248     }
249
250 double Phi(double x_1, double x_2, double x_3, double x_4, double x_5)
251 {
252     double B=0;
253
254     double u[2]={u1,u2};
255     double g[2] = {g1(x_1,x_2,x_3,x_4,x_5), g2(x_3,x_4,x_5)};
256
257     for (int j=0; j<u1+u2; j++)
258      {
259         B = B + u[j]*pow(g[j],2);
260         if (u1==0 && u2==1)
261         {
262             B = pow(g[1],2);
263         }
264         if (u1==1 && u2==0)
265         {
266             B = pow(g[0],2);
267         }
268     }
269
270     return B;
271 }
272
273 double F(double x1F, double x2F, double x3F, double x4F, double x5F)
274 {
275     double FF = C(x5F) + Phi(x1F, x2F, x3F, x4F, x5F);
276     return FF;
277 }
278
279 void GJ()
280 {
281     double G;
282     for(int k=1-(1); k<=n-(1); k++)
283     {
284         for(int i=1-(1); i<=n-(1); i++)
285         {
286             if (i!=k)
287              {
288                G= a[i][k]/a[k][k];
289                for (int j=k; j<=n+1-(1); j++)
290                {
291                    a[i][j]=a[i][j]-G*a[k][j];
292                }
293             }
294         }
295     }
296     double Z[n];
297     for(int i=1-(1); i<=n-(1); i++)
298     {
299         Z[i]=a[i][n+1-(1)]/a[i][i];
300     }
```

```
    if (u1==0 && u2==0)
    {
       X[3]=X[3]+Z[0];
       X[4]=X[4]+Z[1];
    }
    else if (u1==0 && u2==1)
    {
       X[3]=X[3]+Z[0];
    }
    else if (u1==1 && u2==0)
    {
       X[4]=X[4]+Z[0];
    }
}

void MNR (double x1, double x2, double x3, double x4, double x5)
{
    for(int j=0; j<100; j++)
     {
       double x4a=x4;
       double x5a=x5;

       double pg1x4= derivada(g1(x1,x2,x3,x4+dx4,x5), g1(x1,x2,x3,x4,x5),dx4);
       double pg1x5= derivada(g1(x1,x2,x3,x4,x5+dx5), g1(x1,x2,x3,x4,x5), dx5);
       double pg2x4= derivada(g2(x3, x4+dx4, x5), g2(x3, x4, x5),dx4);
       double pg2x5= derivada(g2(x3, x4, x5+dx5), g2(x3, x4, x5), dx5);

          if (u1==0 && u2==0)
       {
          a[0][0]= pg1x4;
          a[0][1]= pg1x5;
          a[0][2]= -g1(x1, x2, x3, x4, x5);
          a[1][0]= pg2x4;
          a[1][1]= pg2x5;
          a[1][2]= -g2(x3, x4, x5);
       }

       if (u1==0 && u2==1)
       {
          a[0][0]= pg1x4;
          a[0][1]= -g1(x1, x2, x3, x4, x5);

       }

       if (u1==1 && u2==0)
       {
          a[0][0]= pg2x5;
          a[0][1]= -g2(x3, x4, x5);
       }

       GJ();

       if (u1==0 && u2==0)
       {
          x4=X[3];
          x5=X[4];
          double absx4_x4a=fabs(x4-x4a);
          double absx5_x5a=fabs(x5-x5a);

          if(absx4_x4a + absx5_x5a<= power)break;
       }

       else if (u1==0 && u2==1)
       {
          x4=X[3];
          double absx4_x4a=fabs(x4-x4a);
          if(absx4_x4a<= power)break;
       }

       else if (u1==1 && u2==0)
       {
          x5=X[4];
          double absx5_x5a=fabs(x5-x5a);
          if(absx5_x5a<= power)break;
       }

         if (j==99) printf("REVISA el MNR \n!!!!!!");
    }
}

void H(double x1h, double x2h, double x3h, double x4h, double x5h)
{
    if(u1==0 && u2==0)
    {
       double pfx1= derivada(F(x1h + dx1, x2h, x3h, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx1);
       double pfx2= derivada(F(x1h, x2h+dx2, x3h, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx2);
       double pfx3= derivada(F(x1h, x2h, x3h+dx3, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx3);
       double pfx4= derivada(F(x1h, x2h, x3h, x4h+dx4, x5h),F(x1h, x2h, x3h, x4h, x5h), dx4);
```

```
389         double pfx5= derivada(F(x1h, x2h, x3h, x4h, x5h+dx5),F(x1h, x2h, x3h, x4h, x5h), dx5);
390
391         MNR(x1h + dx1, x2h, x3h, x4h, x5h);
392         double dx4dx1 = (X[3] - x4h)/dx1;
393         double dx5dx1 = (X[4] - x5h)/dx1;
394         X[3]= x4h; X[4]= x5h;
395         MNR(x1h, x2h + dx2, x3h, x4h, x5h);
396         double dx4dx2 = (X[3] - x4h)/dx2;
397         double dx5dx2 = (X[4] - x5h)/dx2;
398         X[3]= x4h; X[4]= x5h;
399         MNR(x1h, x2h, x3h+dx3, x4h, x5h);
400         double dx4dx3 = (X[3] - x4h)/dx3;
401         double dx5dx3 = (X[4] - x5h)/dx3;
402         X[3]= x4h; X[4]= x5h;
403
404         h1=-(pfx1 + pfx4*dx4dx1 + pfx5*dx5dx1);
405         h2=-(pfx2 + pfx4*dx4dx2 + pfx5*dx5dx2);
406         h3=-(pfx3 + pfx4*dx4dx3 + pfx5*dx5dx3);
407     }
408
409     else if(u1==0 && u2==1)
410      {
411         double pfx1= derivada(F(x1h+dx1, x2h, x3h, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx1);
412         double pfx2= derivada(F(x1h, x2h+dx2, x3h, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx2);
413         double pfx3= derivada(F(x1h, x2h, x3h+dx3, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx3);
414         double pfx4= derivada(F(x1h, x2h, x3h, x4h+dx4, x5h),F(x1h, x2h, x3h, x4h, x5h), dx4);
415         double pfx5= derivada(F(x1h, x2h, x3h, x4h, x5h+dx5),F(x1h, x2h, x3h, x4h, x5h), dx5);
416
417         MNR(x1h + dx1, x2h, x3h, x4h, x5h);
418         double dx4dx1 = (X[3] - x4h)/dx1;
419         X[3]=x4h;
420
421         MNR(x1h, x2h + dx2, x3h, x4h, x5h);
422         double dx4dx2 = (X[3] - x4h)/dx2;
423         X[3]=x4h;
424
425         MNR(x1h, x2h, x3h+dx3, x4h, x5h);
426         double dx4dx3 = (X[3] - x4h)/dx3;
427         X[3]=x4h;
428
429         MNR(x1h, x2h, x3h, x4h, x5h+dx5);
430         double dx4dx5 = (X[3] - x4h)/dx5;
431         X[3]=x4h;
432
433         h1=-(pfx1 + pfx4*dx4dx1);
434         h2=-(pfx2 + pfx4*dx4dx2);
435         h3=-(pfx3 + pfx4*dx4dx3);
436         h5=-(pfx5 + pfx4*dx4dx5);
437     }
438
439     else if(u1==1 && u2==0)
440      {
441         double pfx1= derivada(F(x1h+dx1, x2h, x3h, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx1);
442         double pfx2= derivada(F(x1h, x2h+dx2, x3h, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx2);
443         double pfx3= derivada(F(x1h, x2h, x3h+dx3, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx3);
444         double pfx4= derivada(F(x1h, x2h, x3h, x4h+dx4, x5h),F(x1h, x2h, x3h, x4h, x5h), dx4);
445         double pfx5= derivada(F(x1h, x2h, x3h, x4h, x5h+dx5),F(x1h, x2h, x3h, x4h, x5h), dx5);
446
447         MNR(x1h + dx1, x2h, x3h, x4h, x5h);
448         double dx5dx1 = (X[4] - x5h)/dx1;
449         X[4]=x5h;
450         MNR(x1h, x2h + dx2, x3h, x4h, x5h);
451         double dx5dx2 = (X[4] - x5h)/dx2;
452         X[4]=x5h;
453         MNR(x1h, x2h, x3h+dx3, x4h, x5h);
454         double dx5dx3 = (X[4] - x5h)/dx3;
455         X[4]=x5h;
456         MNR(x1h, x2h, x3h, x4h+dx4, x5h);
457         double dx5dx4 = (X[4] - x5h)/dx4;
458         X[4]=x5h;
459
460         h1=-(pfx1 + pfx5*dx5dx1);
461         h2=-(pfx2 + pfx5*dx5dx2);
462         h3=-(pfx3 + pfx5*dx5dx3);
463         h4=-(pfx4 + pfx5*dx5dx4);
464     }
465
466     else if (u1==1 && u2==1)
467     {
468         h1= -derivada(F(x1h + dx1, x2h, x3h, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx1);
469         h2= -derivada(F(x1h, x2h+dx2, x3h, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx2);
470         h3= -derivada(F(x1h, x2h, x3h+dx3, x4h, x5h),F(x1h, x2h, x3h, x4h, x5h), dx3);
471         h4= -derivada(F(x1h, x2h, x3h, x4h+dx4, x5h),F(x1h, x2h, x3h, x4h, x5h), dx4);
472         h5= -derivada(F(x1h, x2h, x3h, x4h, x5h+dx5),F(x1h, x2h, x3h, x4h, x5h), dx5);
473     }
474 }
475
476 void MGD()
```

```
{
    int it_en_MGD[5];
    if (menos_puntos==1)
    {
       for (int i=0; i<5; i++)
       if (G[i]==1)
       if  (G[0]==1 || G[3]==1)    it_en_MGD[0] = id[0] + 1;
       if (G[1]==1 || G[4]==1) it_en_MGD[1] = id[1] + 1;
       if (G[2]==1 || G[5]==1) it_en_MGD[2] = id[2] + 1;
       if (G[0]==1 || G[1]==1 || G[2]==1)  it_en_MGD[3] = id[3] + 1;
       if (G[3]==1 || G[4]==1 || G[5]==1)  it_en_MGD[4] = id[4] + 1;
    }
    else it_en_MGD[0]= it + 1;

    int K=3+u1+u2;
    if (u1+u2<2)
    {
       MNR(X[0], X[1], X[2], X[3], X[4]);
       asignar_datos(X,it_en_MGD);
    }

    efe= F(X[0], X[1], X[2], X[3], X[4]);

    if (eps<efe)
     {
       if (u1==0 && u2==0)
         {
          H(X[0], X[1], X[2], X[3],X[4]);
          X[0]=X[0]+paso*h1;
          X[1]=X[1]+paso*h2;
          X[2]=X[2]+paso*h3;
       }
       else if (u1==0 && u2==1)
         {
          H(X[0], X[1], X[2], X[3],X[4]);

          double h[5]={h1,h2,h3,h4,h5};

          X[4]=X[4]+paso*h5;
              for (int i=2; i<3; i++)
              {
                 X[i]=X[i] + paso*h[i];
           }
       }

       else if (u1==1 && u2==0)
         {
          H(X[0], X[1], X[2], X[3],X[4]);
          double h[5]={h1,h2,h3,h4,h5};

              for (int i=2; i<4; i++)
              {
                 X[i]=X[i] + paso*h[i];
           }
       }

       else if (u1==1 && u2==1)
       {
          H(X[0],X[1],X[2],X[3],X[4]);
          double h[5]={h1,h2,h3,h4,h5};
              for (int i=2; i<5; i++)
           {
                 X[i]=X[i] + paso*h[i];
           }
              asignar_datos(X,it_en_MGD);
           iteraciones=it;
       }
     }

    else
    {
       iteraciones=it;
       it=itmax + 100000;
       if (u1==0 && u2==0 && menos_puntos==true)
       {
          if(eps>=efe)
          {
              asignar_datos(X,it_en_MGD);
          }
       }
         if (u1==1 && u2==1 && menos_puntos==true)
         {
          if(eps>=efe)
          {
              asignar_datos(X,it_en_MGD);
          }
       }
    }
```

```
565 }
```

En el renglón 566 comienza la función con la que se pasan los datos de C++ a MATLAB, y termina en el renglón 683. Con la función de la línea 685 se crean nuevos vectores para guardar los datos del vector de estados. En cada iteración, el estado es guardado para su posterior uso, utilizando la función de la línea 695, pero si, con el programa se ha decidido graficar menos puntos, entonces las variables de estado son guardadas (siempre y cuando la distancia entre las nuevas variables de estado y las anteriores sea más grandes que una cierta longitud). Esa distancia es definida en las líneas 779-784.

```
566 void cpptomathlab()
567 {
568     if (juntas == 1)
569     {
570       engEvalString(ep, "Size=10");
571       engEvalString(ep, "Ancho=3");
572       engEvalString(ep, "ColorM='magenta'");
573 
574       if (u1==0 && u2==0)
575        {
576          engEvalString(ep, "COLOR=[0 0.35 0]");
577       }
578        else if (u1==0 && u2==1)
579        {
580          engEvalString(ep, "COLOR=[1 0.6 0]");
581       }
582       else if (u1==1 && u2==0)
583        {
584          engEvalString(ep, "COLOR=[0 0.8 0]");
585       }
586 
587       else if (u1==1 && u2==1)
588       {
589          engEvalString(ep, "COLOR=[1 0.3 0]");
590       }
591    }
592 
593    else {engEvalString(ep, "COLOR=[0.3 0.3 0.3]"); engEvalString(ep, "Ancho=2.5");  engEvalString(ep, "Size=30");
          engEvalString(ep, "ColorM=[0 0 0]");}
594 
595    int s1,s2,s3,s4,s5;
596    if (menos_puntos==1) {s1=0;s2=1;s3=2;s4=3;s5=4;}
597    else {s1=s2=s3=s4=s5=0;}
598    mxArray *X1_array, *X2_array, *X3_array, *X4_array, *X5_array;
599 
600    mxArray *E_array =  mxCreateDoubleMatrix(Nci,1,mxREAL);
601    mxArray *it_array = mxCreateDoubleMatrix(1,1,mxREAL);
602    if  (G[0]==1 || G[3]==1)   X1_array = mxCreateDoubleMatrix(SizeArray[s1],1,mxREAL);
603    if (G[1]==1 || G[4]==1) X2_array = mxCreateDoubleMatrix(SizeArray[s2],1,mxREAL);
604    if (G[2]==1 || G[5]==1) X3_array = mxCreateDoubleMatrix(SizeArray[s3],1,mxREAL);
605    if (G[0]==1 || G[1]==1 || G[2]==1)  X4_array = mxCreateDoubleMatrix(SizeArray[s4],1,mxREAL);
606    if (G[3]==1 || G[4]==1 || G[5]==1)  X5_array = mxCreateDoubleMatrix(SizeArray[s5],1,mxREAL);
607    mxArray *nci_array = mxCreateDoubleMatrix(1,1,mxREAL);
608    mxArray *estados_array = mxCreateDoubleMatrix(1,1,mxREAL);
609    mxArray *cimax_array = mxCreateDoubleMatrix(1,1,mxREAL);
610    mxArray *cimin_array = mxCreateDoubleMatrix(1,1,mxREAL);
611    mxArray *conmutacion_array = mxCreateDoubleMatrix(1,1,mxREAL);
612 
613    double *pX1, *pX2, *pX3, *pX4, *pX5;
614    double *pE  = mxGetPr(E_array);
615    double *pit = mxGetPr(it_array);
616    if  (G[0]==1 || G[3]==1)   pX1 = mxGetPr(X1_array);
617    if (G[1]==1 || G[4]==1) pX2 = mxGetPr(X2_array);
618    if (G[2]==1 || G[5]==1) pX3 = mxGetPr(X3_array);
619    if (G[0]==1 || G[1]==1 || G[2]==1)  pX4 = mxGetPr(X4_array);
620    if (G[3]==1 || G[4]==1 || G[5]==1)  pX5 = mxGetPr(X5_array);
621 
622    double *pne = mxGetPr(nci_array);
623    double *pestados = mxGetPr(estados_array);
624    double *pcimax= mxGetPr(cimax_array);
625    double *pcimin= mxGetPr(cimin_array);
626    double *pconmutacion = mxGetPr(conmutacion_array);
627 
628     int k;
629    int f=1;
630    if (menos_puntos==true) f=5;
631    else f=1;
632 
633    for (int i=0; i<f; i++)
634       for (k=base;k<SizeArray[i];k++)
635          {
```

```
636             if  ((G[0]==1 || G[3]==1)) pX1[k]=datos0[k];
637             if ((G[1]==1 || G[4]==1))  pX2[k]=datos1[k];
638             if ((G[2]==1 || G[5]==1))  pX3[k]=datos2[k];
639             if ((G[0]==1 || G[1]==1 || G[2]==1))   pX4[k]=datos3[k];
640             if ((G[3]==1 || G[4]==1 || G[5]==1))   pX5[k]=datos4[k];
641         }
642
643    for (int k=0;k<Nci;k++)
644    {
645       pE[k]=E[k]/1;
646    }
647
648    pit[0]=it;
649    pne[0]= ci;
650    pestados[0]=estados;
651    pcimax[0]= cimax;
652    pcimin[0]= cimin;
653    if (conpco==0) pconmutacion[0]=ci+1;
654    else pconmutacion[0]=contador+1;
655    if  (G[0]==1 || G[3]==1)   engPutVariable(ep,"X1",X1_array);
656    if (G[1]==1 || G[4]==1) engPutVariable(ep,"X2",X2_array);
657    if (G[2]==1 || G[5]==1) engPutVariable(ep,"X3",X3_array);
658    if (G[0]==1 || G[1]==1 || G[2]==1)  engPutVariable(ep,"X4",X4_array);
659    if (G[3]==1 || G[4]==1 || G[5]==1)  engPutVariable(ep,"X5",X5_array);
660
661    engPutVariable(ep,"nci",nci_array);
662    engPutVariable(ep,"estados",estados_array);
663    engPutVariable(ep,"it",it_array);
664    engPutVariable(ep,"E",E_array);
665    engPutVariable(ep,"cimin",cimin_array);
666    engPutVariable(ep,"cimax",cimax_array);
667    engPutVariable(ep,"conmutacion",conmutacion_array);
668
669    engEvalString(ep,"for i=1:estados   X1(i, nci)=x1(i); X2(i, nci)=x2(i); X3(i, nci)=x3(i); X4(i, nci)=x4(i); X5
         (i, nci)=x5(i); end");
670    if (conpco==0) engEvalString(ep, "itera(nci+1)=it;");
671
672    mxDestroyArray(nci_array);
673    mxDestroyArray(estados_array);
674    if  (G[0]==1 || G[3]==1)   mxDestroyArray(X1_array);
675    if (G[1]==1 || G[4]==1) mxDestroyArray(X2_array);
676    if (G[2]==1 || G[5]==1) mxDestroyArray(X3_array);
677    if (G[0]==1 || G[1]==1 || G[2]==1)  mxDestroyArray(X4_array);
678    if (G[3]==1 || G[4]==1 || G[5]==1)  mxDestroyArray(X5_array);
679    mxDestroyArray(cimin_array);
680    mxDestroyArray(cimax_array);
681    mxDestroyArray(conmutacion_array);
682
683 }
684
685 void crear_vectores_datos()
686 {
687    if  (G[0]==1 || G[3]==1)   datos0 = new double[gato+1];
688    if (G[1]==1 || G[4]==1) datos1 = new double[gato+1];
689    if (G[2]==1 || G[5]==1)    datos2 = new double[gato+1];
690    if  (G[0]==1 || G[1]==1 || G[2]==1) datos3 = new double[gato+1];
691    if (G[3]==1 || G[4]==1 || G[5]==1) datos4 = new double[gato+1];
692    if  (G[0]==0 && G[1]==0 && G[2]==0 && G[3]==0 && G[4]==0 && G[5]==0) printf("No se ha solicitado graficar");
693 }
694
695 void asignar_datos(double vdatos[],int coordenada[])
696 {
697    if (menos_puntos==false)
698    {
699       if  (G[0]==1 || G[3]==1)   datos0[coordenada[0]]=vdatos[0];
700       if (G[1]==1 || G[4]==1) datos1[coordenada[0]]=vdatos[1];
701       if (G[2]==1 || G[5]==1) datos2[coordenada[0]]=vdatos[2];
702       if (G[0]==1 || G[1]==1 || G[2]==1)  datos3[coordenada[0]]=vdatos[3];
703       if (G[3]==1 || G[4]==1 || G[5]==1)  datos4[coordenada[0]]=vdatos[4];
704    }
705    if (menos_puntos==true)
706    {
707       data0=data1=data2=data3=data4=0;
708       for (int i=0; i<9; i++)
709       {
710          if (G[i]==1)
711          {
712             int j;
713             if(i<3) j=3;
714             else if(i>=3 && i<6) j=4;
715
716             int k;
717              if(i==0 || i==3 || i==6) k=0;
718             else if(i==1 || i==4)   k=1;
719             else if(i==2 || i==5)   k=2;
720                   double x_actual=X[k];
721                   double y_actual=X[j];
722
```

```
723                     if(distancia(x_anterior[k],x_anterior[j],x_actual,y_actual)>=1e-10  ||  it== -1 ||  (it==pco &&
                            corridas==0))
724                     {
725                         x_anterior[k] = X[k];
726                         x_anterior[j] = X[j];
727 
728                         if  ((G[0]==1 || G[3]==1))
729                         {
730                             if (data0==0)
731                             {
732                                 datos0[coordenada[0]]=vdatos[0];
733                                 if (it!=-1) id[0]=id[0]+1;
734                             }
735                             data0++;
736                         }
737                         if ((G[1]==1 || G[4]==1))
738                         {
739                             if (data1==0)
740                             {
741                                 datos1[coordenada[1]]=vdatos[1];
742                                 if (it!=-1) id[1]=id[1]+1;
743                             }
744                             data1++;
745                         }
746                         if ((G[2]==1 || G[5]==1))
747                         {
748                             if (data2==0)
749                             {
750                                 datos2[coordenada[2]]=vdatos[2];
751                                 if (it!=-1) id[2]=id[2]+1;
752                             }
753                             data2++;
754                         }
755                         if ((G[0]==1 || G[1]==1 || G[2]==1))
756                         {
757                             if (data3==0)
758                             {
759                                 datos3[coordenada[3]]=vdatos[3];
760                                 if (it!=-1) id[3]=id[3]+1;
761                             }
762                             data3++;
763                         }
764                         if ((G[3]==1 || G[4]==1 || G[5]==1))
765                         {
766                             if (data4==0)
767                             {
768                                 datos4[coordenada[4]]=vdatos[4];
769                                 if (it!=-1) id[4]=id[4]+1;
770                             }
771                             data4++;
772                         }
773                     }
774             }
775         }
776     }
777 }
778 
779 double distancia(double x_1, double y_1, double x_2, double y_2)
780 {
781     double d;
782     d=(x_1 - x_2)*(x_1 - x_2) + (y_1 - y_2)*(y_1 - y_2);
783     return d;
784 }
```

De la línea 785 a la número 960, se tiene el código con los comandos para graficar en MATLAB.

```
785 void ejes()
786 {
787     engEvalString(ep, "hold on;");
788     engEvalString(ep, "plot([-5,5], [0, 0], 'LineWidth',2, 'color', 'black');");
789     engEvalString(ep, "plot([0,0], [-5, 5], 'LineWidth',2, 'color', 'black');");
790 }
791 
792 void graficar()
793 {
794     engEvalString(ep, "hold on;");
795 
796     if (G[0]==1)
797     {
798         engEvalString(ep, "figure(1);");
799         ejes();
800         engEvalString(ep, "grid on");
801         engEvalString(ep, "line(X1, X4, 'color',COLOR, 'LineWidth',Ancho);");
802         engEvalString(ep, "title('x_4 VS x_1','FontSize', 14);");
803         engEvalString(ep, "xlabel ('x_1','FontSize', 14);");
```

```
804         engEvalString(ep, "ylabel('x_4','FontSize',14);");
805     }
806 
807     if (G[1]==1)
808     {
809         engEvalString(ep, "figure(2);");
810         ejes();
811         engEvalString(ep, "grid on");
812         engEvalString(ep, "line(X2, X4,'color', COLOR,'LineWidth',Ancho);");
813         engEvalString(ep, "title('x_4 VS x_2');");
814         engEvalString(ep, "xlabel('x_2','FontSize',14);");
815         engEvalString(ep, "ylabel('x_4','FontSize',14);");
816     }
817 
818     if (G[2]==1)
819     {
820         engEvalString(ep, "figure(3);");
821         ejes();
822         engEvalString(ep, "grid on");
823         engEvalString(ep, "line(X3, X4,'color',COLOR,'LineWidth',Ancho);");
824         engEvalString(ep, "title('x_4 vs. x_3','FontSize',14);");
825         engEvalString(ep, "xlabel('x_3','FontSize',14);");
826         engEvalString(ep, "ylabel('x_4','FontSize',14);");
827     }
828 
829     if (G[3]==1)
830     {
831            engEvalString(ep, "figure(4);");
832         ejes();
833         engEvalString(ep, "grid on");
834         engEvalString(ep, "line(X1, X5,'color',COLOR,'LineWidth',Ancho);");
835         engEvalString(ep, "title('x_5 VS x_1','FontSize',14);");
836         engEvalString(ep, "xlabel('x_1','FontSize',14);");
837         engEvalString(ep, "ylabel('x_5','FontSize',14);");
838     }
839 
840     if (G[4]==1)
841     {
842         engEvalString(ep, "figure(5);");
843         ejes();
844         engEvalString(ep, "grid on");
845         engEvalString(ep, "line(X2, X5,'color', COLOR,'LineWidth',Ancho);");
846         engEvalString(ep, "title('x_5 VS x_2');");
847         engEvalString(ep, "xlabel('x_2','FontSize',14);");
848         engEvalString(ep, "ylabel('x_5','FontSize',14);");
849     }
850 
851     if (G[5]==1)
852     {
853         engEvalString(ep, "figure(6);");
854         ejes();
855         engEvalString(ep, "grid on");
856         engEvalString(ep, "line(X3, X5,'color',[0,0,0],'LineWidth',2.5),");
857         engEvalString(ep, "title('x_5 vs. x_3','FontSize',28);");
858         engEvalString(ep, "xlabel('x_3','FontSize',24);");
859         engEvalString(ep, "ylabel('x_5','FontSize',24);");
860         engEvalString(ep, "set(gca,'fontsize',16)");
861     }
862 
863     engEvalString(ep, "nconmu(conmutacion)=conmutacion; itera(conmutacion)=it;");
864 
865 // if (conpco>=1)
866 // {
867 //     engEvalString(ep, "figure(30);");
868 //     engEvalString(ep, "hold on;");
869 //     engEvalString(ep, "plot(nconmu, itera);");
870 //     engEvalString(ep, "hold on;");
871 //     engEvalString(ep, "plot(conmutacion, it,'.','MarkerSize',20, 'MarkerEdgeColor', 'red');");
872 //     engEvalString(ep, "title('NMERO DE PASOS VS NMERO DE CONMUTACIN');");
873 //     engEvalString(ep, "xlabel 'NUMERO DE CONMUTACIN';");
874 //     engEvalString(ep, "ylabel 'NMERO DE PASOS'");
875 //     engEvalString(ep, "hold off;");
876 // }
877 }
878 
879 void graficar_2()
880 {
881     engEvalString(ep, "figure (8);");
882     engEvalString(ep, "plot(cimax:-0.2:cimin, itera+1, color', 'black');");
883     engEvalString(ep, "hold on;");
884     engEvalString(ep, "plot(cimax:-0.2:cimin, itera+1,'.','color','black','MarkerSize',20,'MarkerEdgeColor','
          black');");     //Cuidado con la distancia de separacin !!!!!!!!!!!!!!!!!!!!!!!!!!!!
885     engEvalString(ep, "title('Nmero de pasos vs. condicin inicial para x_2','FontSize',28);');");
886     engEvalString(ep, "xlabel 'Condicin inicial para x_2','FontSize',24);;");
887     engEvalString(ep, "ylabel 'Nmero de pasos','FontSize',24); set(gca,'fontsize',16");
888     engEvalString(ep, "hold off;");
889 }
890 
```

```
void puntosiniciales()
{
       if (G[0]==1)
       {
          engEvalString(ep, "figure(1);");
          engEvalString(ep, "line(X1(1),X4(1),'Marker','o','LineWidth',2.5,'MarkerSize',6,'MarkerEdgeColor','
               black')");
       }
       if (G[1]==1)
       {
          engEvalString(ep, "figure(2);");
          engEvalString(ep, "line(X2(1),X4(1),'Marker','o','LineWidth',2.5,'MarkerSize',6,'MarkerEdgeColor','
               black')");
       }
       if (G[2]==1)
       {
          engEvalString(ep, "figure(3);");
          engEvalString(ep, "line(X3(1),X4(1),'Marker','o','LineWidth',2.5,'MarkerSize',6,'MarkerEdgeColor','
               black')");
       }
       if (G[3]==1)
       {
          engEvalString(ep, "figure(4);");
          engEvalString(ep, "line(X1(1),X5(1),'Marker','o','LineWidth',2.5,'MarkerSize',6,'MarkerEdgeColor','
               black')");
       }
       if (G[4]==1)
       {
          engEvalString(ep, "figure(5);");
          engEvalString(ep, "line(X2(1),X5(1),'Marker','o','LineWidth',2.5,'MarkerSize',6,'MarkerEdgeColor','
               black')");
       }
       if (G[5]==1)
       {
          engEvalString(ep, "figure(6);");
          engEvalString(ep, "line(X3(1),X5(1),'Marker','o','LineWidth',1,'MarkerSize',8,'MarkerEdgeColor','
               black')");
       }
}

void puntosfinales()
{
       if (G[0]==1 || G[1]==1 || G[2]==1)  engEvalString(ep, "lastpoint=size(X4);lastpoint=lastpoint(1,1);");
       if (G[0]==1)
       {
          engEvalString(ep, "figure(1);");
          engEvalString(ep, "line(X1(lastpoint),X4(lastpoint),'color',[1 0 1],'Marker','x','LineWidth',2.5,'
               MarkerSize',Size,'MarkerEdgeColor',ColorM)");
       }
       if (G[1]==1)
       {
          engEvalString(ep, "figure(2);");
          engEvalString(ep, "line(X2(lastpoint),X4(lastpoint),'color',[1 0 1],'Marker','x','LineWidth',2.5,'
               MarkerSize',Size,'MarkerEdgeColor',ColorM)");
       }
       if (G[2]==1)
       {
          engEvalString(ep, "figure(3);");
          engEvalString(ep, "line(X3(lastpoint),X4(lastpoint),'color',[1 0 1],'Marker','x','LineWidth',2.5,'
               MarkerSize',Size,'MarkerEdgeColor',ColorM)");
       }

       if (G[3]==1 || G[4]==1 || G[5]==1)  engEvalString(ep, "lastpoint=size(X5);lastpoint=lastpoint(1,1);");
       if (G[3]==1)
       {
          engEvalString(ep, "figure(4);");
          engEvalString(ep, "line(X1(lastpoint),X5(lastpoint),'color',[1 0 1],'Marker','x','LineWidth',2.5,'
               MarkerSize',Size,'MarkerEdgeColor',ColorM)");
       }
       if (G[4]==1)
       {
          engEvalString(ep, "figure(5);");
          engEvalString(ep, "line(X2(lastpoint),X5(lastpoint),'color',[1 0 1],'Marker','x','LineWidth',2.5,'
               MarkerSize',Size,'MarkerEdgeColor',ColorM)");
       }
       if (G[5]==1)
       {
          engEvalString(ep, "figure(6);");
          engEvalString(ep, "line(X3(lastpoint),X5(lastpoint),'color',[0 0 0],'Marker','x','LineWidth',2.5,'
               MarkerSize',20,'MarkerEdgeColor',ColorM)");
       }
}
```

E.3. Código para diseño de circuito activo con un transistor

Se presenta el código utilizado para el procesamiento de las figuras y de la información utilizadas para el análisis del circuito activo con un trasistor. Algunas funciones son similares a las mostradas en los códigos anteriores, por lo que también se hará una breve revisión del mismo.

Para cargar las librerías se tienen escritas las líneas de código de la 1 a la 7. Desde el renglón 9 hasta el 10 se asignan valores a diferentes variables, entre ellas están las variables de estado en 16-18. Se inicializa MATLAB con la línea 22. Variables y funciones globales son declaradas desde la línea 24 hasta la 51. El programa principal va desde la línea 55 hasta la 244. En 65 se asigna el valor de 1 para elegir las gráficas que se quieren crear, correspondientes a 9 proyecciones. También este programa contiene la opción de elegir entre graficar todos los puntos de la trayectoria o ahorrar espacio de memoria, véase la línea 68. El vector de estados se encuentra en los renglones 86-91. En las líneas 103-108 se encuentran sentencias de decisión para permitir grabar el primer valor del vector de estados, dependiendo de las gráficas solicitadas con los valores de *G[]*. Con la función del renglón 112 se crean los vectores que graban los datos necesarios para poder graficar. En 118 se procesan los valores iniciales y luego se grafican en 119. En 124 se encuentra la llamada a la función de la metodología general de diseño. Si no se ha sobrepasado el número máximo de iteraciones y además ya se ha cumplido el número inicial de pasos previamente solicitados al usuario, entonces se realiza la rutina acotada por las líneas 125 y 185, con la cual se grafica el primer conjunto de datos obtenido.

Otra sección muy importante para mencionar es la que involucra las líneas 248 a la 313, pues corresponde a las ecuaciones que describen el comportamiento del transistor. Luego, de 315 a 324 se cuenta con el código de los voltajes base-emisor, base-colector y colector-emisor. En 326 se cuenta con una función para derivar. En 333 está la función objetivo tradicional. Luego, la función de penalización se describe desde la línea 340. Le sigue la función objetivo total en 379. A través de los renglones 385 al 410 se tiene las funciones que contienen las ecuaciones del comportamiento del circuito con transistor.

```
1  #include <iostream>
2  #include <time.h>
3  #include <math.h>
4  #include<stdlib.h>
5  #include<fstream>
6  #include <stdio.h>
7  #include "engine.h"
8
9  #define     eps      1e-6     //psilon, valor de comparacin contra funcin objetivo
10 #define     power 1e-6
11 #define     Vced  5.0
12 #define     Vbed  1.2
13 #define     E1 4.0       //Voltaje de entrada
14 #define     E2 20.0      //Voltaje de entrada
15 #define     N  6
16 #define     U1 1
17 #define     U2 1
18 #define     U3 1
19 #define     ene   3-U1-U2-U3
20 #define     itmax 100000000
21
22 Engine *ep = engOpen(NULL);
23
24 double Nci, cimax, cimin, distanciaci, X[N], a[3][3+1], x[6], x1, x2, x3, x4, x5, x6, Ib, Ie, Ic;
25 double h1, h2, h3, h4, h5, h6, g1dx4, g1dx5, g1dx6, g2dx4, g2dx5, g2dx6, g3dx4, g3dx5, g3dx6,
26 Vb, Ve, Vc, Vce, Vbe, Vbc, efe, D2, paso, paso1, paso2, paso3, paso4, paso5, paso6, x_anterior[6];
27 int K, M, it, contador, G[9], id[6];
```

```
clock_t t;
bool menos_puntos;

double GMIN= pow(10, -12), Is = pow(10,-16);
double betaF = 100;
double betaR = 1;
double T = 300.15;
double k = 1.38 * pow(10, -23);
double q = 1.60 * pow(10, -19);
double TV = -5*k*T/q;
double Vt = k*T/q;

double derivada(double, double, double), yn(double, double), F(double, double, double,double, double, double),
g1(double, double, double, double), g2(double, double, double, double), g3(double, double, double, double),
Phi(double, double, double, double, double, double), distancia(double,double,double,double);
void MNR(double, double, double, double, double, double), MGD(), Vbe_Vce(double, double , double), Ibce(double,
     double, double),
H(double, double, double, double, double, double), graficar_2(), GJ(), puntosiniciales(), puntosfinales(), ejes()
     ,
graficar(), cpptomathlab(), crear_vectores_datos(), asignar_datos(double[],int[]), liberar_espacio();

double *datos0, *datos1, *datos2, *datos3, *datos4, *datos5, dx1, dx2, dx3, dx4, dx5, dx6;

int pco, mcontador, juntas, *itera, ci,
tiempo, iteraciones, estados, pasos, sumE, conpco, n, u1, u2, u3, D1, SizeArray[6], base, corridas, gato,
data0, data1, data2, data3, data4, data5, E[100];

using namespace std;

int main ()
{
   base=0;      u1=U1; u2=U2; u3=U3; Nci=1;       ci = 1;      distanciaci=0.2;  mcontador=1;
   K= 3+u1+u2+u3;     M= N-K;      n=M;
   cimax=1; cimin=1;
   Nci=1+(cimax-cimin)/distanciaci;
   itera = new int[int(Nci)];
   int U[3]={u1, u2, u3};
   engEvalString(ep, "clear all");

   G[0]=0; G[1]=0; G[2]=0; G[3]=0; G[4]=0; G[5]=0; G[6]=0; G[7]=0; G[8]=1;

   conpco=1;
   menos_puntos = false;

   paso=paso1=paso2=paso3=paso4=paso5=paso6=1e-4;
   if (conpco==0) gato= 10000000;

   for (contador = 0; contador < mcontador; contador=contador+1)
   {
   if (conpco>=1)
      {
      cout << "PASOS ANTES DE CONMUTACION"<< endl;
      cin >> pco;  //Paso de conmutacin
      gato=pco;
      u1=U[0]; u2=U[1]; u3=U[2];
      n=M;
      }

      for (ci=0; ci<Nci; ci=ci+1)
      {
         X[0]=5.0;          //Valores iniciales del vector X
          X[1]=pow(1/0.06,0.5);
          X[2]=1.0;
          X[3]=0.0;             //VB
          X[4]=3.0;          //VE
         X[5]=17.4;          //VC
//
            id[0]=id[1]=id[2]=id[3]=id[4]=id[5]=0;
         double dx[N];
         for (int i=0; i<N; i++) dx[i]=1e-7;
         dx1=dx[0]; dx2=dx[1]; dx3=dx[2]; dx4=dx[3]; dx5=dx[4]; dx6=dx[5];

         printf("# CONMUTACION: %d\n", contador+1);
         printf("efe = %.10f, X0=%.10f, X1=%.10f, X2=%.10f, X3=%.10f, X4=%.10f, X5=%.10f\n",efe, X[0], X[1], X
              [2], X[3], X[4], X[5]);

         x[0]=X[0];  x[1]=X[1];  x[2]=X[2];  x[3]=X[3];  x[4]=X[4];  x[5]=X[5];

         if  (G[0]==1 || G[3]==1 || G[6]==1) x_anterior[0] = x[0];
         if (G[1]==1 || G[4]==1 || G[7]==1)  x_anterior[1] = x[1];
         if (G[2]==1 || G[5]==1 || G[8]==1)  x_anterior[2] = x[2];
         if (G[0]==1 || G[1]==1 || G[2]==1)  x_anterior[3] = x[3];
         if (G[3]==1 || G[4]==1 || G[5]==1)  x_anterior[4] = x[4];
         if (G[6]==1 || G[7]==1 || G[8]==1)  x_anterior[5] = x[5];

         t=clock();
         corridas=0;
         crear_vectores_datos();
```

```
113             it=-1;
114             int coo_inicial[]={0,0,0,0,0,0,0,0,0};
115             asignar_datos(X,coo_inicial);
116             for (int i=0; i<6; i++)     SizeArray[i]=1;
117
118             cpptomathlab();
119             puntosiniciales();
120                engEvalString(ep, "clear all");
121
122             for (it = 0; it <= itmax; it=it+1)
123              {
124                MGD();
125                if ((it+1)%gato==0 && it<=itmax)
126                {
127                   int it_id[6];
128                   efe= F(X[0], X[1], X[2], X[3], X[4], X[5]);
129                   printf("efe = %.10f, X0=%.10f, X1=%.10f, X2=%.10f, X3=%.10f, X4=%.10f, X5=%.10f\n",efe, X[0], X
                          [1], X[2], X[3], X[4], X[5]);
130                   if (menos_puntos == true)
131                   {
132                      for (int i=0; i<6; i++)
133                      {
134                         SizeArray[i]=id[i]+1;
135                         it_id[i] = id[i];
136                      }
137                   }
138                   else
139                   {
140                      SizeArray[0]=gato+1;
141                      it_id[0] = it_id[1] = it_id[2] = it_id[3]= it_id[4] = it_id[5] = it+1;
142                   }
143                   cpptomathlab();
144                   graficar();
145                   corridas= corridas + it + 1;
146                   printf("corridas = %d\n",corridas);
147
148                   double dd[6];
149                   if  (G[0]==1 || G[3]==1 || G[6]==1) dd[0]=datos0[it_id[0]];
150                   if (G[1]==1 || G[4]==1 || G[7]==1)  dd[1]=datos1[it_id[1]];
151                   if (G[2]==1 || G[5]==1 || G[8]==1)  dd[2]=datos2[it_id[2]];
152                   if (G[0]==1 || G[1]==1 || G[2]==1)  dd[3]=datos3[it_id[3]];
153                   if (G[3]==1 || G[4]==1 || G[5]==1)  dd[4]=datos4[it_id[4]];
154                   if (G[6]==1 || G[7]==1 || G[8]==1)  dd[5]=datos5[it_id[5]];
155
156                   int estrategia;
157                   if (conpco==0)
158                   {
159                      cout << "Estrategia"<< endl;
160                      cin >> estrategia;
161                      if (estrategia==0) {u1=0;  u2=0; u3=0;}
162                      else if (estrategia==1) {u1=0;   u2=0; u3=1;}
163                      else if (estrategia==2) {u1=0;   u2=1; u3=0;}
164                      else if (estrategia==3) {u1=0;   u2=1; u3=1;}
165                      else if (estrategia==4) {u1=1;   u2=0; u3=0;}
166                      else if (estrategia==5) {u1=1;   u2=0; u3=1;}
167                      else if (estrategia==6) {u1=1;   u2=1; u3=0;}
168                      else if (estrategia==7) {u1=1;   u2=1; u3=1;}
169                   }
170
171                   if(pco<=it+1 && conpco>0) {u1=0; u2=0; u3=0; n= 3; conpco=0;}
172                   it=-1;
173                   for (int i=0; i<6; i++) id[i]=0;
174
175                   liberar_espacio();
176                   cout << "Ingresar tamao del parmetro de escala"<< endl;
177                   cin >> paso;
178                   cout << "Ingresar nmero de iteraciones siguientes"<< endl;
179                   cin >> gato;
180
181                   paso1=paso; paso2=paso; paso3=paso; paso4=paso; paso5=paso; paso6=paso;
182                   crear_vectores_datos();
183                   asignar_datos(dd,coo_inicial);
184                }
185             }
186
187             tiempo=clock()-t;
188
189             if(u1*u2*u3==1)
190             {
191                it=iteraciones;
192                if (menos_puntos == true)
193                {
194                   for (int i=0; i<6; i++)
195                         SizeArray[i] = id[i] + 1;
196                }
197                else SizeArray[0]= iteraciones + 1;
198             }
199             else{
```

```
200             it=iteraciones;
201             if (menos_puntos == true)
202             {
203                for (int i=0; i<6; i++)
204                      SizeArray[i] = id[i] + 1;
205             }
206             else SizeArray[0]= iteraciones + 2;
207          }
208
209          corridas= corridas + it + 1;
210          pasos = corridas;
211          estados = corridas + 1;
212          it = corridas - 1;
213
214          itera[ci]=it;
215          cpptomathlab();
216          graficar();
217          puntosfinales();
218
219          if(it==itmax + 1)
220          {
221             printf("Error, no se ha alcanzado el valor deseado\n");
222             iteraciones=it-1;
223          }
224
225          printf("Funcin objetivo F: %f\n",F(X[0], X[1], X[2],X[3], X[4], X[5]));
226          printf("Resultado: %.12f, %.12f, %.12f, %.12f, %.12f, %.12f\n",X[0],X[1],X[2],X[3],X[4],X[5]);
227          printf("Iteraciones: %d\n",it);
228          printf("Estados: %d\n", estados);
229          printf("Pasos o corridas: %d\n", pasos);
230          Ibce(X[3], X[4], X[5]);
231          printf("Ib = %f, Ic = %f, Ie = %f\n\n",Ib, Ic, Ie);
232
233          liberar_espacio();
234          delete [] itera;
235       }
236       for (int i=0; i<Nci; i++)
237       {
238          sumE=sumE+E[i];
239       }
240
241       if (conpco==0) graficar_2();
242    }
243    system("PAUSE()");
244 return 0;
245 }
246
247 //-------------------------------------------------------Transistor-----------------------------
248 {
249     Vbe_Vce(x4I, x5I, x6I);
250    double coef;
251    coef=36.6*Vt;
252    double ye = (Vbe/Vt + 1 - coef/Vt)*exp(coef/Vt);
253    double yc = (Vbc/Vt + 1 - coef/Vt)*exp(coef/Vt);
254
255    if (Vbe > TV && Vbc > TV)
256     {
257       if(Vbe<coef && Vbc<coef)
258       {
259          Ic = Is*((exp(Vbe/Vt) - exp(Vbc/Vt)) - (1/betaR)*(exp(Vbc/Vt)-1)) + (Vbe - (1+1/betaR)*Vbc)*GMIN;
260          Ib = Is*(((1/betaF))*(exp(q*Vbe/(k*T))-1) + (1/betaR)*(exp(q*Vbc/(k*T)) - 1) ) + (Vbe/betaF + Vbc/betaR)
                *GMIN;
261       }
262       else if (Vbe>=coef && Vbc<coef)
263       {
264          Ic = Is*((ye - exp(q*Vbc/(k*T))) - (1/betaR)*(exp(Vbc/Vt)-1)) + (Vbe - (1+1/betaR)*Vbc)*GMIN;
265          Ib = Is*(((1/betaF))*(ye-1) + (1/betaR)*(exp(Vbc/Vt) - 1) ) + (Vbe/betaF + Vbc/betaR)*GMIN;
266       }
267       else if (Vbe<coef && Vbc>=coef)
268       {
269          Ic = Is*((exp(Vbe/Vt) - yc) - (1/betaR)*(yc-1)) + (Vbe - (1+1/betaR)*Vbc)*GMIN;
270          Ib = Is*(((1/betaF))*(exp(q*Vbe/Vt)-1) + (1/betaR)*(yc - 1) ) + (Vbe/betaF + Vbc/betaR)*GMIN;
271       }
272       else if (Vbe>=coef && Vbc>=coef)
273       {
274          Ic = Is*((ye - yc) - (1/betaR)*(yc-1)) + (Vbe - (1+1/betaR)*Vbc)*GMIN;
275          Ib = Is*(((1/betaF))*(ye-1) + (1/betaR)*(yc - 1) ) + (Vbe/betaF + Vbc/betaR)*GMIN;
276       }
277    }
278
279    if (Vbe <= TV && Vbc > TV)
280    {
281       if(Vbc<coef)
282       {
283          Ic = -Is*(exp(Vbc/Vt) + (1/betaR)*(exp(Vbc/Vt) - 1)) + (Vbe - (1+1/betaR)*Vbc)*GMIN;
284          Ib = -Is*(1/betaF - (1/betaR)*(exp(Vbc/Vt) - 1)) + (Vbe/betaF + Vbc/betaR)*GMIN;
285       }
286       else
```

```
287         {
288             Ic = -Is*(yc + (1/betaR)*(yc - 1)) + (Vbe - (1+1/betaR)*Vbc)*GMIN;
289             Ib = -Is*(1/betaF - (1/betaR)*(yc - 1)) + (Vbe/betaF + Vbc/betaR)*GMIN;
290         }
291     }
292 
293     if (Vbe <= TV && Vbc <= TV)
294     {
295         Ic = Is/betaR + (Vbe - (1+1/betaR)*Vbc)*GMIN;
296         Ib = -Is*((betaF + betaR)/betaF*betaR) + (Vbe/betaF + Vbc/betaR)*GMIN;
297     }
298 
299     if (Vbe > TV && Vbc <= TV)
300     {
301         if(Vbe<coef)
302         {
303             Ic = Is*(exp(Vbe/Vt) + 1/betaR) + (Vbe - (1 + 1/betaR)*Vbc)*GMIN;
304             Ib = Is*((1/betaF)*(exp(Vbe/Vt) - 1) - 1/betaR) + (Vbe/betaF + Vbc/betaR)*GMIN;
305         }
306         else
307         {
308             Ic = Is*(ye + 1/betaR) + (Vbe - (1 + 1/betaR)*Vbc)*GMIN;
309             Ib = Is*((1/betaF)*(ye - 1) - 1/betaR) + (Vbe/betaF + Vbc/betaR)*GMIN;
310         }
311     }
312     Ie = -(Ib + Ic);
313 }
314 
315 void Vbe_Vce(double x4volt, double x5volt, double x6volt)
316 {
317 
318         Vb= x4volt; //V1
319         Ve= x5volt; //V2
320         Vc= x6volt; //V3
321         Vbe= Vb-Ve;
322          Vbc= Vb-Vc;
323          Vce= Vc-Ve;
324 }
325 
326 double derivada(double f2, double f1, double dx)
327 {
328         double Df = f2-f1;
329         double dfdx= Df/dx;
330         return dfdx;
331 }
332 
333 double C(double x4c, double x5c, double x6c)
334     {
335         Vbe_Vce(x4c, x5c, x6c);
336         double C= pow(Vbe-Vbed,2)+pow(Vce-Vced,2);//+pow(x4c-0.3,2);
337         return C;
338     }
339 
340 double Phi(double x1Phi, double x2Phi, double x3Phi, double x4Phi, double x5Phi, double x6Phi)
341 {
342    double B=0;
343 
344    double u[3]={u1,u2,u3};
345    double g[3] = {g1(x1Phi, x4Phi, x5Phi, x6Phi), g2(x2Phi, x4Phi, x5Phi, x6Phi), g3(x3Phi, x4Phi, x5Phi, x6Phi)
         };
346 
347    for (int j=0; j<u1+u2+u3; j++)
348     {
349       B = B + u[j]*pow(g[j],2);
350       if (u1==0 && u2==0 && u3==1)
351       {
352          B = pow(g[2],2);
353       }
354       if (u1==0 && u2==1 && u3==0)
355       {
356          B = pow(g[1],2);
357       }
358       if (u1==0 && u2==1 && u3==1)
359       {
360          B = pow(g[1],2) + pow(g[2],2);
361       }
362       if (u1==1 && u2==0 && u3==0)
363       {
364          B = pow(g[0],2);
365       }
366       if (u1==1 && u2==0 && u3==1)
367       {
368          B = pow(g[0],2) + pow(g[2],2);
369       }
370       if (u1==1 && u2==1 && u3==0)
371       {
372          B = pow(g[0],2) + pow(g[1],2);
373       }
```

```
374     }
375
376     return B;
377 }
378
379 double F(double x1F, double x2F, double x3F, double x4F, double x5F, double x6F)
380 {
381     double FF = C(x4F, x5F, x6F)+ Phi(x1F, x2F, x3F, x4F, x5F, x6F);
382     return FF;
383 }
384
385 double g1(double x1g1, double x4g1, double x5g1, double x6g1)
386 {
387     Vbe_Vce(x4g1, x5g1, x6g1);
388      double A;
389      Ibce(x4g1, x5g1, x6g1);
390      A = Ib + (Vb - E1)*x1g1*x1g1;
391     return A;
392 }
393
394 double g2(double x2g2, double x4g2, double x5g2, double x6g2)
395 {
396     Vbe_Vce(x4g2, x5g2, x6g2);
397      double A;
398      Ibce(x4g2, x5g2, x6g2);
399     A = Ie + Ve*x2g2*x2g2;
400     return A;
401 }
402
403 double g3(double x3g3, double x4g3, double x5g3, double x6g3)
404 {
405     Vbe_Vce(x4g3, x5g3, x6g3);
406      double A;
407      Ibce(x4g3, x5g3, x6g3);
408     A = Ic + (Vc - E2)*x3g3*x3g3;
409     return A;
410 }
```

El Método de Newton-Raphson abarca desde la línea 413 hasta la 568. El método de Gauss-Jordan desde el renglón 570 hasta el 628. La función de dirección de movimiento de la trayectoria, se tiene desde la línea 631 hasta la número 841. La metodología general de diseño va desde la 845 hasta la 972.

```
413 void MNR (double x1, double x2, double x3, double x4, double x5, double x6)
414 {
415     for(int j=0; j<100000; j++)
416      {
417        double x4a=x4;
418        double x5a=x5;
419        double x6a=x6;
420
421        double g_1= g1(x1,x4,x5,x6);
422        double g_2=g2(x2, x4, x5, x6);
423        double g_3=g3(x3,x4,x5,x6);
424
425        double pg1x4= derivada(g1(x1,x4+dx4,x5,x6), g_1, dx4);
426        double pg1x5= derivada(g1(x1,x4,x5+dx5,x6), g_1, dx5);
427        double pg1x6= derivada(g1(x1,x4,x5,x6+dx6), g_1, dx6);
428
429        double pg2x4= derivada(g2(x2, x4+dx4, x5, x6), g_2, dx4);
430        double pg2x5= derivada(g2(x2, x4, x5+dx5, x6), g_2, dx5);
431        double pg2x6= derivada(g2(x2, x4, x5, x6+dx6), g_2, dx6);
432
433        double pg3x4= derivada(g3(x3,x4+dx4,x5,x6), g_3, dx4);
434        double pg3x5= derivada(g3(x3,x4,x5+dx5,x6), g_3, dx5);
435        double pg3x6= derivada(g3(x3,x4,x5,x6+dx6), g_3, dx6);
436
437          if (u1==0 && u2==0 && u3==0)
438        {
439           a[0][0]= pg1x4;
440           a[0][1]= pg1x5;
441           a[0][2]= pg1x6;
442           a[0][3]= -g_1;
443           a[1][0]= pg2x4;
444           a[1][1]= pg2x5;
445           a[1][2]= pg2x6;
446           a[1][3]= -g_2;
447           a[2][0]= pg3x4;
448           a[2][1]= pg3x5;
449           a[2][2]= pg3x6;
450           a[2][3]= -g_3;
451        }
452
453          if (u1==0 && u2==0 && u3==1)
```

```
454         {
455            a[0][0]= pg1x4;
456            a[0][1]= pg1x5;
457            a[0][2]= -g_1;
458            a[1][0]= pg2x4;
459            a[1][1]= pg2x5;
460            a[1][2]= -g_2;
461         }
462
463           if (u1==0 && u2==1 && u3==0)
464         {
465            a[0][0]= pg1x4;
466            a[0][1]= pg1x6;
467            a[0][2]= -g_1;
468            a[1][0]= pg3x4;
469            a[1][1]= pg3x6;
470            a[1][2]= -g_3;
471         }
472
473           if (u1==0 && u2==1 && u3==1)
474         {
475            a[0][0]= pg1x4;
476            a[0][1]= -g_1;
477         }
478
479           if (u1==1 && u2==0 && u3==0)
480         {
481            a[0][0]= pg2x5;
482            a[0][1]= pg2x6;
483            a[0][2]= -g_2;
484            a[1][0]= pg3x5;
485            a[1][1]= pg3x6;
486            a[1][2]= -g_3;
487         }
488
489           if (u1==1 && u2==0 && u3==1)
490         {
491            a[0][0]= pg2x5;
492            a[0][1]= -g_2;
493         }
494
495           if (u1==1 && u2==1 && u3==0)
496         {
497            a[0][0]= pg3x6;
498            a[0][1]= -g_3;
499         }
500
501         GJ();
502
503         if (u1==0 && u2==0 && u3==0)
504         {
505            x4=X[3];
506            x5=X[4];
507            x6=X[5];
508            double absx4_x4a=fabs(x4-x4a);
509            double absx5_x5a=fabs(x5-x5a);
510            double absx6_x6a=fabs(x6-x6a);
511
512            if(absx4_x4a + absx5_x5a + absx6_x6a<= power){break;}
513         }
514
515         else if (u1==0 && u2==0 && u3==1)
516         {
517            x4=X[3];
518            x5=X[4];
519            double absx4_x4a=fabs(x4-x4a);
520            double absx5_x5a=fabs(x5-x5a);
521
522            if(absx4_x4a + absx5_x5a <= power){break;}
523         }
524
525         else if (u1==0 && u2==1 && u3==0)
526         {
527            x4=X[3];
528            x6=X[5];
529            double absx4_x4a=fabs(x4-x4a);
530            double absx6_x6a=fabs(x6-x6a);
531
532            if(absx4_x4a + absx6_x6a<= power){break;}
533         }
534         else if (u1==0 && u2==1 && u3==1)
535         {
536            x4=X[3];
537            double absx4_x4a=fabs(x4-x4a);
538
539            if(absx4_x4a <= power){break;}
540
541         }
```

```
542         else if (u1==1 && u2==0 && u3==0)
543         {
544            x5=X[4];
545            x6=X[5];
546            double absx5_x5a=fabs(x5-x5a);
547            double absx6_x6a=fabs(x6-x6a);
548
549            if(absx5_x5a + absx6_x6a<= power){break;}
550         }
551
552         else if (u1==1 && u2==0 && u3==1)
553         {
554            x5=X[4];
555            double absx5_x5a=fabs(x5-x5a);
556
557            if(absx5_x5a <= power){break;}
558         }
559         else if (u1==1 && u2==1 && u3==0)
560         {
561            x6=X[5];
562            double absx6_x6a=fabs(x6-x6a);
563            if(absx6_x6a<= power){break;}
564         }
565
566          if (j==99999) {printf("REVISA el MNR\n!!!!!!"); system("PAUSE");}
567      }
568  }
569
570  void GJ()
571  {
572     double G;
573     for(int k=1-(1); k<=n-(1); k++)
574     {
575        for(int i=1-(1); i<=n-(1); i++)
576        {
577           if (i!=k)
578           {
579              G= a[i][k]/a[k][k];
580              for (int j=k; j<=n+1-(1); j++)
581              {
582                 a[i][j]=a[i][j]-G*a[k][j];
583              }
584           }
585        }
586     }
587
588     double Z[n];
589     for(int i=1-(1); i<=n-(1); i++)
590     {
591        Z[i]=a[i][n+1-(1)]/a[i][i];
592     }
593
594
595     if (u1==0 && u2==0 && u3==0)
596     {
597        X[3]=X[3]+Z[0];
598        X[4]=X[4]+Z[1];
599        X[5]=X[5]+Z[2];
600     }
601     else if (u1==0 && u2==0 && u3==1)
602     {
603        X[3]=X[3]+Z[0];
604        X[4]=X[4]+Z[1];
605     }
606     else if (u1==0 && u2==1 && u3==0)
607     {
608        X[3]=X[3]+Z[0];
609        X[5]=X[5]+Z[1];
610     }
611     else if (u1==0 && u2==1 && u3==1)
612     {
613        X[3]=X[3]+Z[0];
614     }
615     else if (u1==1 && u2==0 && u3==0)
616     {
617        X[4]=X[4]+Z[0];
618        X[5]=X[5]+Z[1];
619     }
620     else if (u1==1 && u2==0 && u3==1)
621     {
622        X[4]=X[4]+Z[0];
623     }
624     else if (u1==1 && u2==1 && u3==0)
625     {
626        X[5]=X[5]+Z[0];
627     }
628  }
629
```

```
//-------------------------------------------------------H---------------
void H(double x1h, double x2h, double x3h, double x4h, double x5h, double x6h)
{
   double F_xh = F(x1h, x2h, x3h, x4h, x5h, x6h);
   double pfx1= derivada(F(x1h+dx1, x2h, x3h, x4h, x5h, x6h),F_xh, dx1);
   double pfx2= derivada(F(x1h, x2h+dx2, x3h, x4h, x5h, x6h),F_xh, dx2);
   double pfx3= derivada(F(x1h, x2h, x3h+dx3, x4h, x5h, x6h),F_xh, dx3);
   double pfx4= derivada(F(x1h, x2h, x3h, x4h+dx4, x5h, x6h),F_xh, dx4);
   double pfx5= derivada(F(x1h, x2h, x3h, x4h, x5h+dx5, x6h),F_xh, dx5);
   double pfx6= derivada(F(x1h, x2h, x3h, x4h, x5h, x6h+dx6),F_xh, dx6);

   if(u1==0 && u2==0 && u3==0)
   {
      MNR(x1h + dx1, x2h, x3h, x4h, x5h, x6h);
      double dx4dx1 = (X[3] - x4h)/dx1;
      double dx5dx1 = (X[4] - x5h)/dx1;
      double dx6dx1 = (X[5] - x6h)/dx1;
      X[3]= x4h; X[4]= x5h; X[5]= x6h;

      MNR(x1h, x2h + dx2, x3h, x4h, x5h, x6h);
      double dx4dx2 = (X[3] - x4h)/dx2;
      double dx5dx2 = (X[4] - x5h)/dx2;
      double dx6dx2 = (X[5] - x6h)/dx2;
      X[3]= x4h; X[4]= x5h; X[5]= x6h;

      MNR(x1h, x2h, x3h+dx3, x4h, x5h, x6h);
      double dx4dx3 = (X[3] - x4h)/dx3;
      double dx5dx3 = (X[4] - x5h)/dx3;
      double dx6dx3 = (X[5] - x6h)/dx3;
      X[3]= x4h; X[4]= x5h; X[5]= x6h;

      h1=-(pfx1 + pfx4*dx4dx1 + pfx5*dx5dx1 + pfx6*dx6dx1);
      h2=-(pfx2 + pfx4*dx4dx2 + pfx5*dx5dx2 + pfx6*dx6dx2);
      h3=-(pfx3 + pfx4*dx4dx3 + pfx5*dx5dx3 + pfx6*dx6dx3);
   }
   if(u1==0 && u2==0 && u3==1)
   {
      MNR(x1h + dx1, x2h, x3h, x4h, x5h, x6h);
      double dx4dx1 = (X[3] - x4h)/dx1;
      double dx5dx1 = (X[4] - x5h)/dx1;
      X[3]= x4h; X[4]= x5h;

      MNR(x1h, x2h + dx2, x3h, x4h, x5h, x6h);
      double dx4dx2 = (X[3] - x4h)/dx2;
      double dx5dx2 = (X[4] - x5h)/dx2;
      X[3]= x4h; X[4]= x5h;

      MNR(x1h, x2h, x3h+dx3, x4h, x5h, x6h);
      double dx4dx3 = (X[3] - x4h)/dx3;
      double dx5dx3 = (X[4] - x5h)/dx3;
      X[3]= x4h; X[4]= x5h;

      MNR(x1h, x2h, x3h, x4h, x5h, x6h+dx6);
      double dx4dx6 = (X[3] - x4h)/dx6;
      double dx5dx6 = (X[4] - x5h)/dx6;
      X[3]= x4h; X[4]= x5h;

      h1=-(pfx1 + pfx4*dx4dx1 + pfx5*dx5dx1);
      h2=-(pfx2 + pfx4*dx4dx2 + pfx5*dx5dx2);
      h3=-(pfx3 + pfx4*dx4dx3 + pfx5*dx5dx3);
      h6=-(pfx6 + pfx4*dx4dx6 + pfx5*dx5dx6);
   }
   if(u1==0 && u2==1 && u3==0)
   {
      MNR(x1h + dx1, x2h, x3h, x4h, x5h, x6h);
      double dx4dx1 = (X[3] - x4h)/dx1;
      double dx6dx1 = (X[5] - x6h)/dx1;
      X[3]= x4h; X[5]= x6h;

      MNR(x1h, x2h + dx2, x3h, x4h, x5h, x6h);
      double dx4dx2 = (X[3] - x4h)/dx2;
      double dx6dx2 = (X[5] - x6h)/dx2;
      X[3]= x4h; X[5]= x6h;

      MNR(x1h, x2h, x3h+dx3, x4h, x5h, x6h);
      double dx4dx3 = (X[3] - x4h)/dx3;
      double dx6dx3 = (X[5] - x6h)/dx3;
      X[3]= x4h; X[5]= x6h;

      MNR(x1h, x2h, x3h, x4h, x5h + dx5, x6h);
      double dx4dx5 = (X[3] - x4h)/dx5;
      double dx6dx5 = (X[5] - x6h)/dx5;
      X[3]= x4h; X[5]= x6h;

      h1=-(pfx1 + pfx4*dx4dx1 + pfx6*dx6dx1);
      h2=-(pfx2 + pfx4*dx4dx2 + pfx6*dx6dx2);
      h3=-(pfx3 + pfx4*dx4dx3 + pfx6*dx6dx3);
      h5=-(pfx5 + pfx4*dx4dx5 + pfx6*dx6dx5);
```

```
718     }
719     if(u1==0 && u2==1 && u3==1)
720     {
721        MNR(x1h + dx1, x2h, x3h, x4h, x5h, x6h);
722        double dx4dx1 = (X[3] - x4h)/dx1;
723        X[3]= x4h;
724
725        MNR(x1h, x2h + dx2, x3h, x4h, x5h, x6h);
726        double dx4dx2 = (X[3] - x4h)/dx2;
727        X[3]= x4h;
728
729        MNR(x1h, x2h, x3h+dx3, x4h, x5h, x6h);
730        double dx4dx3 = (X[3] - x4h)/dx3;
731        X[3]= x4h;
732
733        MNR(x1h, x2h, x3h, x4h, x5h + dx5, x6h);
734        double dx4dx5 = (X[3] - x4h)/dx3;
735        double dx6dx5 = (X[5] - x6h)/dx3;
736        X[3]= x4h;
737
738        MNR(x1h, x2h, x3h, x4h, x5h, x6h + dx6);
739        double dx4dx6 = (X[3] - x4h)/dx6;
740        double dx6dx6 = (X[5] - x6h)/dx6;
741        X[3]= x4h;
742
743        h1=-(pfx1 + pfx4*dx4dx1);
744        h2=-(pfx2 + pfx4*dx4dx2);
745        h3=-(pfx3 + pfx4*dx4dx3);
746        h5=-(pfx5 + pfx4*dx4dx5);
747        h6=-(pfx6 + pfx4*dx4dx6);
748     }
749     if(u1==1 && u2==0 && u3==0)
750     {
751        MNR(x1h + dx1, x2h, x3h, x4h, x5h, x6h);
752        double dx5dx1 = (X[4] - x5h)/dx1;
753        double dx6dx1 = (X[5] - x6h)/dx1;
754        X[4]= x5h; X[5]= x6h;
755
756        MNR(x1h, x2h + dx2, x3h, x4h, x5h, x6h);
757        double dx5dx2 = (X[4] - x5h)/dx2;
758        double dx6dx2 = (X[5] - x6h)/dx2;
759        X[4]= x5h; X[5]= x6h;
760
761        MNR(x1h, x2h, x3h+dx3, x4h, x5h, x6h);
762        double dx5dx3 = (X[4] - x5h)/dx3;
763        double dx6dx3 = (X[5] - x6h)/dx3;
764        X[4]= x5h; X[5]= x6h;
765
766        MNR(x1h, x2h, x3h, x4h + dx4, x5h, x6h);
767        double dx5dx4 = (X[4] - x5h)/dx4;
768        double dx6dx4 = (X[5] - x6h)/dx4;
769        X[4]= x5h; X[5]= x6h;
770
771        h1=-(pfx1 + pfx5*dx5dx1 + pfx6*dx6dx1);
772        h2=-(pfx2 + pfx5*dx5dx2 + pfx6*dx6dx2);
773        h3=-(pfx3 + pfx5*dx5dx3 + pfx6*dx6dx3);
774        h4=-(pfx4 + pfx5*dx5dx4 + pfx6*dx6dx4);
775     }
776     if(u1==1 && u2==0 && u3==1)
777     {
778        MNR(x1h + dx1, x2h, x3h, x4h, x5h, x6h);
779        double dx5dx1 = (X[4] - x5h)/dx1;
780        X[4]= x5h;
781
782        MNR(x1h, x2h + dx2, x3h, x4h, x5h, x6h);
783        double dx5dx2 = (X[4] - x5h)/dx2;
784        X[4]= x5h;
785
786        MNR(x1h, x2h, x3h+dx3, x4h, x5h, x6h);
787        double dx5dx3 = (X[4] - x5h)/dx3;
788        X[4]= x5h;
789
790        MNR(x1h, x2h, x3h, x4h + dx4, x5h, x6h);
791        double dx5dx4 = (X[4] - x5h)/dx4;
792        X[4]= x5h;
793
794        MNR(x1h, x2h, x3h, x4h, x5h, x6h + dx6);
795        double dx5dx6 = (X[4] - x5h)/dx6;
796        X[4]= x5h;
797
798        h1=-(pfx1 + pfx5*dx5dx1);
799        h2=-(pfx2 + pfx5*dx5dx2);
800        h3=-(pfx3 + pfx5*dx5dx3);
801        h4=-(pfx4 + pfx5*dx5dx4);
802        h6=-(pfx6 + pfx5*dx5dx6);
803     }
804     if(u1==1 && u2==1 && u3==0)
805     {
```

```
806         MNR(x1h + dx1, x2h, x3h, x4h, x5h, x6h);
807         double dx6dx1 = (X[5] - x6h)/dx1;
808          X[5]= x6h;
809 
810         MNR(x1h, x2h + dx2, x3h, x4h, x5h, x6h);
811         double dx6dx2 = (X[5] - x6h)/dx2;
812         X[5]= x6h;
813 
814         MNR(x1h, x2h, x3h+dx3, x4h, x5h, x6h);
815         double dx6dx3 = (X[5] - x6h)/dx3;
816         X[5]= x6h;
817 
818         MNR(x1h, x2h, x3h, x4h + dx4, x5h, x6h);
819         double dx6dx4 = (X[5] - x6h)/dx4;
820         X[5]= x6h;
821 
822         MNR(x1h, x2h, x3h, x4h, x5h + dx5, x6h);
823         double dx6dx5 = (X[5] - x6h)/dx5;
824         X[5]= x6h;
825 
826         h1=-(pfx1 + pfx6*dx6dx1);
827         h2=-(pfx2 + pfx6*dx6dx2);
828         h3=-(pfx3 + pfx6*dx6dx3);
829         h4=-(pfx4 + pfx6*dx6dx4);
830         h5=-(pfx5 + pfx6*dx6dx5);
831     }
832     if(u1==1 && u2==1 && u3==1)
833     {
834         h1=-(pfx1);
835         h2=-(pfx2);
836         h3=-(pfx3);
837         h4=-(pfx4);
838         h5=-(pfx5);
839         h6=-(pfx6);
840     }
841 }
842 
843 //-------------------------------------------------------------MGD-------------
844 
845 void MGD()
846 {
847     int it_en_MGD[6];
848     if (menos_puntos==1)
849         {
850               for (int i=0; i<6; i++)
851               if (G[i]==1)
852               if  (G[0]==1 || G[3]==1 || G[6]==1) it_en_MGD[0] = id[0] + 1;
853               if (G[1]==1 || G[4]==1 || G[7]==1)  it_en_MGD[1] = id[1] + 1;
854               if (G[2]==1 || G[5]==1 || G[8]==1)  it_en_MGD[2] = id[2] + 1;
855               if (G[0]==1 || G[1]==1 || G[2]==1)  it_en_MGD[3] = id[3] + 1;
856               if (G[3]==1 || G[4]==1 || G[5]==1)  it_en_MGD[4] = id[4] + 1;
857               if (G[6]==1 || G[7]==1 || G[8]==1)  it_en_MGD[5] = id[5] + 1;
858         }
859     else
860     {
861         it_en_MGD[0]= it + 1;
862     }
863     int K=3+u1+u2+u3;
864     if (u1+u2+u3<3)
865     {
866        MNR(X[0], X[1], X[2], X[3], X[4], X[5]);
867        asignar_datos(X,it_en_EGD);
868     }
869 
870     efe= F(X[0], X[1], X[2], X[3], X[4], X[5]);
871     if (eps<efe)
872      {
873        if (u1==0 && u2==0 && u3==0)
874          {
875           H(X[0], X[1], X[2], X[3],X[4], X[5]);
876           X[0]=X[0]+paso*h1;
877           X[1]=X[1]+paso*h2;
878           X[2]=X[2]+paso*h3;
879         }
880 
881         if (u1==0 && u2==0 && u3==1)
882           {
883           H(X[0], X[1], X[2], X[3],X[4], X[5]);
884           X[0]=X[0]+paso*h1;
885           X[1]=X[1]+paso*h2;
886           X[2]=X[2]+paso*h3;
887 
888           X[5]=X[5]+paso*h6;
889         }
890 
891         if (u1==0 && u2==1 && u3==0)
892           {
893           H(X[0], X[1], X[2], X[3],X[4], X[5]);
```

```
894             X[0]=X[0]+paso*h1;
895             X[1]=X[1]+paso*h2;
896             X[2]=X[2]+paso*h3;
897             X[4]=X[4]+paso*h5;
898         }
899
900         if (u1==0 && u2==1 && u3==1)
901           {
902             H(X[0], X[1], X[2], X[3],X[4], X[5]);
903             X[0]=X[0]+paso*h1;
904             X[1]=X[1]+paso*h2;
905             X[2]=X[2]+paso*h3;
906             X[4]=X[4]+paso*h5;
907             X[5]=X[5]+paso*h6;
908         }
909
910         if (u1==1 && u2==0 && u3==0)
911           {
912             H(X[0], X[1], X[2], X[3],X[4], X[5]);
913             X[0]=X[0]+paso*h1;
914             X[1]=X[1]+paso*h2;
915             X[2]=X[2]+paso*h3;
916             X[3]=X[3]+paso*h4;
917         }
918
919         if (u1==1 && u2==0 && u3==1)
920           {
921             H(X[0], X[1], X[2], X[3],X[4], X[5]);
922             X[0]=X[0]+paso*h1;
923             X[1]=X[1]+paso*h2;
924             X[2]=X[2]+paso*h3;
925             X[3]=X[3]+paso*h4;
926             X[5]=X[5]+paso*h6;
927         }
928
929         if (u1==1 && u2==1 && u3==0)
930           {
931             H(X[0], X[1], X[2], X[3],X[4], X[5]);
932             X[0]=X[0]+paso*h1;
933             X[1]=X[1]+paso*h2;
934             X[2]=X[2]+paso*h3;
935             X[3]=X[3]+paso*h4;
936             X[4]=X[4]+paso*h5;
937         }
938
939         if (u1==1 && u2==1 && u3==1)
940           {
941             H(X[0], X[1], X[2], X[3],X[4], X[5]);
942             X[0]=X[0]+paso1*h1;
943             X[1]=X[1]+paso2*h2;
944             X[2]=X[2]+paso3*h3;
945             X[3]=X[3]+paso4*h4;
946             X[4]=X[4]+paso5*h5;
947             X[5]=X[5]+paso6*h6;
948             asignar_datos(X,it_en_;MGD);
949             iteraciones=it;
950         }
951      }
952
953     else
954     {
955         iteraciones=it;
956         it=itmax + 100000;
957         if (u1==0 && u2==0 && u3==0 && menos_puntos==true)
958         {
959             if(eps>=efe)
960             {
961                 asignar_datos(X,it_en_MGD);
962             }
963         }
964           if (u1==1 && u2==1 && u3==1 && menos_puntos == true)
965           {
966             if(eps>=efe)
967             {
968                 asignar_datos(X,it_en_MGD);
969             }
970         }
971     }
972 }
```

La función que pasa la información de C++ a MATLAB se encuentra en las líneas de código 975 hasta la 1137.

```
975 void cpptomathlab()
976 {
977     if (juntas == 0)
```

```
978     {
979       engEvalString(ep, "Size=15");
980       engEvalString(ep, "Ancho=8");
981       engEvalString(ep, "ColorM=[0 0 0]");
982
983       if (u1==0 && u2==0 && u3==0)
984        {
985          engEvalString(ep, "COLOR=[0 0 0]");
986          engEvalString(ep, "estilo='-'");
987          engEvalString(ep, "Ancho=2.5");
988       }
989        else if (u1==0 && u2==0 && u3==1)
990        {
991          engEvalString(ep, "COLOR=[0 0 0]");
992           engEvalString(ep, "estilo='-'");
993           engEvalString(ep, "Ancho=2.5");
994       }
995       else if (u1==0 && u2==1 && u3==0)
996        {
997          engEvalString(ep, "COLOR=[0 0 0]");
998           engEvalString(ep, "estilo='-'");
999           engEvalString(ep, "Ancho=2.5");
1000      }
1001
1002      else if (u1==0 && u2==1 && u3==1)
1003      {
1004         engEvalString(ep, "COLOR=[0 0 0]");
1005          engEvalString(ep, "estilo='-'");
1006          engEvalString(ep, "Ancho=2.5");
1007      }
1008
1009      else if (u1==1 && u2==0 && u3==0)
1010      {
1011         engEvalString(ep, "COLOR=[0 0 0]");
1012            engEvalString(ep, "estilo='-'");
1013          engEvalString(ep, "Ancho=2.5");
1014      }
1015
1016      else if (u1==1 && u2==0 && u3==1)
1017      {
1018         engEvalString(ep, "COLOR=[0 0 0]");
1019         engEvalString(ep, "estilo='-'");
1020          engEvalString(ep, "Ancho=2.5");
1021      }
1022
1023      else if (u1==1 && u2==1 && u3==0)
1024      {
1025         engEvalString(ep, "COLOR=[0 0 0]");
1026            engEvalString(ep, "estilo='-'");
1027          engEvalString(ep, "Ancho=2.5");
1028      }
1029
1030      else if (u1==1 && u2==1 && u3==1)
1031      {
1032 //          engEvalString(ep, "COLOR=[1 0.3 0.7]");
1033          engEvalString(ep, "COLOR=[0 0 0]");
1034          engEvalString(ep, "estilo='-'");
1035          engEvalString(ep, "Ancho=2.5");
1036      }
1037   }
1038   else {
1039         engEvalString(ep, "Ancho=4"); engEvalString(ep, "Size=12"); engEvalString(ep, "ColorM=[0.3 0.3 0.3]");
1040       }
1041   mxArray *X1_array, *X2_array, *X3_array, *X4_array, *X5_array, *X6_array;
1042   mxArray *E_array =  mxCreateDoubleMatrix(Nci,1,mxREAL);
1043   mxArray *it_array = mxCreateDoubleMatrix(Nci,1,mxREAL);
1044   int s1,s2,s3,s4,s5,s6;
1045   if (menos_puntos==1) {s1=0;s2=1;s3=2;s4=3;s5=4;s6=5;}
1046   else {s1=s2=s3=s4=s5=s6=0;}
1047   if  (G[0]==1 || G[3]==1 || G[6]==1) X1_array = mxCreateDoubleMatrix(SizeArray[s1],1,mxREAL);
1048   if (G[1]==1 || G[4]==1 || G[7]==1)  X2_array = mxCreateDoubleMatrix(SizeArray[s2],1,mxREAL);
1049   if (G[2]==1 || G[5]==1 || G[8]==1)  X3_array = mxCreateDoubleMatrix(SizeArray[s3],1,mxREAL);
1050   if (G[0]==1 || G[1]==1 || G[2]==1)  X4_array = mxCreateDoubleMatrix(SizeArray[s4],1,mxREAL);
1051   if (G[3]==1 || G[4]==1 || G[5]==1)  X5_array = mxCreateDoubleMatrix(SizeArray[s5],1,mxREAL);
1052   if (G[6]==1 || G[7]==1 || G[8]==1)  X6_array = mxCreateDoubleMatrix(SizeArray[s6],1,mxREAL);
1053
1054   mxArray *nci_array = mxCreateDoubleMatrix(1,1,mxREAL);
1055   mxArray *estados_array = mxCreateDoubleMatrix(1,1,mxREAL);
1056   mxArray *cimax_array = mxCreateDoubleMatrix(1,1,mxREAL);
1057   mxArray *cimin_array = mxCreateDoubleMatrix(1,1,mxREAL);
1058   mxArray *conmutacion_array = mxCreateDoubleMatrix(1,1,mxREAL);
1059   mxArray *distanciaci_array = mxCreateDoubleMatrix(1,1,mxREAL);
1060
1061   double *pX1, *pX2, *pX3, *pX4, *pX5, *pX6;
1062   double *pE  = mxGetPr(E_array);
1063   double *pit = mxGetPr(it_array);
1064   if  (G[0]==1 || G[3]==1 || G[6]==1) pX1 = mxGetPr(X1_array);
1065   if (G[1]==1 || G[4]==1 || G[7]==1)  pX2 = mxGetPr(X2_array);
```

```
1066    if (G[2]==1 || G[5]==1 || G[8]==1)  pX3 = mxGetPr(X3_array);
1067    if (G[0]==1 || G[1]==1 || G[2]==1)  pX4 = mxGetPr(X4_array);
1068    if (G[3]==1 || G[4]==1 || G[5]==1)  pX5 = mxGetPr(X5_array);
1069    if (G[6]==1 || G[7]==1 || G[8]==1)  pX6 = mxGetPr(X6_array);
1070    double *pne = mxGetPr(nci_array);
1071    double *pestados = mxGetPr(estados_array);
1072    double *pcimax= mxGetPr(cimax_array);
1073    double *pcimin= mxGetPr(cimin_array);
1074    double *pconmutacion = mxGetPr(conmutacion_array);
1075    double *pdistanciaci= mxGetPr(distanciaci_array);
1076
1077    int k;
1078
1079    int f;
1080    if (menos_puntos==true) f=6;
1081    else f=1;
1082
1083    for (int i=0; i<f; i++)
1084       for (k=base;k<SizeArray[i];k++)
1085          {
1086             if  ((G[0]==1 || G[3]==1 || G[6]==1))  pX1[k]=datos0[k];
1087             if ((G[1]==1 || G[4]==1 || G[7]==1))   pX2[k]=datos1[k];
1088             if ((G[2]==1 || G[5]==1 || G[8]==1))   pX3[k]=datos2[k];
1089             if ((G[0]==1 || G[1]==1 || G[2]==1))   pX4[k]=datos3[k];
1090             if ((G[3]==1 || G[4]==1 || G[5]==1))   pX5[k]=datos4[k];
1091             if ((G[6]==1 || G[7]==1 || G[8]==1))   pX6[k]=datos5[k];
1092          }
1093
1094    for (k=0;k<Nci;k++)
1095    {
1096       pE[k]=E[k]/1;
1097       pit[k]=itera[k];
1098    }
1099
1100    pne[0]= ci;
1101    pestados[0]=estados;
1102    pcimax[0]= cimax;
1103    pcimin[0]= cimin;
1104    pdistanciaci[0]= distanciaci;
1105    if (conpco==0) pconmutacion[0]=ci+1;
1106    else pconmutacion[0]=contador+1;
1107
1108    if  (G[0]==1 || G[3]==1 || G[6]==1) engPutVariable(ep,"X1",X1_array);
1109    if (G[1]==1 || G[4]==1 || G[7]==1)  engPutVariable(ep,"X2",X2_array);
1110    if (G[2]==1 || G[5]==1 || G[8]==1)  engPutVariable(ep,"X3",X3_array);
1111    if (G[0]==1 || G[1]==1 || G[2]==1)  engPutVariable(ep,"X4",X4_array);
1112    if (G[3]==1 || G[4]==1 || G[5]==1)  engPutVariable(ep,"X5",X5_array);
1113    if (G[6]==1 || G[7]==1 || G[8]==1)  engPutVariable(ep,"X6",X6_array);
1114
1115    engPutVariable(ep,"nci",nci_array);
1116    engPutVariable(ep,"estados",estados_array);
1117    engPutVariable(ep,"it",it_array);
1118    engPutVariable(ep,"E",E_array);
1119    engPutVariable(ep,"cimin",cimin_array);
1120    engPutVariable(ep,"cimax",cimax_array);
1121    engPutVariable(ep,"distanciaci",distanciaci_array);
1122    engPutVariable(ep,"conmutacion",conmutacion_array);
1123
1124    engEvalString(ep,"for i=1:estados   X1(i, nci)=x1(i); X2(i, nci)=x2(i); X3(i, nci)=x3(i); X4(i, nci)=x4(i); X5
          (i, nci)=x5(i); X6(i, nci)=x6(i); end");
1125
1126    mxDestroyArray(nci_array);
1127    mxDestroyArray(estados_array);
1128    if  (G[0]==1 || G[3]==1 || G[6]==1) mxDestroyArray(X1_array);
1129    if (G[1]==1 || G[4]==1 || G[7]==1)  mxDestroyArray(X2_array);
1130    if (G[2]==1 || G[5]==1 || G[8]==1)  mxDestroyArray(X3_array);
1131    if (G[0]==1 || G[1]==1 || G[2]==1)  mxDestroyArray(X4_array);
1132    if (G[3]==1 || G[4]==1 || G[5]==1)  mxDestroyArray(X5_array);
1133    if (G[6]==1 || G[7]==1 || G[8]==1)  mxDestroyArray(X6_array);
1134    mxDestroyArray(cimin_array);
1135    mxDestroyArray(cimax_array);
1136    mxDestroyArray(conmutacion_array);
1137 }
```

La línea de código donde comienzan las funciones que se usan para crear los vectores de datos y para asignar la información a estos vectores, comienzan en la línea 1138 y en la 1149, respectivamente. La función utilizada para medir la distancia entra las variables nuevas de estado y las anteriores se encuentra desde la línea 1244; recuérdese que es utilizada para que la función *asignar_datos*() pueda determinar qué datos considerar y cuáles excluir, con el propósito de ahorrar espacio de memoria, en el caso de ser solicitado. Del renglón 1251 al 1259 se tienen instrucciones para liberar espacio de memoria, cuando los vectores *datos0, datos1,*

datos2, *datos3*, *datos4* y *datos5* ya puedan ser borrados.

```
1138 void crear_vectores_datos()
1139 {
1140    if  (G[0]==1 || G[3]==1 || G[6]==1) datos0 = new double[gato+1];
1141    if (G[1]==1 || G[4]==1 || G[7]==1)  datos1 = new double[gato+1];
1142    if (G[2]==1 || G[5]==1 || G[8]==1) datos2 = new double[gato+1];
1143    if  (G[0]==1 || G[1]==1 || G[2]==1) datos3 = new double[gato+1];
1144    if (G[3]==1 || G[4]==1 || G[5]==1) datos4 = new double[gato+1];
1145    if (G[6]==1 || G[7]==1 || G[8]==1)  datos5 = new double[gato+1];
1146    if  (G[0]==0 && G[1]==0 && G[2]==0 && G[3]==0 && G[4]==0 && G[5]==0 && G[6]==0 && G[7]==0 && G[8]==0) printf("
         No se ha solicitado graficar");
1147 }
1148
1149 void asignar_datos(double vdatos[],int coordenada[])
1150 {
1151    if (menos_puntos==false)
1152    {
1153       if  (G[0]==1 || G[3]==1 || G[6]==1) datos0[coordenada[0]]=vdatos[0];
1154       if (G[1]==1 || G[4]==1 || G[7]==1)  datos1[coordenada[0]]=vdatos[1];
1155       if (G[2]==1 || G[5]==1 || G[8]==1)  datos2[coordenada[0]]=vdatos[2];
1156       if (G[0]==1 || G[1]==1 || G[2]==1)  datos3[coordenada[0]]=vdatos[3];
1157       if (G[3]==1 || G[4]==1 || G[5]==1)  datos4[coordenada[0]]=vdatos[4];
1158       if (G[6]==1 || G[7]==1 || G[8]==1)  datos5[coordenada[0]]=vdatos[5];
1159    }
1160
1161    if (menos_puntos==true)
1162    {
1163       data0=data1=data2=data3=data4=data5=0;
1164       for (int i=0; i<9; i++)
1165       {
1166          if (G[i]==1)
1167          {
1168             int j;
1169             if(i<3) j=3;
1170             else if(i>=3 && i<6) j=4;
1171             else if(i>=6 && i<9) j=5;
1172
1173             int k;
1174              if(i==0 || i==3 || i==6) k=0;
1175             else if(i==1 || i==4 || i==7) k=1;
1176             else if(i==2 || i==5 || i==8) k=2;
1177                   double x_actual=X[k];
1178                   double y_actual=X[j];
1179
1180                   if(distancia(x_anterior[k],x_anterior[j],x_actual,y_actual)>=1e-10  ||  it== -1 ||  (it==pco &&
                          corridas==0))
1181                   {
1182                      x_anterior[k] = X[k];
1183                      x_anterior[j] = X[j];
1184                      if  ((G[0]==1 || G[3]==1 || G[6]==1))
1185                      {
1186                         if (data0==0)
1187                         {
1188                            datos0[coordenada[0]]=vdatos[0];
1189                            if (it!=-1) id[0]=id[0]+1;
1190                         }
1191                         data0++;
1192                      }
1193                      if ((G[1]==1 || G[4]==1 || G[7]==1))
1194                      {
1195                         if (data1==0)
1196                         {
1197                            datos1[coordenada[1]]=vdatos[1];
1198                            if (it!=-1) id[1]=id[1]+1;
1199                         }
1200                         data1++;
1201                      }
1202                      if ((G[2]==1 || G[5]==1 || G[8]==1))
1203                      {
1204                         if (data2==0)
1205                         {
1206                            datos2[coordenada[2]]=vdatos[2];
1207                            if (it!=-1) id[2]=id[2]+1;
1208                         }
1209                         data2++;
1210                      }
1211                      if ((G[0]==1 || G[1]==1 || G[2]==1))
1212                      {
1213                         if (data3==0)
1214                         {
1215                            datos3[coordenada[3]]=vdatos[3];
1216                            if (it!=-1) id[3]=id[3]+1;
1217                         }
1218                         data3++;
1219                      }
1220                      if ((G[3]==1 || G[4]==1 || G[5]==1))
1221                      {
```

```
1222                            if (data4==0)
1223                            {
1224                               datos4[coordenada[4]]=vdatos[4];
1225                               if (it!=-1) id[4]=id[4]+1;
1226                            }
1227                            data4++;
1228                         }
1229                         if ((G[6]==1 || G[7]==1 || G[8]==1))
1230                         {
1231                            if (data5==0)
1232                            {
1233                               datos5[coordenada[5]]=vdatos[5];
1234                               if (it!=-1) id[5]=id[5]+1;
1235                            }
1236                            data5++;
1237                         }
1238                      }
1239                   }
1240                }
1241             }
1242          }
1243
1244          double distancia(double x_1, double y_1, double x_2, double y_2)
1245          {
1246             double d;
1247             d=(x_1 - x_2)*(x_1 - x_2) + (y_1 - y_2)*(y_1 - y_2);
1248             return d;
1249          }
1250
1251          void liberar_espacio()
1252          {
1253             if  (G[0]==1 || G[3]==1 || G[6]==1) delete [] datos0;
1254             if (G[1]==1 || G[4]==1 || G[7]==1)  delete [] datos1;
1255             if (G[2]==1 || G[5]==1 || G[8]==1) delete [] datos2;
1256             if  (G[0]==1 || G[1]==1 || G[2]==1) delete [] datos3;
1257             if (G[3]==1 || G[4]==1 || G[5]==1) delete [] datos4;
1258             if (G[6]==1 || G[7]==1 || G[8]==1)  delete [] datos5;
1259          }
```

Con la función de la línea 1260, se grafican los ejes de las gráficas. Luego, con la función que se describe desde la línea 1268 hasta la 1385, se generan las trayectorias de las proyecciones de los retratos de fase. Después, con la función de la línea 1387 se genera la gráfica de comparación entre el número de iteraciones y las condiciones iniciales.

```
1260          void ejes()
1261          {
1262             engEvalString(ep, "hold on;");
1263             engEvalString(ep, "plot([-5,5], [0, 0],'LineWidth',2, 'color', 'black');");
1264             engEvalString(ep, "plot([0,0], [-5, 12],'LineWidth',2, 'color', 'black');");
1265          }
1266
1267
1268          void graficar()
1269          {
1270             engEvalString(ep, "hold on;");
1271
1272             if (G[0]==1)
1273             {
1274                engEvalString(ep, "figure(1);");
1275                ejes();
1276                engEvalString(ep, "grid on");
1277                engEvalString(ep, "line(X1, X4, 'color',COLOR, 'LineWidth',Ancho);");
1278                engEvalString(ep, "title('x_4 VS x_1','FontSize', 28);");
1279                engEvalString(ep, "xlabel ('x_1','FontSize', 24);");
1280                engEvalString(ep, "ylabel ('x_4','FontSize', 24);");
1281             }
1282
1283             if (G[1]==1)
1284             {
1285                engEvalString(ep, "figure(2);");
1286                ejes();
1287                engEvalString(ep, "grid on");
1288                engEvalString(ep, "line(X2, X4, 'color', COLOR, 'LineWidth',Ancho);");
1289                engEvalString(ep, "title('x_4 VS x_2','FontSize', 28);");
1290                engEvalString(ep, "xlabel ('x_2','FontSize', 24) ;");
1291                engEvalString(ep, "ylabel ('x_4','FontSize', 24) ;");
1292             }
1293
1294             if (G[2]==1)
1295             {
1296                engEvalString(ep, "figure(3);");
1297                ejes();
1298                engEvalString(ep, "grid on");
1299                engEvalString(ep, "line(X3, X4, 'color',COLOR, 'LineWidth',Ancho);");
```

```
1300        engEvalString(ep, "title('x_4 vs. x_3', 'FontSize', 28);");
1301        engEvalString(ep, "xlabel('x_3','FontSize', 24);");
1302        engEvalString(ep, "ylabel('x_4','FontSize', 24);");
1303    }
1304
1305    if (G[3]==1)
1306    {
1307        engEvalString(ep, "figure(4);");
1308        ejes();
1309        engEvalString(ep, "grid on");
1310        engEvalString(ep, "line(X1, X5, 'color',COLOR, 'LineWidth',Ancho);");
1311        engEvalString(ep, "title('x_5 VS x_1','FontSize', 28);");
1312        engEvalString(ep, "xlabel('x_1','FontSize', 24);");
1313        engEvalString(ep, "ylabel('x_5','FontSize', 24);");
1314    }
1315
1316    if (G[4]==1)
1317    {
1318        engEvalString(ep, "figure(5);");
1319        ejes();
1320        engEvalString(ep, "grid on");
1321        engEvalString(ep, "line(X2, X5, 'color', COLOR, 'LineWidth',Ancho);");
1322        engEvalString(ep, "title('x_5 VS x_2','FontSize', 28);");
1323        engEvalString(ep, "xlabel('x_2','FontSize', 24);");
1324        engEvalString(ep, "ylabel('x_5','FontSize', 24);");
1325    }
1326
1327    if (G[5]==1)
1328    {
1329        engEvalString(ep, "figure(6);");
1330        ejes();
1331        engEvalString(ep, "grid on");
1332        engEvalString(ep, "line(X3, X5, 'color',COLOR, 'LineWidth',Ancho);");
1333        engEvalString(ep, "title('x_5 vs. x_3', 'FontSize', 28);");
1334        engEvalString(ep, "xlabel('x_3','FontSize', 24);");
1335        engEvalString(ep, "ylabel('x_5','FontSize', 24);");
1336    }
1337
1338    if (G[6]==1)
1339    {
1340        engEvalString(ep, "figure(7);");
1341        ejes();
1342        engEvalString(ep, "grid on");
1343        engEvalString(ep, "line(X1, X6, 'color',COLOR, 'LineWidth',Ancho);");
1344        engEvalString(ep, "title('x_6 VS x_1','FontSize', 28);");
1345        engEvalString(ep, "xlabel('x_1','FontSize', 24);");
1346        engEvalString(ep, "ylabel('x_6','FontSize', 24);");
1347    }
1348
1349    if (G[7]==1)
1350    {
1351        engEvalString(ep, "figure(8);");
1352        ejes();
1353        engEvalString(ep, "grid on");
1354        engEvalString(ep, "line(X2, X6, 'color', COLOR, 'LineWidth',Ancho);");
1355        engEvalString(ep, "title('x_6 VS x_2', 'FontSize', 28);");
1356        engEvalString(ep, "xlabel('x_2','FontSize', 24);");
1357        engEvalString(ep, "ylabel('x_6','FontSize', 24);");
1358    }
1359
1360    if (G[8]==1)
1361    {
1362        engEvalString(ep, "figure(9);");
1363        ejes();
1364        engEvalString(ep, "grid on");
1365        engEvalString(ep, "line(X3, X6, 'color',COLOR, 'LineWidth',Ancho,'LineStyle',estilo);");
1366        engEvalString(ep, "title('x_6 vs. x_3', 'FontSize', 28);");
1367        engEvalString(ep, "xlabel('x_3','FontSize', 24);");
1368        engEvalString(ep, "ylabel('x_6','FontSize', 24);");
1369    }
1370
1371    engEvalString(ep, "nconmu(conmutacion)=conmutacion;");
1372
1373    if (conpco>=1)
1374    {
1375        engEvalString(ep, "figure(30);");
1376        engEvalString(ep, "hold on;");
1377        engEvalString(ep, "plot(nconmu, itera);");
1378        engEvalString(ep, "hold on;");
1379        engEvalString(ep, "plot(conmutacion, it, '.','MarkerSize',20, 'MarkerEdgeColor', 'red');");
1380        engEvalString(ep, "title('NMERO DE PASOS VS NMERO DE CONMUTACIN');");
1381        engEvalString(ep, "xlabel 'NUMERO DE CONMUTACIN';");
1382        engEvalString(ep, "ylabel 'NMERO DE PASOS'");
1383        engEvalString(ep, "hold off;");
1384    }
1385 }
1386
1387 void graficar_2()
```

```
1388 {
1389     engEvalString(ep, "figure (20);");
1390     engEvalString(ep, "line(cimax:-distanciaci:cimin, it);");
1391     engEvalString(ep, "line(cimax:-distanciaci:cimin, it, 'Marker','.','MarkerSize',20, 'MarkerEdgeColor','red');
          ");
1392     engEvalString(ep, "title('ITERATIONS VS STARTING CONDITION FOR x 2','FontSize',14);");
1393     engEvalString(ep, "xlabel ('STARTING CONDITION FOR x 2','FontSize',14);");
1394     engEvalString(ep, "ylabel ('NUMBER OF ITERATIONS','FontSize',14)");
1395 }
```

La función para graficar los puntos iniciales, se describe desde la línea 1396 hasta la 1443. Por último, la función que sirve para marcar los puntos finales o cruces sobre el final de las trayectorias, se define desde el renglón 1445 hasta el 1497.

```
1396 void puntosiniciales()
1397 {
1398       if (G[0]==1)
1399       {
1400          engEvalString(ep, "figure(1);");
1401          engEvalString(ep, "line(X1(1), X4(1),'Marker','o', 'LineWidth',2.5, 'MarkerSize',Size, 'MarkerEdgeColor
                ', 'g')");
1402       }
1403       if (G[1]==1)
1404       {
1405          engEvalString(ep, "figure(2);");
1406          engEvalString(ep, "line(X2(1), X4(1),'Marker','o', 'LineWidth',2.5, 'MarkerSize',Size, 'MarkerEdgeColor
                ', 'g')");
1407       }
1408       if (G[2]==1)
1409       {
1410          engEvalString(ep, "figure(3);");
1411          engEvalString(ep, "line(X3(1), X4(1),'Marker','o', 'LineWidth',2.5, 'MarkerSize',Size, 'MarkerEdgeColor
                ', 'g')");
1412       }
1413       if (G[3]==1)
1414       {
1415          engEvalString(ep, "figure(4);");
1416          engEvalString(ep, "line(X1(1), X5(1),'Marker','o', 'LineWidth',2.5, 'MarkerSize',Size, 'MarkerEdgeColor
                ', 'g')");
1417       }
1418       if (G[4]==1)
1419       {
1420          engEvalString(ep, "figure(5);");
1421          engEvalString(ep, "line(X2(1), X5(1),'Marker','o', 'LineWidth',2.5, 'MarkerSize',Size, 'MarkerEdgeColor
                ', 'g')");
1422       }
1423       if (G[5]==1)
1424       {
1425          engEvalString(ep, "figure(6);");
1426          engEvalString(ep, "line(X3(1), X5(1),'Marker','o', 'LineWidth',2.5, 'MarkerSize',Size, 'MarkerEdgeColor
                ', 'g')");
1427       }
1428       if (G[6]==1)
1429       {
1430          engEvalString(ep, "figure(7);");
1431          engEvalString(ep, "line(X1(1), X6(1),'Marker','o', 'LineWidth',2.5, 'MarkerSize',Size, 'MarkerEdgeColor
                ', 'g')");
1432       }
1433       if (G[7]==1)
1434       {
1435          engEvalString(ep, "figure(8);");
1436          engEvalString(ep, "line(X2(1), X6(1),'Marker','o', 'LineWidth',2.5, 'MarkerSize',Size, 'MarkerEdgeColor
                ', 'g')");
1437       }
1438       if (G[8]==1)
1439       {
1440          engEvalString(ep, "figure(9);");
1441          engEvalString(ep, "line(X3(1), X6(1),'Marker','o', 'LineWidth',1, 'MarkerSize',Size, 'MarkerEdgeColor', 
                [0 0 0])");
1442       }
1443 }
1444
1445 void puntosfinales()
1446 {
1447       if (G[0]==1 || G[1]==1 || G[2]==1)  engEvalString(ep, "lastpoint = size(X4); lastpoint = lastpoint(1,1);");
1448       if (G[0]==1)
1449       {
1450          engEvalString(ep, "figure(1);");
1451          engEvalString(ep, "line(X1(lastpoint), X4(lastpoint), 'color', [1 0 1], 'Marker','x', 'LineWidth',2.5, '
                MarkerSize',Size, 'MarkerEdgeColor', ColorM)");
1452       }
1453       if (G[1]==1)
1454       {
1455          engEvalString(ep, "figure(2);");
```

```
1456            engEvalString(ep, "line(X2(lastpoint), X4(lastpoint), 'color', [1 0 1], 'Marker','x','LineWidth',2.5,'
                    MarkerSize',Size,'MarkerEdgeColor',ColorM)");
1457        }
1458        if (G[2]==1)
1459        {
1460            engEvalString(ep, "figure(3);");
1461            engEvalString(ep, "line(X3(lastpoint), X4(lastpoint), 'color', [1 0 1], 'Marker','x','LineWidth',2.5,'
                    MarkerSize',Size,'MarkerEdgeColor',ColorM)");
1462        }
1463
1464        if (G[3]==1 || G[4]==1 || G[5]==1)  engEvalString(ep, "lastpoint = size(X5); lastpoint = lastpoint(1,1);");
1465        if (G[3]==1)
1466        {
1467            engEvalString(ep, "figure(4);");
1468            engEvalString(ep, "line(X1(lastpoint), X5(lastpoint), 'color', [1 0 1], 'Marker','x','LineWidth',2.5,'
                    MarkerSize',Size,'MarkerEdgeColor',ColorM)");
1469        }
1470        if (G[4]==1)
1471        {
1472            engEvalString(ep, "figure(5);");
1473            engEvalString(ep, "line(X2(lastpoint), X5(lastpoint), 'color', [1 0 1], 'Marker','x','LineWidth',2.5,'
                    MarkerSize',Size,'MarkerEdgeColor',ColorM)");
1474        }
1475        if (G[5]==1)
1476        {
1477            engEvalString(ep, "figure(6);");
1478            engEvalString(ep, "line(X3(lastpoint), X5(lastpoint), 'color', [1 0 1], 'Marker','x','LineWidth',2.5,'
                    MarkerSize',Size,'MarkerEdgeColor',ColorM)");
1479        }
1480
1481        if (G[6]==1 || G[7]==1 || G[8]==1)  engEvalString(ep, "lastpoint = size(X6); lastpoint = lastpoint(1,1);");
1482        if (G[6]==1)
1483        {
1484            engEvalString(ep, "figure(7);");
1485            engEvalString(ep, "line(X1(lastpoint), X6(lastpoint), 'color', [1 0 1], 'Marker','x','LineWidth',2.5,'
                    MarkerSize',Size,'MarkerEdgeColor',ColorM)");
1486        }
1487        if (G[7]==1)
1488        {
1489            engEvalString(ep, "figure(8);");
1490            engEvalString(ep, "line(X2(lastpoint), X6(lastpoint), 'color', [1 0 1], 'Marker','x','LineWidth',2.5,'
                    MarkerSize',Size,'MarkerEdgeColor',ColorM)");
1491        }
1492        if (G[8]==1)
1493        {
1494            engEvalString(ep, "figure(9);");
1495            engEvalString(ep, "line(X3(lastpoint), X6(lastpoint), 'color', [1 0 1], 'Marker','x','LineWidth',1,'
                    MarkerSize',Size,'MarkerEdgeColor',ColorM)"); //CAmbi LineWidth a 5
1496        }
1497 }
```

Apéndice F

Evidencia de publicaciones

Participé en el Congreso Internacional de Investigación de Academia Journals Puebla 2020 y en la XVI Semana Nacional de Ingeniería Electrónica y I Semana Iberoamericana de Ingeniería Electrónica: SENIE 2020. Presenté un artículo para cada uno de los dos eventos. El primer artículo ya fue publicado en el portal de internet AcademiaJournals.com y el segundo será publicado en la revista **Pistas Educativas**, entre el mes de enero y febrero de 2021.

ACADEMIA JOURNALS

OPUS PRO SCIENTIA ET STUDIUM

Congreso Internacional de Investigación de
Academia Journals Puebla 2020

Certificado

Otorgado a

Jorge Espinosa García
Alexandre Zemliak

por su artículo titulado

The study of the trajectories of the design process of electronic circuits

(Artículo PP135)

El artículo fue presentado en el Congreso Internacional de Academia Journals Puebla 2020 que tuvo lugar los días 13 y 14 de agosto de 2020 y que fue organizado en colaboración con la Universidad IEU en Puebla, Pue., México.
El artículo fue publicado en el portal de Internet AcademiaJournals.com en las siguientes modalidades: (1) volúmenes con **ISSN 1946-5351** Vol. 12, No. 5, 2020 online e indexación en la base de datos *Fuente Académica Plus* de EBSCOhost y (2) libro online *ebook* con ISBN 978-1-939982-55-1, mismo que lleva por título *Investigación en la Educación Superior - Puebla 2020*.

Dr. Rafael Moras
Jefe Comité de Programa del Congreso
Editor, Academia Journals

UNIVERSIDAD AUTONOMA METROPOLITANA
Casa abierta al tiempo Azcapotzalco

XVI Semana Nacional de Ingeniería Electrónica
SENIE 2020
I Semana Iberoamericana de Ingeniería Electrónica

DIVISION DE CIENCIAS BASICAS E INGENIERIA
CBI UAM-Azcapotzalco

La Universidad Autónoma Metropolitana Unidad Azcapotzalco a través de la

División de Ciencias Básicas e Ingeniería

Otorga la presente

CONSTANCIA A:

Jorge Espinosa García, Alexander Zemliak, Fernando Reyes Cortés

Por la presentación del artículo titulado:

TRAJECTORY ANALYSIS FOR THE DIFFERENT STRATEGIES OF CIRCUIT DESIGN

En la XVI Semana Nacional de Ingeniería Electrónica y I Semana Iberoamericana de Ingeniería Electrónica

Ciudad de México a 25 de Noviembre de 2020

Dra. Teresa Merchand Hernández
Directora de la División de CBI
Universidad Autónoma Metropolitana
Unidad Azcapotzalco

Dr. Miguel Magos Rivera
Presidente del Comité Organizador
Senie 2020

Dr. José Raúl Miranda Tello
Jefe de Departamento de Electrónica
Universidad Autónoma Metropolitana
Unidad Azcapotzalco

Bibliografía

Baeyens, E., Herreros, A. & Perán, J. R. (2016). A Direct Search Algorithm for Global Optimization [Artículo. Editor académico: George Karakostas]. MDPI.

Belaire, M. B. F. (2009). *Programación matemática para la economía y la empresa* (1era.) [Pág. 28]. Universitat de València.

Bunch, J. & D.J. Rose, E. (1976). Sparse Matrix Computations. Acad. Press, N.Y.

Castillo, E., Conejo, A. J., Pedregal, P., García, R. & Alguacil, N. (2002). *Formulación y Resolución de Modelos de Programación Matemática en Ingeniería y Ciencia* (1era.) [Pág. 75]. Universidad de Castilla - La Mancha: Escuela Técnica Superior de Ingenieros Industriales, Escuela Técnica Superior de Ingenieros de Caminos, Canales y Puertos.

Clapp, M. (2015). *Análisis matemático* (1era.) [Págs. 8, 75, 77-78, 101-102]. Instituto de Matemáticas, UNAM.

Cottle, R. W. & Thapa, M. N. (2017). *Linear and nonlinear optimization*. Springer.

cplusplus.com. (2020). *function clock* [© cplusplus.com, 2000-2020 - All rights reserved - v3.2//Último acceso 06 de Enero de 2021]. © cplusplus.com, 2000-2020 - All rights reserved - v3.2//Último acceso 06 de Enero de 2021. http://www.cplusplus.com/reference/ctime/clock/

Dantzig, G. B. & Thapa, M. N. (1997). *Linear programming 1 : Introduction* (1era.) [Pág. XXXIV]. Springer-Verlag New York.

Duff, I. & Reid, J. (1979). Some Design Features of a Sparse Matrix Code [vol. 5, no. 1, pages 18-35.]. ACM Trans. on Mathematical Software.

Fletcher, R. (1981). Practical Methods of Optimization [vol. 1, 1980, vol. 2, 1981]. John Wiley & Sons, N.Y.

Flores, W. M. (2010). *Introducción a los Métodos Numéricos* (1era.) [Pág. 200]. Escuela de Matemática,Instituto Tecnológico de Costa Rica.

Gerald, C. F. & Wheatley, P. O. (2004). *Applied Numerical Analysis* (Séptima) [Págs. 335, 428]. Pearson Education, Inc.

Guzmán, I. R. V. (2000). Algoritmos de Filtrado y Control Óptimo en Sistemas de Tipo Volterra con Incertidumbres Determinísticas [Tesis de maestría]. Universisdad Autónoma de Nuevo León, San Nicolás de los Garza N. L.

Henrici, P. (1964). *Elements of numerical analysis* [Págs. 99, 267]. John Wiley & Sons, Inc.

James, G. (2015). *Modern Engineering Mathematics* (Fifth) [Págs. 829-830]. Pearson Education.

James, G. & Dyke, P. (2018). *Advance Modern Engineering Mathematics* (Fifth) [Págs. 740, 781]. Pearson Education.

Karloff, H. (1961). *Linear programming* (1era.) [Pág. 5]. Birkhäuser Boston.

Kashirskiy, I. (1976). General Optimization Methods [vol. 19, no. 6, págs. 21-25]. Izvestiya VUZ USSR - Radioelectronica.

Kashirskiy, I. & Trokhimenko, I. (1979). General Optimization for Electronic Circuits. Tekhnika, Kiev.

Massara, R. E. (1991). *Optimization methods in electronic circuit desing* (First). Longman Scientific; Technical.

Massobrio, G. & Antognetti, P. (1993). *Semiconductor Device Modeling With SPICE* (Segunda). McGraw Hill, Inc.

Mastorakis, N. & Sakellaris, J. (2009). *Advances in Numerical Methods* (1era.). Springer.

Miranda, M. J. & Fackler, P. L. (2002). *Applied Computational Economics and Finance* [Págs. 33-34]. Massachusetts Institute of Technology.

Neumann, E. (2001). Runge-Kutta Algorithm [Contenido supervisado por la página web el 5 de febrero de 2018. //Último acceso 19 de Noviembre de 2020]. https://www.myphysicslab.com/explain/runge-kutta-en.html

Nieves, A. & Domínguez, F. C. (2014). *Métodos Numéricos Aplicados a la Ingeniería* (Cuarta) [Págs. 51, 295, 309-310, 538-541, 551-552]. Grupo Editorial Patria.

Ochotta, E., R.A.Rutenbar & Carley, L. (1996). Synthesis of High-Performance Analog Circuits in ASTRX/OBLX [vol. 15, no. 3, págs. 273-294]. IEEE Trans. CAD.

Osterby, O. & Zlatev, Z. (1983). Direct Methods for Sparse Matrices. Springer-Verlag, N.Y.

R. K. Brayton, A. L. S.-V., G. D. Hachtel. (1981). A survey of optimization techniques for integrated-circuit design [vol. 69, no. 10, pp. 1334-1362]. IEEE Proc.

Rabat, N., Ruehli, A., Mahoney, G. & Coleman, J. (1985). A Survey of Macromodeling [Abril 1985, págs. 139-143, 428]. IEEE Int. Symp. Circuits Systems.

Rao, S. S. (1996). *Engineering Optimization: Theory and Practice* (Third). New Age International (P) Limited Publishers.

Rizzoli, V., Costanzo, A. & Cecchetti, C. (1990). Numerical optimization of broadband nonlinear microwave circuits [vol. 1, págs. 335-338]. IEEE MTT-S Int. Symp.

Ruehli, A., Sangiovanni-Vincentelli, A. & Rabbat, G. (1982). Time Analysis of Large-Scale Circuits Containing One-Way Macromodels [vol. CAS-29, no. 3, págs. 185-191.]. IEEE Trans. Circuits Syst.

Sangiovanni-Vincentelli, A., Chen, L. & Chua, L. (1977). An Efficient Cluster Algorithm for Tearing Large-Scale Networks [vol. CAS-24, no. 12, págs. 709- 717]. IEEE Trans. Circuits Syst.

Taha, H. A. (2012). *Investigación de operaciones* (9na.) [Págs. 14-15]. Pearson Educación.

Wang, R. (2015). Fixed points [Último acceso 13 de Noviembre de 2020]. http://fourier.eng.hmc.edu/e176/lectures/NM/node17.html

Wu, F. (1976). Solution of Large-Scale Networks by Tearing [vol. CAS-23, no. 12, págs. 706-713.]. IEEE Trans. Circuits Syst.

Zemliak, A. (1999). General Methodology for System Design, Chapter of the book "Modern Applied Mathematics Techniques in Circuits, Systems and Control" [págs. 150-155]. Ed. N. Mastorakis, WSES Press.

Zemliak, A. (2001). Analog System Design Problem Formulation by Optimum Control Theory [vol. E84-A, No. 8, pp. 2029-2041.]. IEICE Transactions on Fundamentals of Electronics, Communications; Computer Sciences.

Zemliak, A. (2002). Novel Approach to the Time-Optimal System Design Methodology [vol. 1, no. 2, 2002, págs. 222-231]. WSEAS Trans. Syst.

Zemliak, A. (2004). Main Properties Study of the Time-Optimal System Design Algorithm [vol. 3, no. 4, págs. 759-764]. WSEAS Trans. Circuits Syst.

Zemliak, A. (2014). Analog circuit optimization on basis of control theory approach [vol. 33, no. 6, págs. 2180-2204]. COMPEL: The International Journal for Computation; Mathematics in Electrical; Electronic Engineering.