FACULTAD DE INGENIERÍA ENSENADA

PROGRAMA DE POSGRADO
EN CIENCIAS E INGENIERÍA

TESIS

Toolbox para el análisis de sistemas no lineales

Presenta:
Roberto Andrés Camacho Pérez

Director

Dr. Manuel Moisés Miranda Velasco

Ensenada, Baja California, México. Diciembre de 2010

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

FACULTAD DE INGENIERÍA
UNIDAD ENSENADA

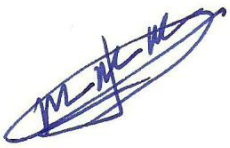
"Toolbox para el análisis de sistemas no lineales"

TESIS

Que para obtener el grado de maestría en ingeniería presenta:

Roberto Andrés Camacho Pérez

Aprobada por:



Dr. Manuel Moisés Miranda Velazco
Director de tesis

M.I. Víctor Rafael Velázquez Mejía
Miembro del comité

Dr. José de Jesús Zamarripa Topete
Miembro del comité

Ensenada Baja California, México. Noviembre 2010

RESUMEN

El avance de la tecnología y los nuevos lenguajes de programación, hacen una combinación perfecta para desarrollar tecnología automatizada, capaz de realizar trabajos manuales en poco tiempo y así, permitir al investigador ocuparse en otro tipo de labores. Este trabajo presenta el desarrollo de una aplicación computacional para el análisis de sistemas no lineales, en el cual se utilizan las herramientas Matlab y el lenguaje de programación que trabaja en diferentes sistemas operativos como lo es Java, ampliamente utilizados en el ámbito científico. Ambos se interconectan a través de las clases generadas por la herramienta de Matlab llamada Deployment Tool y se logra un sistema distribuible sin la necesidad de tener instalado Matlab.

Se utilizó la ecuación de Lorenz para demostrar este trabajo, y se comparó con otros programas que utilizan los investigadores como lo es Matlab, Dinamics Solver, XPPAUT y Simnom. Como resultado muestra una gráfica idéntica en el primer programa, en los varían, ya que el intervalo de solución es de un rango mayor, sin embargo el resultado es cualitativamente igual.

Toolbox para el análisis de sistemas no lineales es una aplicación basada en sistemas Windows XP, Vista y Seven, de 32 y 64 bits. Con la posibilidad de migrarlo al sistema operativo Linux y Mac OSX.

DEDICATORIAS

A mis padres:

Armando Camacho Camacho y María Acela Pérez González.

A mis hermanos:

Armando, José Luis y Sandy.

A mi sobrina:

Katherine Mouett Camacho

AGRADECIMIENTOS

Agradezco sinceramente:

- A mis padres por el apoyo que me han brindado a lo largo de toda mi vida, por su cariño y su amor incondicional, esto es de ustedes. De nuevo Gracias.
- A mis hermanos, por confiar en mí y por su ayuda que siempre tuve a lo largo de mi carrera.
- Al Dr. Manuel Moisés Miranda Velasco, por confiar en mí, por ser un gran maestro y por su apoyo a lo largo de esta tesis.
- Al Dr. José de Jesús Zamarripa Topete, por compartirme sus conocimientos y no dejar que me perdiera en la trayectoria de este trabajo.
- Al Maestro Víctor R. Velázquez Mejía, por ayudarme a encontrar un camino para la realización de este sistema y por estar siempre dispuesto.
- A mis compañeros Gaby, Abraham, Eduardo, Mary, Javier, Memo y por último pero no menos importante Arturo, gracias por todo su apoyo.
- A mi novia Erika, por tu paciencia y apoyo.
- Al Consejo Nacional de Ciencia y Tecnología (CONACyT), por el apoyo económico brindado.

CONTENIDO

Página

CAPÍTULO I

Página

CAPÍTULO II

Página

CAPÍTULO III

Página

CAPÍTULO IV

CAPÍTULO V

Página

LISTADO DE FIGURAS

Página

Página

LISTADO DE TABLAS

Página

CAPÍTULO I

INTRODUCCIÓN

1.1. Antecedentes

La tecnología moderna y las computadoras han cambiado la forma de resolver las dificultades en el mundo actual.

La ingeniería en computación representa uno de los campos de la ingeniería que más ha evolucionado en los últimos años. Son innumerables las innovaciones que ha habido para realizar nuevos sistemas que permitan, tanto a las personas como a las instituciones, realizar satisfactoriamente sus actividades.

Desde la primera generación de computadoras donde su construcción se basaba en bulbos y los dispositivos de almacenamiento eran bastante limitados, su programación se realizaba mediante tarjetas perforadas en lenguajes de bajo nivel, las aplicaciones primordiales se limitaban a cálculos matemáticos simples. Conforme evolucionó la tecnología, las computadoras fueron capaces de almacenar más información, ejecutar programas más eficientes, en tiempo y costos mucho menor. Hoy en día la tecnología forma parte de la mayoría de las personas, tanto en comunicaciones, seguridad, educación, entre otros. Cuanto más crece la necesidad de tecnología dentro de la sociedad se requiere de otras

alternativas, que tal vez ya están presentes y su estudio no es nuevo, como es el caso de la aplicación de la computación en los sistemas de análisis no lineales. Para entender lo que es un sistema no lineal, el siguiente párrafo presenta dos problemas de este tipo: el caos y atractores extraños.

En la sociedad actual, se puede encontrar que las personas hablan de términos provenientes de literatura científica, por ejemplo: caos, atractores extraños, efecto mariposa, impredecibilidad del tiempo atmosférico, etc. Hasta se ha hecho cine con estos temas como Chaos, de los hermanos Tavianni, Parque Jurásico de Steven Spielberg y Efecto Mariposa de Fernando Colomo. También se puede encontrar estos términos en la literatura, ejemplos: El cuento, ¿El aleteo de una mariposa en Nueva York puede provocar un tifón en Pekín? Está incluido en el libro de cuentos L'ángelo Nero (1991) de Antonio Tabucchi, aunque a decir verdad, resulta complicado encontrar relaciones entre el mismo relato y lo que su título significa. En el libro A Sound of Thunder de Ray Bradbury se plantea una curiosa historia. La muerte de una mariposa prehistórica, con su consiguiente falta de descendencia, cambia el resultado de la elección presidencial en Estados Unidos, en el momento presente. En la novela Storm de George R. Stewart, un meteorólogo recuerda el comentario de uno de sus profesores acerca de que, un hombre que estornudara en China podría dar lugar a que la gente tuviera que quitar la nieve con palas en la ciudad de New York.

Actualmente tanto en inglés como en castellano, la palabra caos (chaos en inglés) tiene dos significados:

(a) Estado amorfo e indefinido que se supone anterior a la constitución del cosmos.

(b) Confusión, desorden.

Las ideas sobre los atractores extraños son ideas novedosas de los citados años 70, por lo tanto se ha impulsado una forma nueva de abordar un problema muy importante en Física y Matemáticas como es el problema de la turbulencia en los fluidos.

Por lo tanto ¿Qué es un sistema no lineal?

Un sistema físico, matemático o de otro tipo es no lineal cuando, las ecuaciones de movimiento, evolución o de comportamiento no cumplen el teorema de superposición. En particular, el movimiento de sistemas no lineales se puede descomponer en dos o más sub problemas más sencillos de tal manera que el problema original se obtiene como “superposición” o “suma” de estos sub problemas más sencillos. Algunos sistemas no lineales tienen soluciones exactas o integrables, mientras que otros tienen comportamiento caótico, por lo tanto no se pueden reducir a una forma simple ni se pueden resolver. Aunque algunos sistemas no lineales y ecuaciones de interés general han sido extensamente estudiados, la vasta mayoría son pobremente comprendidos.

Los sistemas no lineales surgen hace poco menos de siglo y medio cuando Henri Poincaré demostró que el problema gravitacional de los tres cuerpos (Sol, Luna y Tierra) mencionado por Isaac Newton, no tiene solución (1889), ya que las ecuaciones de movimiento son no lineales [1].

A principios de 1952, el físico Enrico Fermi, con la ayuda de John Pasta, especialista en computación y el matemático Stanislaw Ulam, realizaron un experimento en el cual eligieron un conjunto de N = 32 masas puntuales iguales que pueden moverse a lo largo de una recta. Cada masa está unida a la anterior mediante un resorte y los extremos de esta cadena están fijos. Los resortes fueron considerados como cuasi ideales, la colocar una fuerza lineal, característica de resortes que se comportan idealmente como osciladores armónicos, una pequeña fuerza perturbativa, cuadrática con la distancia entre las correspondientes masas vecinas. Ellos esperaban que, al introducir la menor perturbación al sistema descrito por el Hamiltoniano no-perturbado, con el tiempo la energía se distribuiría equitativamente entre todos los modos normales transformándose el movimiento en ergódico.

Para su sorpresa, lo que observaron fue que al partir de un estado en el que toda la energía estaba concentrada en el armónico fundamental, la energía comenzaba efectivamente a distribuirse entre los demás modos, pero esto ocurría hasta solamente el cuarto o quinto. Luego, con el tiempo, empezaba a concentrase nuevamente en el primer modo para luego recomenzar a

distribuirse nuevamente entre esos pocos primeros armónicos, para seguir con un comportamiento cuasi-periódico que “modulaba” al comportamiento periódico de los modos normales no perturbados.

Donde esperaban ver el desorden de la ergodicidad, ellos encontraron en realidad orden. Esto resultó inesperado para estos científicos a tal punto que Fermi llegó a hablar de “un pequeño descubrimiento”. Esto constituye un ejemplo, quizás el primero de una regla bastante aceptada en los estudios modernos de sistemas no lineales y complejos: que lo interesante generalmente está en encontrar no lo que uno esperaría sino lo inesperado [24].

Por lo anterior mencionado, ¿por qué hubo que esperar más de medio siglo para que los sistemas no lineales aparecieran?, la respuesta a esta pregunta es muy sencilla, como se mencionó anteriormente, las razones principales son:

- La falta de tecnología.
- Los programas para resolverlos.

Las aplicaciones de las ecuaciones no lineales son de interés en física, matemáticas, entre otras ciencias y la mayoría de los problemas físicos son implícitamente no lineales en su naturaleza. Ejemplos físicos de sistemas lineales son relativamente raros. Las ecuaciones no lineales son difíciles de resolver y dan origen a interesantes fenómenos como la teoría del caos citado en un principio.

Es por eso que en la actualidad con apoyo de las computadoras, y los programas se ha podido dar un mejor análisis a los sistemas no lineales.

Hoy en día se encuentran diferentes aplicaciones para la solución de sistemas no lineales por ejemplo:

- Dinamics Solver.
- Dstool.
- Matlab.
- XPPAUT.
- Simnon.
- Ant 4.669.
- Entre otros.

Estos sistemas son realizados en diferentes lenguajes como:

- C / C++.
- Fortran.
- Delphi.

Estos programas mencionados anteriormente sólo manejan herramientas que son construidas por el programador, y al ser un sistema cerrado éste no permite hacer modificaciones. Además, estos sistemas no son tan amigables para el usuario.

Existe poco desarrollo de programas para la solución de sistemas no lineales que sean amigables con el usuario, uno de los más nuevos fue realizado por el Dr. Juan M. Aguirregabiria, llamado Dinamics Solver, el cual tiene bastantes herramientas para el análisis de sistemas no lineales.

En el caso de México, no se trabajan sistemas relacionados con esta tesis, los investigadores utilizan los programas ya realizados y ellos mismos programan los algoritmos en Matlab.

En esta tesis se desarrollará un Toolbox para análisis de sistemas no lineales, soportado por una plataforma amigable con el lenguaje de programación Java, e interactúa con el sistema matemático más utilizado actualmente llamado Matlab.

1.2. Planteamiento del problema

Las ecuaciones diferenciales ordinarias no siempre tienen soluciones, y para su análisis es necesario el uso de las computadoras. Pero como se mencionó anteriormente los sistemas que existen no son amigables con el usuario, y están limitados a modificaciones, es por eso la realización de esta tesis.

El programa que se realizará le permitirá al usuario ingresar la(s) fórmula(s), elegir el algoritmo numérico (ODE) con el que se quiere trabajar e inicializar los parámetros, y el programa automáticamente le mostrará en pantalla el análisis

del algoritmo elegido. Además el código fuente será Opensource, para que pueda ser modificado para futuras actualizaciones y ampliaciones.

1.3. Preguntas de investigación

Las preguntas que surgen a partir de lo expuesto anteriormente son:

1. ¿Cómo resuelve el usuario no experto en programación el análisis de los algoritmos de sistemas no lineales?
2. ¿Cómo se puede ahorrar tiempo el investigador o estudiante, en buscar una herramienta de análisis de sistema no lineales que se adapte a sus necesidades?

1.4. Hipótesis

Para contestar las preguntas de investigación se propone la siguiente hipótesis:
El análisis de sistemas no lineales es posible solucionarlo al utilizar un sistema que interactúe con Matlab para mostrar la solución, sin la necesidad de programar una sola línea.

1.5. Objetivos del proyecto

Para llevar a cabo este proyecto se plantea primero un objetivo general y este en objetivos específicos para lograr una mejor secuencia en su desarrollo.

1.5.1. Objetivo general

Para validar la hipótesis o no, se plantea el siguiente objetivo general:
Realizar un sistema para análisis de sistemas no lineales que codifique las fórmulas, muestre parámetros para su inicialización y plasme en otra ventana la gráfica generada con Matlab.

1.5.2. Objetivos específicos

Para el desarrollo del objetivo general de esta tesis se realizaron los siguientes puntos:

- Analizar las ecuaciones diferenciales ordinarias.
- Diseñar la interfaz del software y el funcionamiento interno del mismo.
- Programar las funciones de Matlab para la generación de las clases.
- Realizar de pruebas.
- Validar el Software.

1.6. Justificación del estudio

El problema principal de los programas actuales, es su restricción a modificar o implementar algoritmos, y el inconveniente para trabajar en Matlab es el tener conocimiento en programación, es por esto que la realización de un programa que realice el análisis de los sistemas no lineales será de gran utilidad no sólo para los estudiantes, si no para los investigadores que no invertirán tiempo en programar o en buscar un sistema que se adapte a sus necesidades.

1.7. Limitaciones del estudio

Las siguientes delimitaciones son necesarias e importantes, para poder terminar a tiempo el proyecto, tanto en sus alcances, aplicaciones y características.

1.7.1. Delimitación geográfica

El lugar donde se utilizará será para uso principal de la UABC, Facultad de Ingeniería Ensenada.

1.7.2. Delimitación tecnológica

- Trabajará en plataforma Java.

- El programa estará limitado a las capacidades del Toolbox.
- Será de código abierto.
- Se va a trabajar por medio de variables de estado.
- El modelo a seguir será por medio de ecuaciones diferenciales ordinarias.

1.7.3. Delimitación de usuarios

Este programa está diseñado para estudiantes o investigadores, que trabajan con sistemas no lineales.

Algunos de los interesados en ésta herramienta son:

- Grupo de control no lineal de CICESE: Dr. Joaquín Álvarez.
- Grupo de Mecatrónica del CINVESTAV (México).
- Tecnológico de Monterrey campus Puebla: Dr. Hugo González.
- UABC campus Ensenada: Dr. Manuel Moisés Miranda Velazco.

CAPÍTULO II

MARCO TEÓRICO

2.1. Introducción

Debido a la complejidad y pérdida de tiempo para obtener el resultado de los análisis de sistemas no lineales manualmente, se empiezan a realizar programas con ayuda de la computadora que ejecutan estos análisis de una forma más rápida y concreta.

Actualmente existen programas tanto comerciales como escolares, que realizan estos análisis. Pero los programas existentes no son tan amigables ni para el estudiante que empieza con este tipo de análisis, ni para el investigador. Además, en los sistemas como Matlab el usuario necesita saber programar para obtener el resultado.

A continuación se mencionan los diferentes programas o herramientas que utilizan los investigadores y estudiantes, para obtener el resultado de estos sistemas no lineales.

2.2. Técnicas para la solución de sistemas no lineales

El autor del libro de J.C. Sprott menciona varias técnicas para la solución de sistemas no lineales enfocados a las series de tiempo no lineal al considerar sistemas físicos, financieros y psicológicos.

Maneja varios algoritmos para el análisis de series de tiempo no lineal. También comenta que la mejor técnica para entender las herramientas con las que trabaja, es implementar sus propios programas para cada algoritmo, por lo tanto el autor implementó algoritmos en Matlab en su página de Internet disponibles al público [6].

2.3. Conversación con el Dr. Michael Small

Se hizo contacto por correo electrónico con el Dr. Michael Small para preguntar cuales son las herramientas utiliza para generar soluciones de los sistemas no lineales: su respuesta fue que utiliza varios sistemas como Time Series Data y Matlab.

Comenta que actualmente no ha encontrado un sistema lo suficientemente completo para realizar sus análisis [4].

2.4. Publicación sobre análisis de oscilaciones

La investigación de esta publicación titulada: análisis de oscilaciones no lineales en sistemas complejos de mecánica de hidrodinámica [13], estudia la metodología de análisis y simulación interactiva de sistemas dinámicos oscilantes de mecánica e hidrodinámica.

En este, se utilizan diferentes herramientas que interconectadas entre ellas dan al usuario un análisis precisos de los experimentos realizados en el océano.

Ya tienen definidos los programas que necesitan para realizar el trabajo, son programas comerciales que son utilizados para diferentes análisis, los programas son:

- AQWA: Para dinámica de diferentes cuerpos.
- UM: análisis de bifurcaciones.
- XPPAUT (XPPAUT, 2004).
- También incluye otros programas como MAPLE, MATHEMATICA, MATLAB.

2.5. Controlweb

Es una herramienta para el análisis y simulación, de sistemas de control en internet, permite analizar y simular, sistemas lineales de control con una variable de entrada y una de salida por medio de la red. Ver figura 1.

El objetivo de este sitio es que los alumnos o investigadores que estudiar el tema de sistemas de control puedan tener acceso en cualquier lugar del mundo sólo con una conexión a la red.

Esta página está realizada mediante páginas Web y applets Java que la hacen accesible a través de Internet.

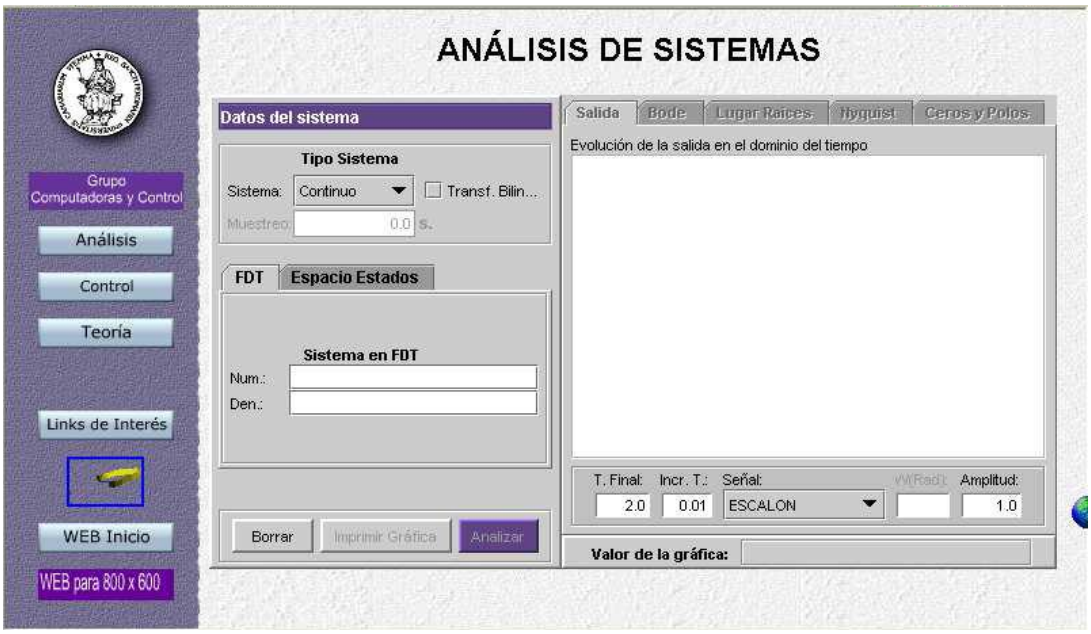


Figura 1. Pantalla principal de ControlWeb.

2.6. Conversación con el Dr. Hugo González

El autor de esta tesis se comunicó vía correo electrónico con el doctor [5], el cual comenta que efectivamente trabaja con sistemas no lineales y utiliza varios programas para el análisis como son:

- INSITE - Chua & Parker.
- NDT - Ditto y cols.
- CHAOSTIN - Hugo González & Clark Austin - desarrollado sobre Delphi.
- Simnon.
- Mathematica.
- Matlab (no en toolbox).

2.7. Aplicaciones que resuelven el análisis de sistemas no lineales

Una vez que se empezaron a realizar programas para diferentes áreas científicas, surgieron también herramientas para la solución de sistemas no lineales, con los cuales los usuarios no perdían tiempo para encontrar una solución aproximada.

Actualmente, algunos científicos utilizan estos programas ya que no quieren perder tiempo en encontrar otra herramienta que se adapte a sus necesidades.

A continuación se describen algunos de los programas que más se utilizan en la actualidad.

2.7.1. Dinamics Solver

Es una herramienta para estudiar las ecuaciones diferenciales, sistemas dinámicos no lineales, caos determinista, mecánica, etc. Además puede dibujar figuras matemáticas, como los fractales.

El sistema lo realizó el Dr. Juan M. Aguirregabiria, en su sitio web http://tp.lc.ehu.es/jma/ds/ds.html se puede descargar esta herramienta y su tutorial.

En la figura 2 se puede apreciar la pantalla principal de Dinamics Solver.

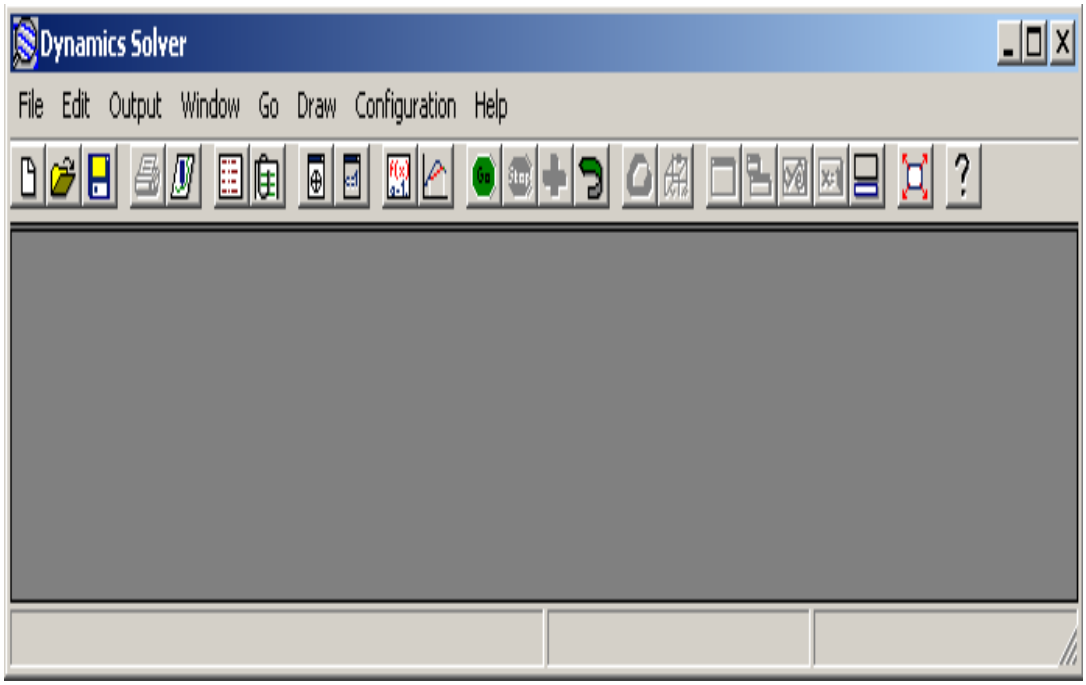


Figura 2. Pantalla principal de Dinamics Solver.

Este sistema maneja sus propias gráficas como se muestra en la figura 3: ejemplo que muestra la solución de las ecuaciones de Lorenz.

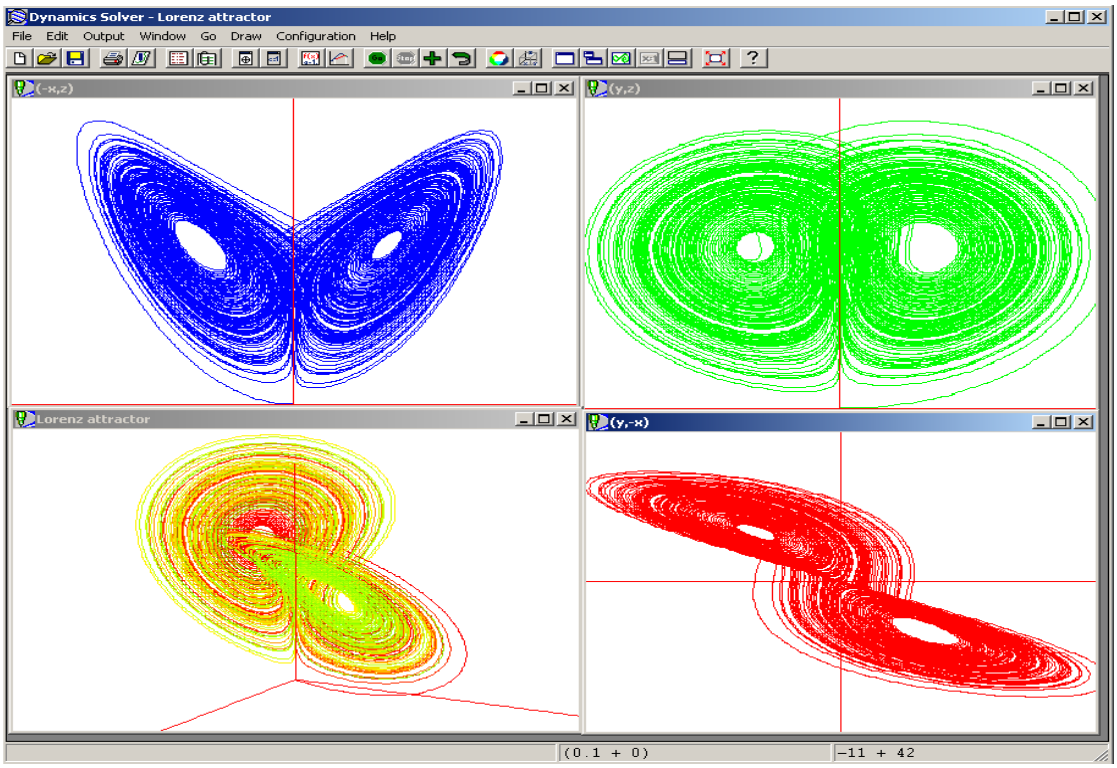


Figura 3. Ejemplo de ecuación de Lorenz.

La forma de trabajar Dinamics Solver es por medio de muchas ventanas, lo que es atractivo es que vienen instalados muchos ejemplos, sólo se necesita elegir con cuales ecuaciones o sistemas se quiere trabajar. Las figuras siguientes muestran algunas de las ventanas que maneja este programa. En el sub menú "Definitions" es donde contiene las ecuaciones, variables, parámetros y condiciones iniciales con las que trabaja el sistema.

La figura 4 muestra la ventana donde se inicializan las variables.

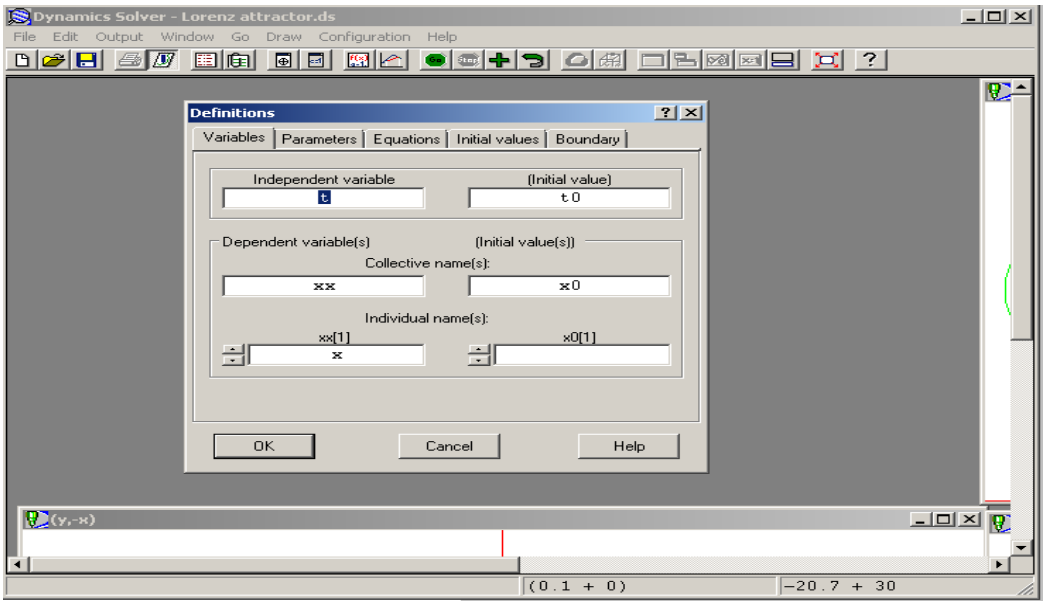


Figura 4. Ventana de Variables.

La figura 5 muestra los parámetros utilizados, aquí se inicializan o se modifican los valores con lo que se quiere trabajar.

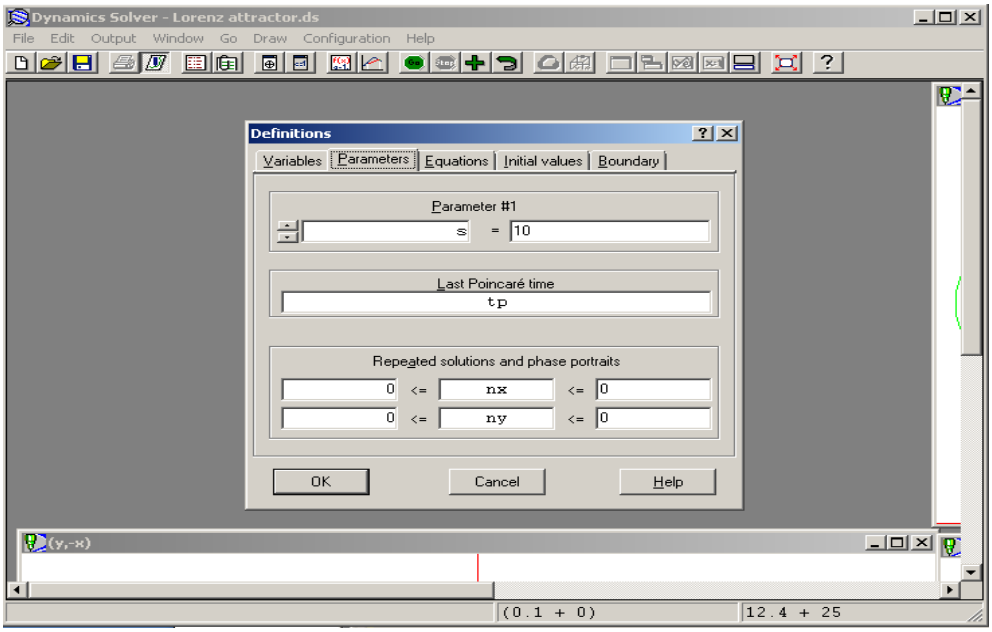


Figura 5. Ventana de parámetros.

Para agregar o modificar las ecuaciones la figura 6 muestra la ventana para hacerlo.

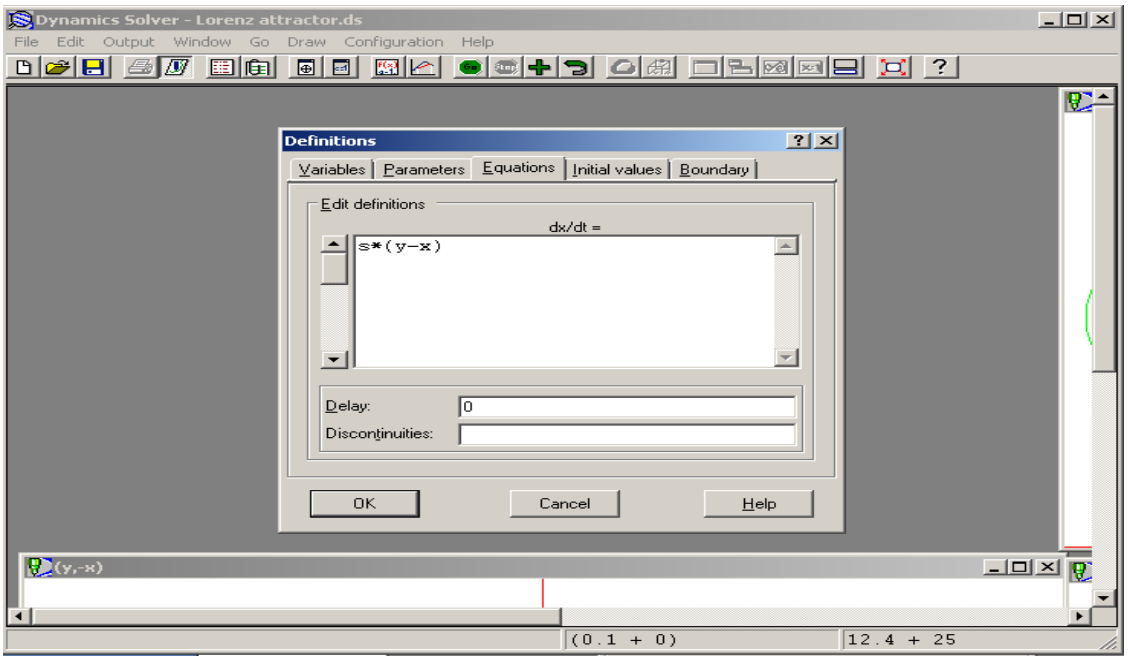


Figura 6. Ventana de ecuaciones.

Y para finalizar con el sub menú “Definitions”, la figura 7 muestra esa ventana.

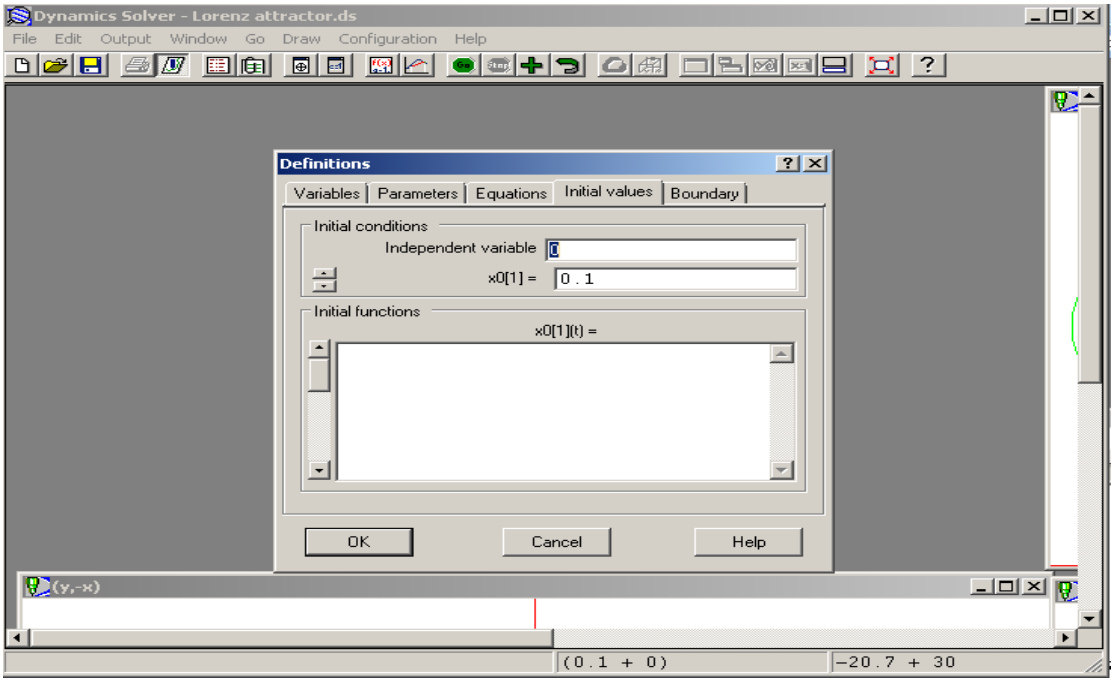


Figura 7. Ventana de condiciones iniciales.

Se puede observar y como se mencionó anteriormente, Dinamics Solver trabaja por medio de ventanas, las figuras anteriores sólo mostraron un sub menú del sistema, lo cual al trabajar con este programa puede llegar a ser un poco complejo o engorroso.

2.7.2. Simnon

SIMNON es una aplicación tan útil para la simulación de procesos químicos como centrales eléctricas. Esto puede ser usado para algoritmos de control complejos, para modelos financieros, la dinámica de robot. Cada sistema, que

puede ser definido en términos (condiciones) matemáticos, también puede ser simulado en Simnon.

Los miles de personas en todo el mundo usan SIMNON como un instrumento eficiente para simular procesos y productos. Las universidades y centros de investigación, en más de 40 países han encontrado a SIMNON como un gran programa para la simulación interactiva.

Los inconvenientes de utilizar este sistema para la solución de sistemas no lineales se describen a continuación:

- Utiliza su propia sintaxis y estructura: es como aprender otro lenguaje de programación, en la figura 8 muestra la pantalla principal, donde se puede observar la ventana para editar las instrucciones.
- Actualización: se dejo de actualizar en el año de 2003.
- Archivos: al igual que Matlab, se tienen que teclear 2 archivos para la solución de un sistema no lineal.
- No es amigable al usuario.

Figura 8. Pantalla Principal con un ejemplo en Simnon.

A continuación se muestran los dos archivos que se necesitan para obtener la gráfica.

Lorenz.t es un archivo con las variables que se van a visualizar en pantalla y su color.

MACRO LORENZ
" Version: 1.0
" Abstract:
" Description: Macro for the Lorenz demo
" Revision: 1.0
" Author: SSPA Systems, Gothenburg, Sweden
" Created: 1993-05-28
" Enter commands here:
syst Lorenzeq
store x y z
error 1e-6 " Phase plane plot requires better accuracy
simu 0 20
" msgbox 'Simulation ready! Let us plot...'
newplot
SWITCH color blue
ashow y(x)

```
text 'Lorenz equation: y(x)'
newplot
SWITCH color red
ashow z(x)
text 'Lorenz equation: z(x)'
newplot
SWITCH color black
ashow z(y)
text 'Lorenz equation: z(y)'
newplot
split 2 2
area 1 1
switch color blue
ashow y(x)
text 'Lorenz equation: y(x)'
switch color red
ashow z(x)
text 'z(x)'
switch color black
ashow z(y)
text 'z(y)'
" msgbox 'Plotting ready! Look at the Plot Windows, and make some zooming...'
switch color reset

END
```

A continuación se presenta el segundo código llamado Lorenzeq.t, que contiene las ecuaciones, los parámetros y condiciones inicializadas.

```
CONTINUOUS SYSTEM LORENZEQ
" Version:      1.0
" Abstract:
" Description:  Lorenz demo
" Revision:     1.0
" Author:       SSPA Systems, Gothenburg, Sweden
" Created:      1993-05-28

" Inputs and outputs:

" States, derivates and time:
STATE x y z
DER dx dy dz
```

" Initializations:

" Equations:
dx=a(y-x)*
*dy=b*x-y-x*z*
*dz=x*y-c*z*

" Parameter values:
a: 10
b: 28
c: 2.6666667

x: -8
y: -8
z: 24

END

Como se puede observar, si lo que se requiere es obtener el resultado de una forma sencilla, Simnon no es una buena opción, ya que se requiere aprender cómo funciona y la sintaxis de sus comandos.

2.7.3. XPPAUT

Es un programa científico para la solución de sistemas no lineales, ecuaciones diferenciales, especial para la modelación de diversos problemas matemáticos.
La figura 9 muestra un ejemplo de cómo se comporta un archivo que contiene de ejemplo llamado pendulum.ode.

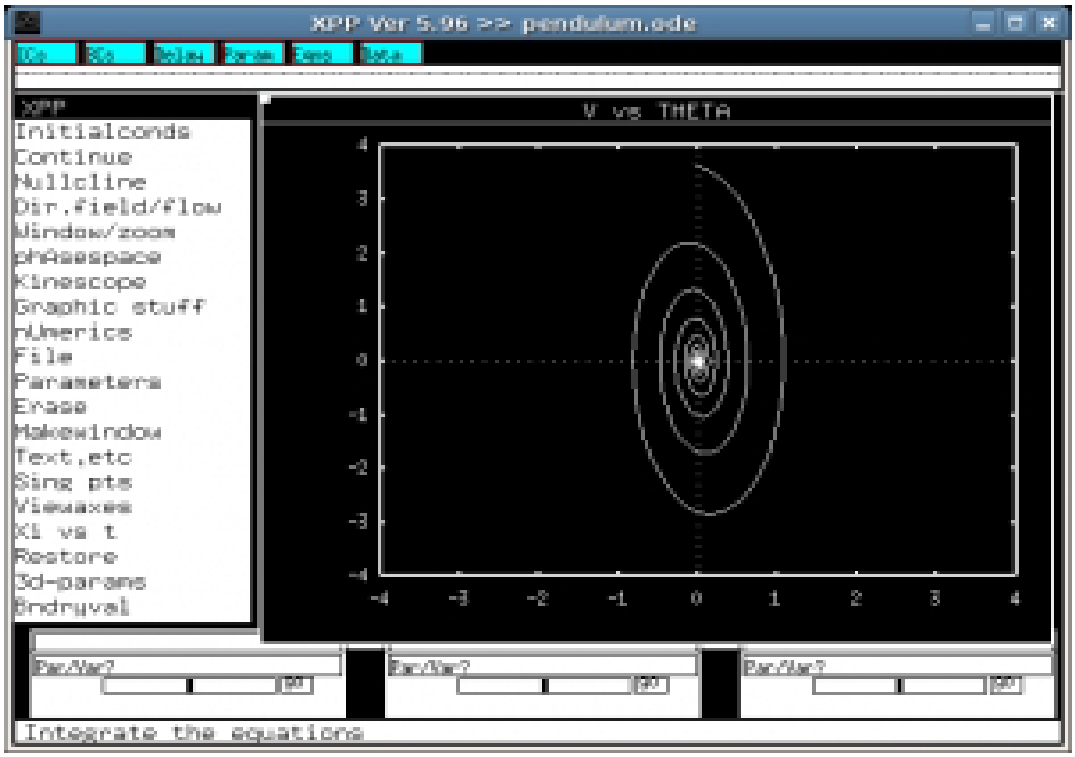


Figura 9. Pantalla con ejemplo en XPPAUT.

El siguiente código, es con la nomenclatura de XPPAUT:

```
# the famous Lorenz equation set up for 3d view
init x=-8  y=-8  z=24
par r=27  s=10  b=2.66666
x'=s*(-x+y)
y'=r*x-y-x*z
z'=-b*z+x*y
@ dt=.025, total=40, xplot=x,yplot=y,zplot=z,axes=3d
@ xmin=-20,xmax=20,ymin=-30,ymax=30,zmin=0,zmax=50
@ xlo=-1.5,ylo=-2,xhi=1.5,yhi=2
@ maxstor=20000
@ phi=60
Done
```

Se puede ver que a diferencia de los sistemas anteriores que se necesitan dos archivos, este sistema utiliza un solo archivo donde se declaran las ecuaciones, inicializan variables y se manda a imprimir en pantalla. En la figura 10, se muestra el resultado del código anterior.

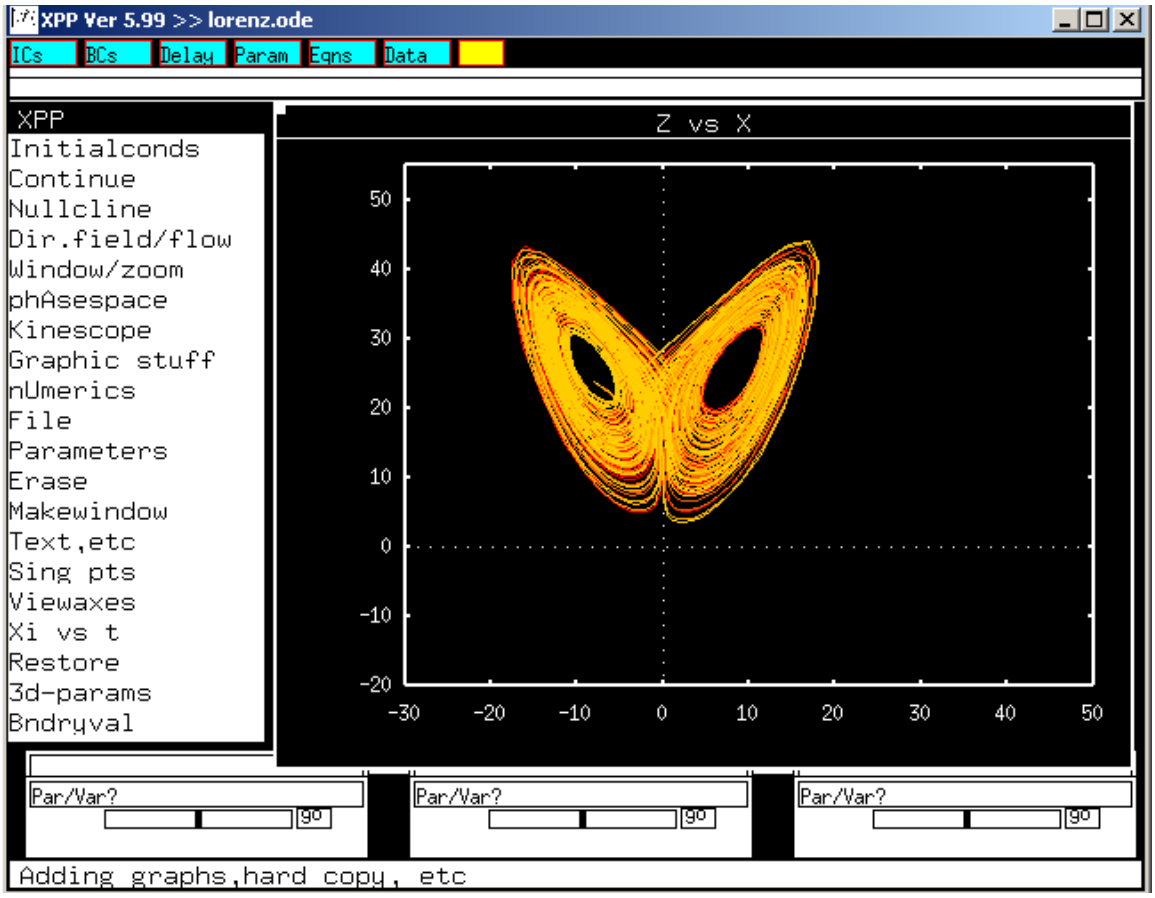


Figura 10. Resultado del código de Lorenz.ode.

Como resumen de este programa, XPPAUT muestra un código sencillo, pero la interfaz gráfica es confusa ya que trabaja en MS-Dos.

2.7.4. Matlab

Es un programa matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Además de ser multiplataforma.

Entre sus características básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la generación de interfaces de usuario (GUI), la comunicación con programas en otros lenguajes y con otros dispositivos hardware. El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes); y las de Simulink con los paquetes de bloques. La figura 11 muestra la pantalla principal de Matlab, donde se puede apreciar el command Window, donde se le dan las instrucciones a Matlab.

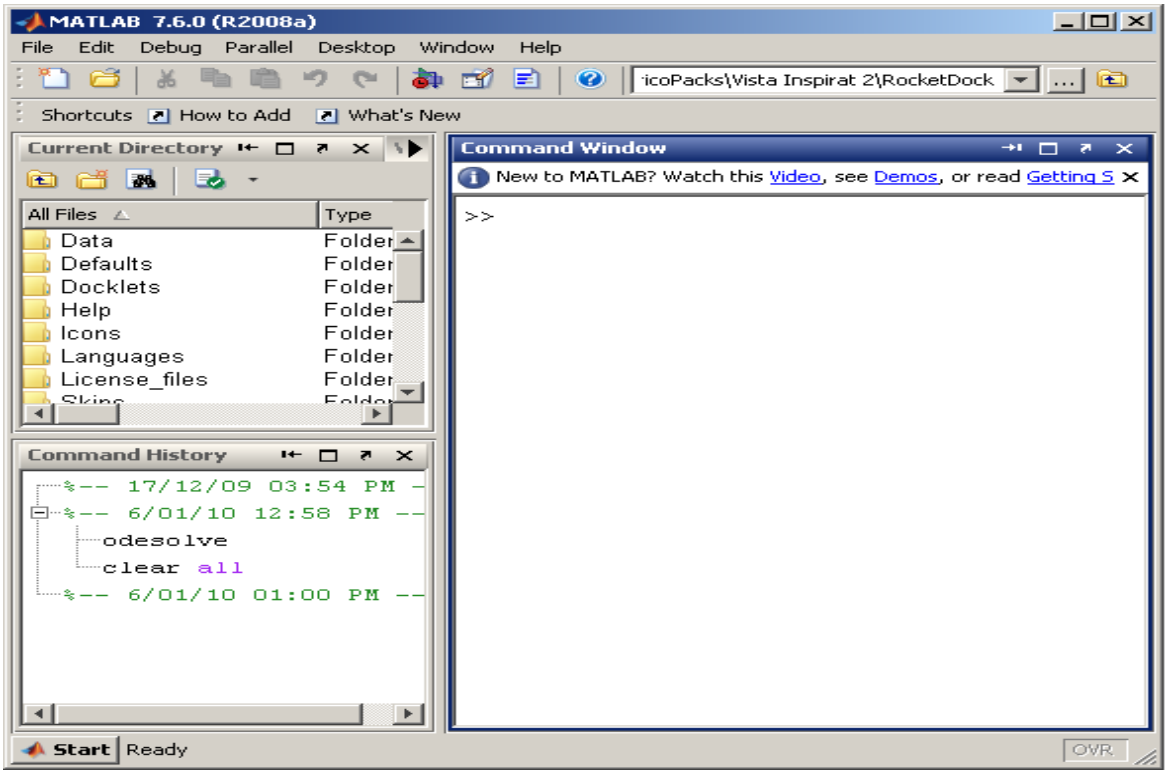


Figura 11. Pantalla principal de Matlab.

Al utilizar este programa, el usuario debe tener conocimiento de la programación del lenguaje de Matlab, conocer su sintaxis, su estructura, palabras reservadas, etcétera. Además se tienen que hacer más de un archivo .m (extensión de Matlab) para obtener la gráfica.

Por mencionar un ejemplo la Figura 12 es un archivo llamado Lorenz.m en el cual se ingresan las ecuaciones que se utilizarán y las variables la gráfica necesita.

Lorenz.m

```
function  xp=lorenz(t,x,a,b,c)
%definicion de ecuaciones
auxiliares (opcional)
u=x(1)^2;
%definicion de ecuaciones
xp(1,1)=a*(x(2)-x(1))-u+u;
xp(2,1)=b*x(1)-x(2)-x(1)*x(3);
xp(3,1)=x(1)*x(2)-c*x(3);
```

Figura 12. Archivo llamado Lorenz.m.

A continuación se muestra otro archivo en el cual se inicializan variables y se da la instrucción de graficar (figura 13).

Lorenzr.m

```
%valor de los parametros
a=10;
b=28;
c=2.6666667;
%valor de x0 y t0
x0=[-8;-8;24];
t0=[0 20];
options=odeset('RelTol',1e-6);
[t,x]=ode113(@lorenz,t0,x0,options,a,b,c);
%u=x(:,1).^2;
figure
plot(x(:,1),x(:,2))
figure
plot(x(:,2),x(:,3))
figure
plot(x(:,1),x(:,3))
```

Figura 13. Archivo llamado Lorenzr.m.

La figura 14, presenta la gráfica generada por los archivos anteriormente mencionados.

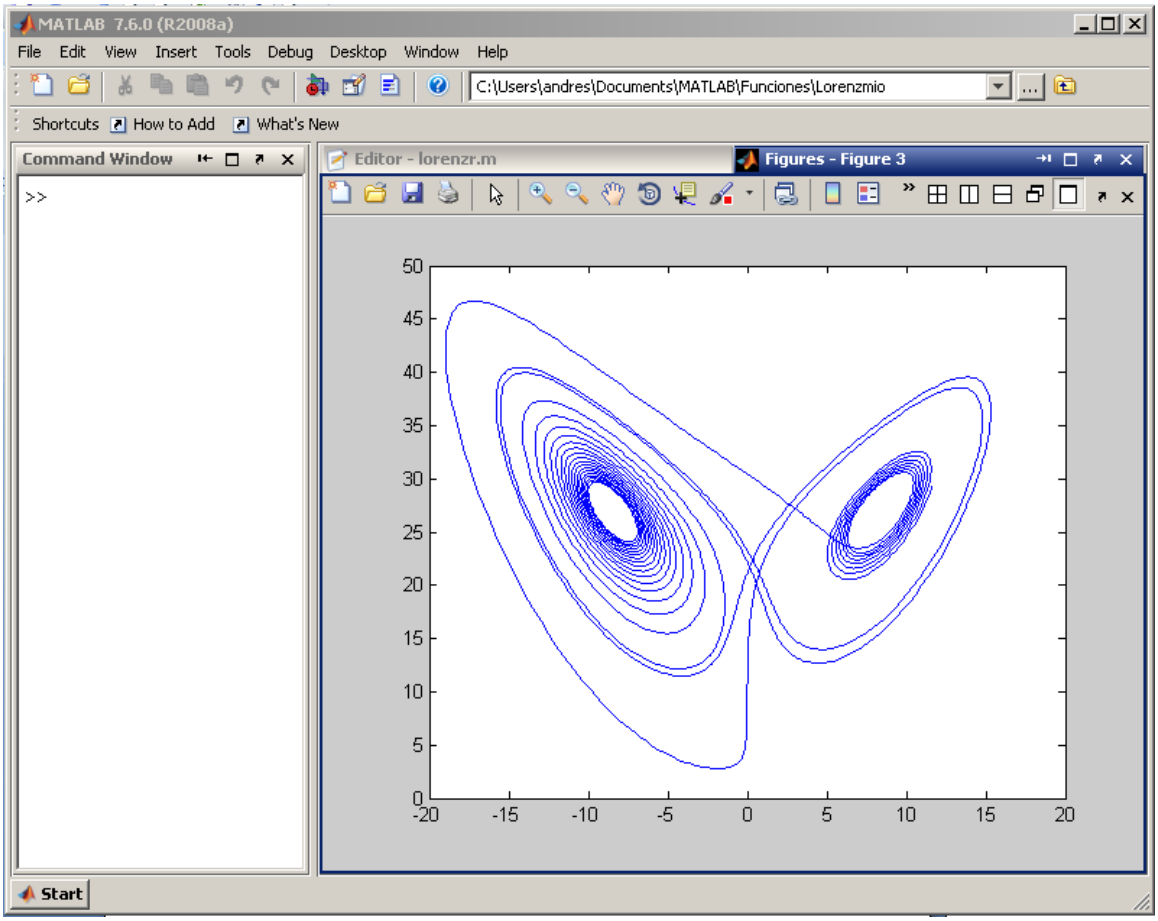


Figura 14. Resultado del análisis.

Por lo tanto, al resolver una ecuación, el usuario puede tener errores o con el sólo hecho de olvidar un parámetro la solución será errónea.

La tabla 1, muestra un resumen de las características de los programas analizados anteriormente.

Tabla 1. Características de los programas analizados.

Nombre del programa:	**¿Se necesita programar?:**	**Ultima Actualización:**	**¿Es código libre?:**	**¿Es Entendible?:**
Dinamics Solver.	No, se trabaja por medio de ventanas.	2008.	Si, se puede descargar de la web.	Sí, pero es engorroso, ya que hay muchas ventanas.
Simnom.	Si, se necesitan dos códigos.	2003.	No.	Sí, pero es necesario estudiar su propio lenguaje.
XPPAUT.	Si, solo un código.	Febrero de 2010.	No, se puede descargar el programa de la web, pero no se puede descargar el código.	No, además de que tiene su propio lenguaje, no es entendible.
Matlab.	Si, se necesitan dos códigos.	Septiembre de 2010.	No, es necesario tener licencia.	Sí, pero es necesario conocer su lenguaje.

2.8. JAVA

Es una plataforma virtual de software desarrollada por Sun Microsystems, de tal manera que los programas creados en ella puedan ejecutarse sin cambios en diferentes tipos de arquitecturas y dispositivos computacionales ("Diferentes plataformas").

La plataforma Java consta de las siguientes partes:

- El lenguaje de programación, mismo.
- La máquina virtual de Java o JRE, que permite la portabilidad en ejecución.
- El API Java, una biblioteca estándar para el lenguaje.

Originalmente llamado OAK por los ingenieros de Sun Microsystems, Java fue diseñado para correr en computadoras incrustadas. Sin embargo, en 1995, dada la atención que se producía en la Web, Sun Microsystems la distribuyó para sistemas operativos tales como Microsoft Windows.

El lenguaje mismo se inspira en la sintaxis de C++, pero su funcionamiento es más similar al de Smalltalk que a éste. Incorpora sincronización y manejo de tareas en el lenguaje mismo (similar a Ada) e incorpora interfaces como un mecanismo alternativo a la herencia múltiple de C++.

A fines del siglo XX, Java llegó a ser el lenguaje de mayor acogida para programas de servidor, que utiliza una tecnología llamada JSP (Java Server Page), similar a otras tecnologías del lado del servidor como ASP (Active Server Page) de Microsoft o PHP. Sumado a JSP la tecnología de JavaBeans, permitía adaptar al mundo web el patrón MVC (modelo-vista-controlador) que ya se había aplicado con éxito a interfaces gráficas.

Java llegó a ser extremadamente popular cuando Sun Microsystems introdujo la especificación J2EE (Java 2 Enterprise Edition). Este modelo permite, entre otras cosas, lograr una separación entre la presentación de los datos al usuario (JSP o Applets), el modelo de datos (EJB), y el control (Servlets). Enterprise Java Beans (EJB) es una tecnología de objetos distribuidos que pudo lograr el sueño de muchas empresas como Microsoft e IBM de generar una plataforma de objetos distribuidos con un monitor de transacciones. Con este nuevo estándar, empresas como BEA, IBM, Sun Microsystems, Oracle y otros crearon nuevos "servidores de aplicaciones" que tuvieron gran acogida en el mercado.

Además de programas del servidor, Java permite escribir programas de interfaz gráfica o textual. También se pueden correr programas de manera incorporada o incrustada en los navegadores web de Internet en forma de Java applets, aunque no llegó a popularizarse como se esperaba en un principio.

Los programas en Java generalmente son compilados a un lenguaje intermedio llamado bytecode, que luego son interpretados por una máquina virtual (JVM). Esta última sirve como una plataforma de abstracción entre la máquina y el lenguaje, para permitir que se pueda "escribir el programa una vez, y correrlo en cualquier lado". También existen compiladores nativos de Java, tanto software libre como no libre. El compilador GCC de GNU compila Java a código de máquina con algunas limitaciones al año 2002.

Con la evolución de las diferentes versiones, no sólo se han producido cambios en el lenguaje, sino que se han producido cambios mucho más importantes en sus bibliotecas asociadas, que han pasado de unos pocos cientos en Java 1.0, a más de tres mil en Java 5.0. En particular, se han añadido APIs completamente nuevas, tales como Swing, que es utilizado en este trabajo de tesis y Java2D.

2.9. Sumario

Se observa que existen diferentes programas que analizan los sistemas no lineales, cada uno de ellos con interfaces y modo de operación diferente, en los cuales la mayoría de éstos, la forma de operar es la programación, característica principal que el Toolbox para el análisis de sistemas no lineales suprime, para facilitarle el trabajo al usuario.

CAPÍTULO III

METODOLOGÍA DE LA INVESTIGACIÓN

3.1. Introducción

Conforme pasa el tiempo, surgen computadoras mejor equipadas, con más memoria, aumento de espacio en disco duro, mayor velocidad y sistemas más eficientes para cualquier campo de la ciencia. Es el caso de un sistema matemático de carácter científico mencionado anteriormente llamado Matlab que, debido a su velocidad, su precisión de procesado de ecuaciones matemáticas y las gráficas que realiza se decidió trabajar con este lenguaje e interactuar con el lenguaje de alto nivel Java para demostrar la hipótesis de esta tesis.

Para realizar la interacción de estos lenguajes mencionados anteriormente, se requiere de hacer pequeños módulos, realimentándolos continuamente y hacer un desarrollo de calidad.

La figura 15 muestra un diagrama de flujo, del procedimiento general utilizado para obtener el resultado.

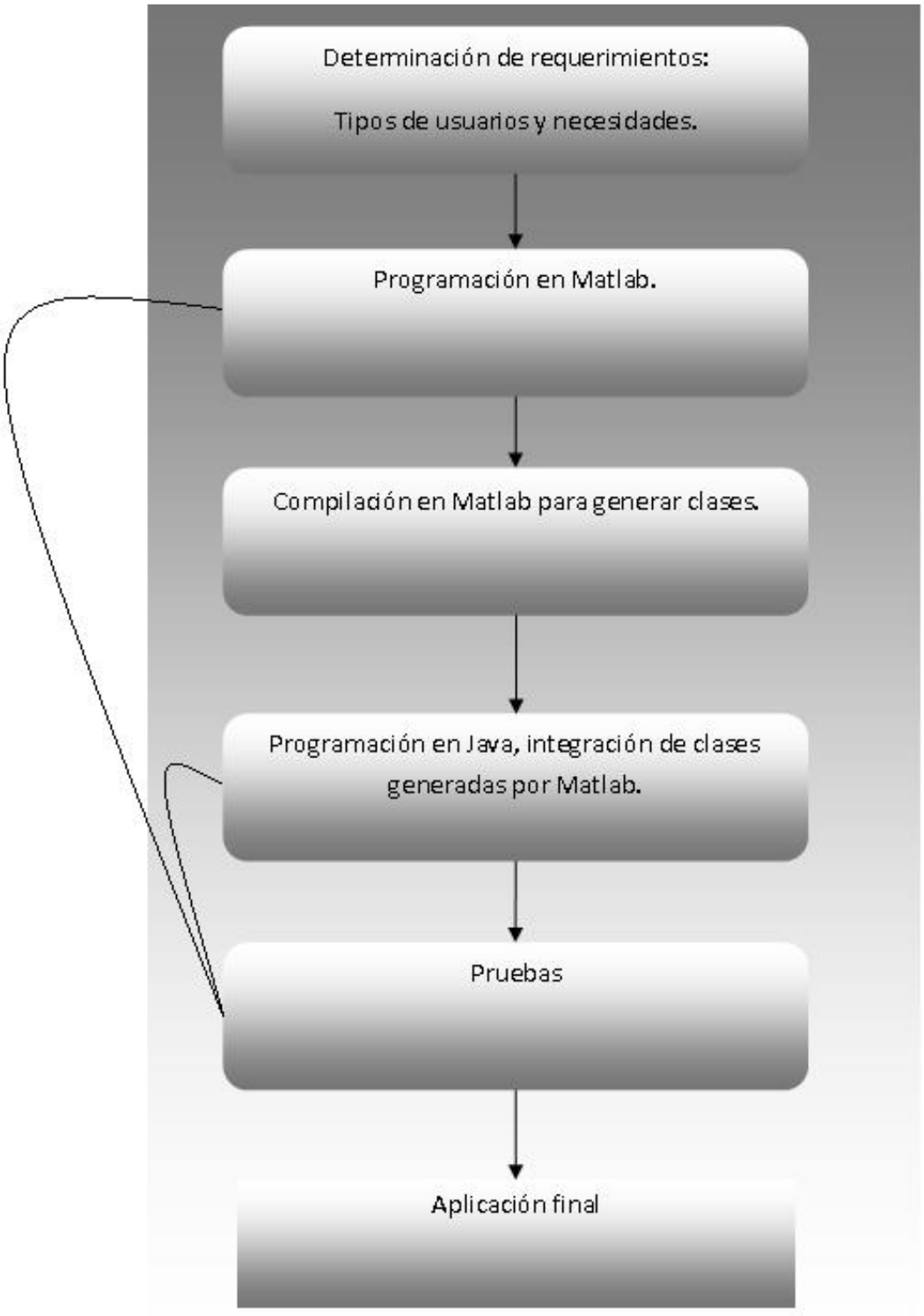


Figura 15. Procedimiento general utilizado para obtener la aplicación.

3.2. Ingeniería de software

La Ingeniería de Software es la rama de la ingeniería que aplica los principios de la ciencia de la computación y las matemáticas, para lograr soluciones costo-efectivas (eficaces en costo o económicas) a los problemas de desarrollo de software", es decir, "permite elaborar consistentemente productos correctos, utilizables y costo-efectivos" [2].

El proceso de desarrollo de software "es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo. Concretamente, define el qué, cuándo hacerlo y cómo alcanzar un cierto objetivo" [7].

Una de ellas llamada “Programación estructurada”, sugerida por Dijkstra, que con el apoyo de Bohm y Jacopini, de que todos los programas se puedan expresar como una combinación de secuencias de instrucciones, ramificaciones e iteraciones. La instrucción GOTO, un salto sin regreso de una parte a otra del programa, no es necesaria y la eliminación de este conduce a la programación estructurada. Sin embargo, se han encontrado nuevas perspectivas y paradigmas útiles que se fundamentan en este tipo de programación, ya que uno de los problemas que se encontró es que no está diseñada para hacer

corresponder las entidades del programa con las entidades del mundo real, lo que dificulta manejarla y adaptarla cuando cambia los requerimientos.

Actualmente existe la orientada a objetos (OO), el uso de objetos con datos y funcionalidad, ha demostrado ser una forma de pensamiento muy efectiva. La OO es un medio de expresión e implementación, en lugar de arquitectura o diseño. La OO es efectiva porque los objetos pueden representar las partes del mundo real de las aplicaciones y poder combinarlos con los componentes del Software.

3.3. Equipo y software

En esta sección se describe el equipo el software que es requerido para la realización del proyecto.

Computadora:

- **Requerimientos mínimos:**
  - Procesador de 32 o 64 bits.
  - Memoria RAM de 512 MB.
  - Memoria en disco duro mínimo de 500 MB.

Software mínimo requerido:

- Máquina virtual Java (JVM).
- MCR (Matlab Component Runtime).
- Windows instaler 3.1.

3.4. Sistema operativo

El sistema operativo en el que este proyecto es compatible es Windows XP, Vista y 7 (seven), con la finalidad de poder migrarlo a cualquier otro sistema operativo.

3.5. Lenguajes para la realización de esta tesis

Para la realización de este proyecto fue necesario de dos sistemas principales:

- Java.
- Matlab.

A continuación se menciona porque se utilizaron estos lenguajes.

3.5.1. Java

Se eligió este lenguaje por su facilidad de trabajar con Matlab para exportar las clases. Además es un lenguaje multiplataforma. Lenguaje que proviene de C++, sin embargo, Java pretende mejorar a C++ en muchos aspectos como el orientado a objetos.

Para realizar esta tesis se utilizó la plataforma de desarrollo Netbeans, el cual permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con las API's de NetBeans y un archivo especial que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

3.5.2. Matlab

Es un entorno de computación y desarrollo de aplicaciones integrado, orientado para llevar a cabo proyectos en donde se encuentran implicados elevados cálculos matemáticos y su visualización gráfica.

Matlab cuenta con su propio lenguaje de programación, muy parecido al que maneja C++. Y sus herramientas para poder trabajar con otros lenguajes como lo es Java hace muy práctico a Matlab para realizar este proyecto.

3.6. Usuarios

El programa puede ser usado por alumnos, profesores o personas, que están iniciándose en los sistemas no lineales, pero los principales usuarios son hallados en centros de investigación científica, en este caso investigadores o profesores de la Facultad de Ingeniería Ensenada de la Universidad Autónoma de Baja California.

3.7. Desarrollo del toolbox

En los siguientes subtítulos se presenta la manera en que se implementó el Sistema, con los cuales se desarrollan los puntos de los objetivos específicos para después empezar a realizar las pruebas necesarias.

3.7.1. Metodología

Una metodología de desarrollo de Software es un marco de trabajo para estructurar, planificar y controlar, el proceso de desarrollo en sistemas de información.

Con el paso del tiempo han existido varias metodologías para el desarrollo de Software. A continuación se muestran algunas:

- Programación estructurada.
- Programación estructurada Jackson.
- Programación Orientada a Objetos.
- Scrum.
- Programación extrema.
- Proceso Racional Unificado (RUP).
- Proceso Unificado Ágil. (AUP).
- Entre otras.

La metodología utilizada para la realización de esta tesis se llama Programación Extrema (XP).

XP forma parte del conjunto de métodos ágiles que centran su atención en ser un modelo de desarrollo común, sencillo y adaptable, a las características cambiantes y exigentes del cliente.

Las actividades principales de XP son las siguientes:

- Análisis de requerimientos.
- Diseño.
- Implementación.
- Pruebas.
- Despliegue.
- Gestión de cambios.

- Proyecto.

3.7.2. Análisis de requerimientos

Como lo maneja la metodología XP los requerimientos son la primera actividad a realizar.

Para construir algo primero debe entenderse que lo que debe ser ese “algo” y el proceso de entender y documentar este algo se llama “análisis de requerimientos”. En general, los requerimientos expresan “que” se supone que debe hacer esa aplicación, no intentan expresar “como” lograr estas funciones.

Los requerimientos para esta tesis son:

1. El Toolbox tendrá un lugar para poner las ecuaciones diferenciales y las ecuaciones auxiliares.
2. El límite de condiciones iniciales y de parámetros son 10.
3. El Toolbox verificará si hay errores de sintaxis en las ecuaciones.
4. Si hay errores de sintaxis no permitirá hacer algo hasta que se corrija el error o cancelar el trabajo que se va a realizar.
5. Si no hay errores el Toolbox automáticamente generará las condiciones iniciales sólo para inicializarse.
6. Tendrá otro apartado para que los parámetros se inicialicen.

7. Tanto los parámetros como las condiciones iniciales sólo podrán aceptar números positivos y negativos.
8. Se permitirá ingresar la variable independiente y su valor.
9. Se ingresará el valor inicial y valor final del intervalo de solución deseado.
10. Se podrá elegir el ODE con el que se quiera trabajar, entre los que se pueden ser: ode45, ode23, ode113, ode23s, ode23t.
11. Para seleccionar la salida tendrá dos opciones, en 2D o 3D.
12. Se mostrará una ventana que de opciones para elegir las variables con los que se quiere trabajar ya sea en 2D o 3D.
13. Habrá una opción para modificar el grado de tolerancia.
14. Se podrán guardar tanto el ODE File como los valores de parámetros y condiciones iniciales.

3.7.3. Diseño

El diseño de software es un proceso y un modelado a la vez. Es un conjunto de pasos repetitivos que permiten al diseñador describir todos los aspectos del software a construir.

El análisis de requerimientos proporciona la información necesaria para la creación y desarrollo de la fase del diseño. Una vez detallado el diseño es fácil la implementación del sistema.

La figura 14 muestra un diagrama, en el cual se puede visualizar el flujo de datos a través del Toolbox. Este se diseñó para conocer la secuencia de las operaciones con las que se trabajan en la programación del sistema.

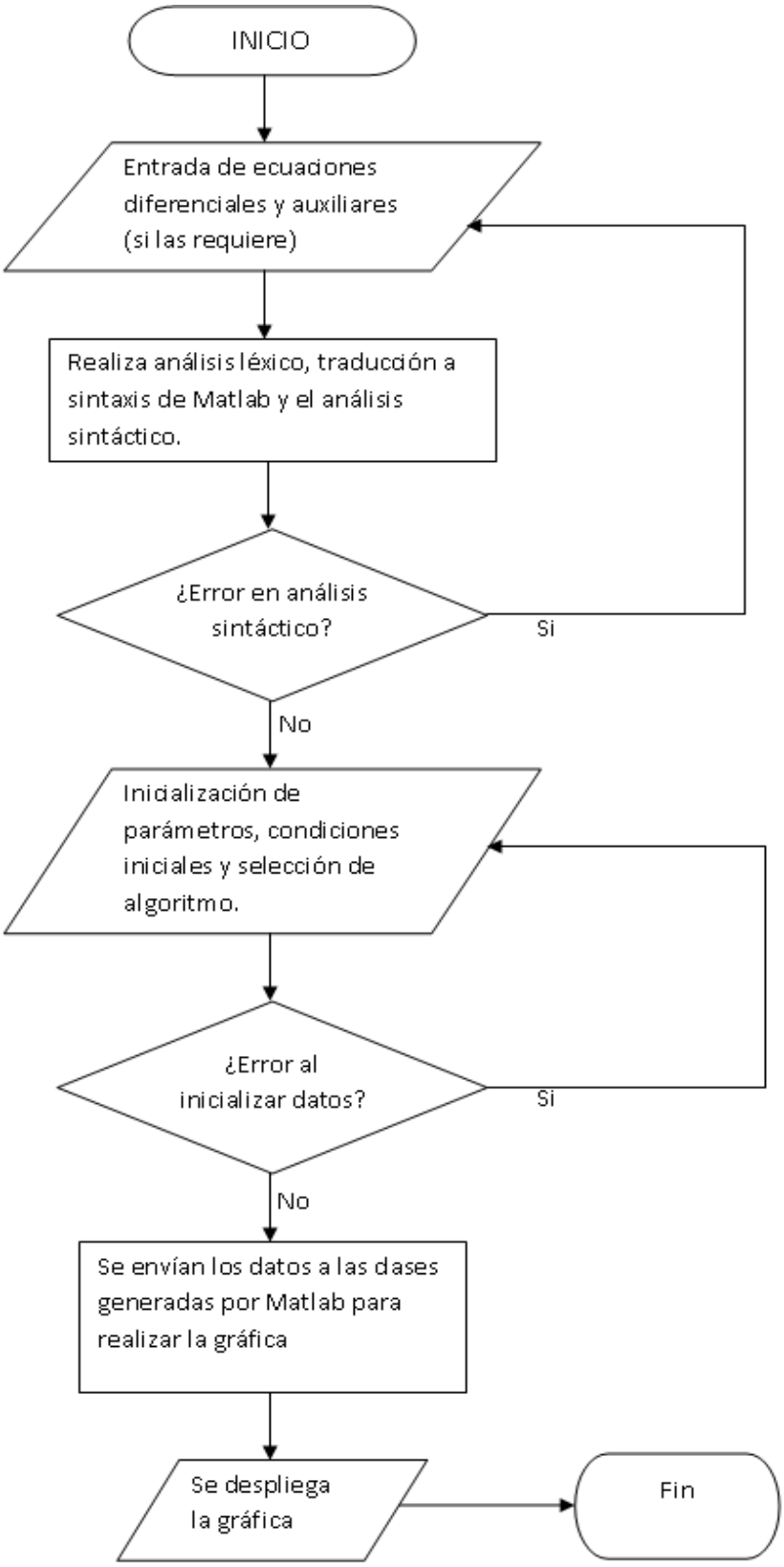


Figura 14. Diagrama de flujo del sistema.

3.7.4. Implementación

La implementación tiene como propósito la realización de un producto que satisfaga los requerimientos planteados durante el análisis, apoyándose en el modelo creado durante la etapa del diseño.

Para la implementación de esta tesis, la figura 15 muestra un diagrama a bloques del sistema realizado y después se explica cada una de las etapas.

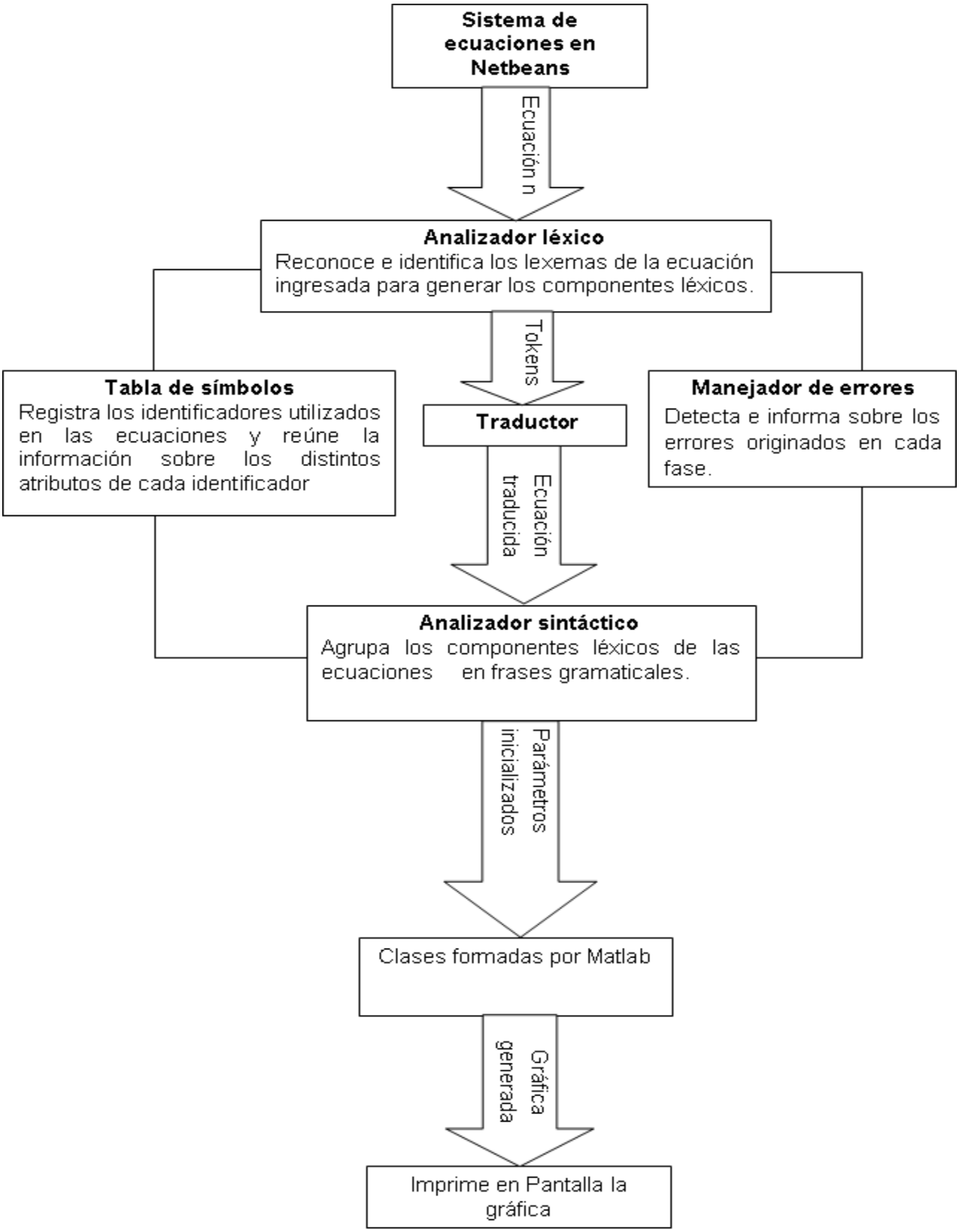


Figura 15. Diagrama a bloques del sistema.

A continuación se hace una descripción de cada uno de los componentes del diagrama a bloques antes presentado.

3.7.4.1. Sistema de ecuaciones en Netbeans

En el sistema se encuentra un apartado donde el usuario ingresa las ecuaciones con las que se va a trabajar, tanto las ecuaciones diferenciales como las auxiliares. Ver figura 16.

Figura 16. Apartado para ingresar ecuaciones.

A partir de que el usuario ingresa las ecuaciones, al presionar el botón de Analizar, el sistema envía las ecuaciones al analizador léxico.

3.7.4.2. Analizador léxico

El analizador léxico es la primera parte de un compilador. Su función es leer los caracteres de entrada y ubicarlas dentro de categorías, conocidas como

unidades léxicas, las cuales son empleadas por el analizador sintáctico para determinar si la ecuación que se capturó esta correcta o incorrectamente escrita gramaticalmente.

Funciones del analizador léxico

La principal función de esta primera etapa consiste en leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis. Esta interacción suele aplicarse al convertir al analizador léxico en una subrutina o corrutina del analizador sintáctico. Recibida la orden [Ingresa el siguiente componente léxico del analizador sintáctico], el léxico lee los caracteres de entrada hasta que pueda identificar el siguiente componente léxico. El cual devuelve al sintáctico según el formato convenido.

Además de esta función principal, el analizador léxico también realiza otras de gran importancia:

- Eliminar los comentarios del programa y espacios en blanco, tabuladores, retorno de carro, etc., en general, todo aquello que carezca de significado según la sintaxis del lenguaje.
- Reconocer los identificadores de usuario, números, palabras reservadas del lenguaje, etc., y tratarlos correctamente con respecto a la tabla de

símbolos (sólo en los casos en que este analizador deba tratar con dicha estructura).

- Llevar la cuenta del número de línea por la que va, por si se produce algún error, dar información acerca de dónde se ha producido.
- Avisar de errores léxicos. Por ejemplo, si el carácter '@' no pertenece al lenguaje, se debe emitir un error.

Token, patrón y lexema

Desde un punto de vista muy general, se puede abstraer el programa que implementa un análisis lexicográfico mediante una estructura como:

ecuación 1;

ecuacion2;

ecuacion3;

.

.

.

ecuación N.

Donde cada acción a ejecutar es un fragmento de programa que describe cuál ha de ser la acción del analizador léxico una vez que la secuencia de entrada coincida con la expresión regular. Normalmente esta acción suele finalizar con la devolución de una categoría léxica.

Para esto se debe conocer los siguientes conceptos en el análisis léxico:

- **Patrón**: es una expresión regular. La cual a su vez es una regla que genera la secuencia de caracteres que puede representar a un determinado componente léxico.
- **Token:** es la categoría léxica asociada a un patrón. Cada token se convierte en un número o código identificador único. En algunos casos, cada número tiene asociada información adicional necesaria para las fases posteriores de la etapa de análisis. El concepto de token coincide directamente con el concepto de terminal desde el punto de vista de la gramática utilizada por el analizador sintáctico.
- **Lexema:** es cada secuencia de caracteres concreta que encaja con un patrón.

Tokens a reconocer por el analizador léxico

El diseño de un analizador léxico inicia cuando define los tokens a reconocer para entonces seguir con los siguientes pasos [1]:

- Escribir la definición regular que denote al lenguaje generado por cada token a reconocer.
- Aplicar las reglas de Thompson a cada definición regular, con el fin de generar al Autómata Finito No Determinista (AFND) que reconozca las

cadenas del lenguaje denotado por la expresión regular para cada token a reconocer.

- El AFND producido por las reglas de Thompson sirve como entrada para el algoritmo de construcción de subgrupos, el cual genera el Autómata Finito Determinista (AFD) equivalente al AFND de Thompson.
- Luego se aplica el algoritmo de particiones al AFD generado por el algoritmo de construcción de subgrupos, para obtener un AFD reducido u óptimo.

El analizador léxico que se codificará consiste en el reconocimiento de 7 tokens:

- **Delim:** consiste de los delimitadores encontrados al codificar en cualquier lenguaje, también son conocidos como los caracteres blancos. Se reconocerán el caracter espacio, nueva línea: \n, el retorno de carro: \r, el tabulador: \t.
- **Id:** denota a todos los identificadores que empiezan con letra o guión bajo, seguidos de cualquier cantidad de letras, dígitos y guiones bajos.
- **opAsig:** representa a los operadores de asignación =, +=, -=, *=, /=.
- **opArit:** denota a los operadores aritméticos +, -, *, /, ^.
- **Num:** contiene al lenguaje de los números enteros y números reales con al menos una cifra en ambos partes.
- **Sep:** los separadores en este caso sólo son los caracteres ( ) paréntesis circulares.

- **termInstr:** se refiere al caracter ; (punto y coma) usado como terminador de instrucción en varios lenguajes de programación y seleccionado en este trabajo.

Para la realización del análisis léxico se realizaron dos clases, Léxico y Autómata. Estas se pueden encontrar en el apéndice B.

3.7.4.3. Traductor de ecuaciones a sintaxis de Matlab

La traducción es la tercera fase para concluir con el análisis. En esta etapa, se debe hacer una traducción del lenguaje humano al lenguaje de Matlab, por mencionar un ejemplo, el usuario por lo general escribe ecuaciones como la ecuación 1.

$$\mathbf{x}' = \mathbf{a} * (\mathbf{y} - \mathbf{x}) \tag{1}$$

La ecuación 2, muestra la forma en la que Matlab puede ejecutar.

$$\mathbf{x'} = \mathbf{a} * (\mathbf{x(2)} - \mathbf{x(1)}) \tag{2}$$

Donde:

Ecuación (1) es una ecuación diferencial ordinaria.

$\mathrm{x} = \mathrm{f(y)}$ es la variable dependiente.

y = variable independiente.

$\mathrm{x}' = \frac{\mathrm{dx}}{\mathrm{dy}}$ es de derivada de x con respecto a y.

a = es un parámetro.

Para realizar el siguiente código muestra la traducción de la expresión ingresada por el usuario al lenguaje que permite Matlab.

```
private void generar_x()
 {
      int i=0,pos;
      while(id[i]!=null)
      {
         CondIni[i][0]= id[i];
         CondIni[i][1]="x(" + String.valueOf(i+1) + ")";
         i++;
      }
 }
 private void generar_matlab()
 {
     lexemas_matlab=new String[50];
     String tempo="";
     String tempo2="";
     lexemas_matlab[0]=lexemas_A[0];
     int i=1;
     while(lexemas_A[i]!=null)
     {
       int j=0;
       while(CondIni[j][0]!=null)
         {
         tempo=CondIni[j][0]; tempo2=lexemas_A[i];
         if(tempo.equals(tempo2)){
            lexemas_matlab[i]=CondIni[j][1];
            break;
         }
         else lexemas_matlab[i]=lexemas_A[i]; j++;
         }
        if(tempo2.equals(";")){
            i++; lexemas_matlab[i]=lexemas_A[i]; i++;
            lexemas_matlab[i]=lexemas_A[i]; i++;
        }
        else i++;
     }
 }
```

Con estos dos métodos y la ayuda de otros se hace la traducción. El método de generar_x, transforma las variables ingresadas al lenguaje de Matlab, en el segundo: generar_matlab() se generan las ecuaciones completas con las que se va a trabajar ya convertidas las variables ingresadas. El apéndice D muestra el código completo.

3.7.4.4. Análisis sintáctico

Todo lenguaje de programación obedece a unas reglas que describen la estructura sintáctica de los programas bien formados que acepta [1].

- Las gramáticas formales ofrecen ventajas significativas a los diseñadores de lenguajes y a los desarrolladores de compiladores.
- Las gramáticas son especificaciones sintácticas y precisas de lenguajes de programación.
- A partir de una gramática se puede generar automáticamente un analizador sintáctico.
- El proceso de generación automática anterior puede llevar a descubrir ambigüedades.
- Una gramática proporciona una estructura a un lenguaje de programación, para facilitar el generar código y detectar errores.
- Es más fácil ampliar o modificar el lenguaje si está descrito con una gramática.

La clase sintáctico (apéndice C) permite definir objetos analizadores sintácticos que reconocen una sentencia previamente analizada por un analizador léxico.

3.7.4.5. Gramática utilizada por un analizador sintáctico

El estudio de análisis sintácticos para lenguajes basados en gramáticas formales hace más fácil la comprensión del compilador. La formalización del lenguaje que acepta un compilador sistematiza su construcción y permite corregir, quizás más fácilmente, errores de muy difícil localización, como es la ambigüedad en el reconocimiento de ciertas sentencias.

La gramática que acepta el analizador sintáctico es una gramática de contexto libre, puesto que no es fácil comprender gramáticas más complejas, ni construir automáticamente autómatas reducidos que reconozcan las sentencias que aceptan.

Una gramática G queda ser definida por una tupla de cuatro elementos (N, T, P, S), donde:

N = No terminales.

T = Terminales.

P = Reglas de Producción.

S = Axioma Inicial.

Por ejemplo, una gramática no ambigua que reconoce las operaciones aritméticas podría ser la figura 17, la cual es la utilizada para la realización de este trabajo. En ésta se tiene que **N** = {**E**, **T**, **F**} ya que **E**, **T** y **F** aparecen a la izquierda de alguna regla; **T = {id, num, (,), +, *}; P** son las siete reglas de producción, y **S** = **E**, pues por defecto el axioma inicial es el antecedente de la primera regla de producción.

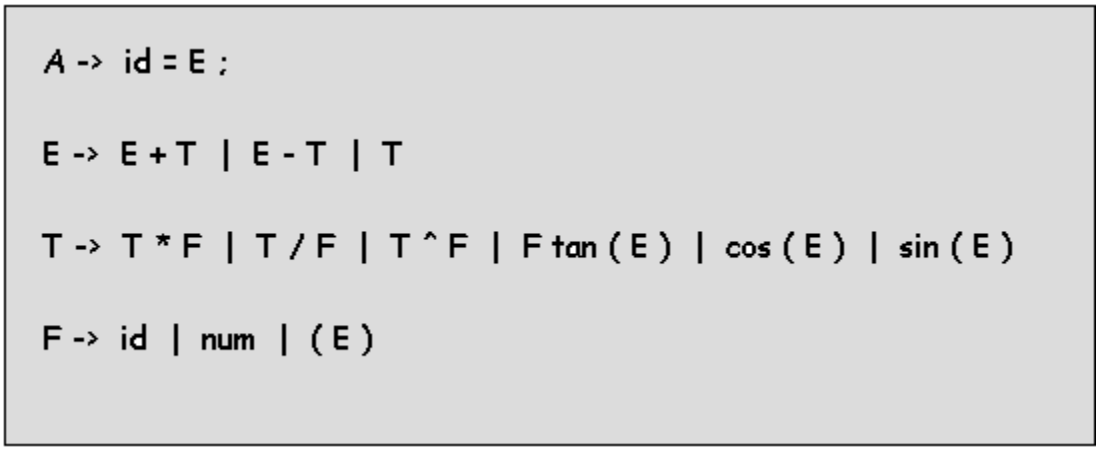


Figura 17. Gramática utilizada en el Toolbox.

Para realizar este sistema se utilizó un **análisis sintáctico descendente predictivo no recursivo** y se utiliza una pila en lugar de hacerlo implícitamente mediante llamadas recursivas.

3.7.4.6. Desarrollo de la aplicación en java

Una vez que se tienen el analizador léxico, analizador sintáctico y traductor para evaluar las ecuaciones ingresadas, lo siguiente es programar el código que

utilizará Matlab para generar las clases que se ocuparán a la hora de programar en Java.

Para una mejor explicación, se muestra un diagrama a bloques en la figura 11, desde el análisis de expresiones hasta la generación de la gráfica esperada.

Debido a que Matlab trabaja todos sus datos en forma de vectores, es necesario cierta conversión entre las variables de tipo dato nativo de Java, tales como int, string, double, etc., a uno de tipo de Matlab, por ejemplo, MW Arrays (MWNumericArray, MWArray, etc.).

En este caso se trabajó con MWNumericArray para enviar la matriz con los valores de las condiciones iniciales ingresadas en el Toolbox.

3.7.4.7. Entorno gráfico

El sistema maneja una interfaz sencilla (figura 18).

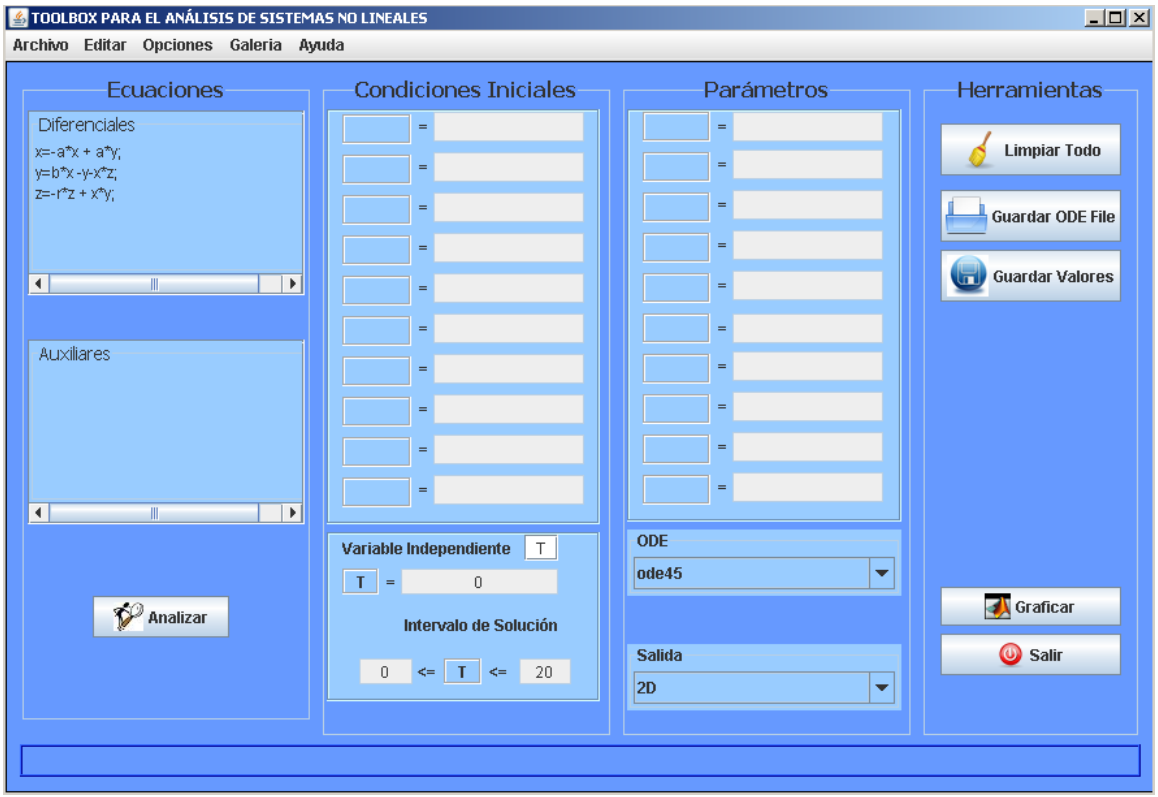


Figura 18. Pantalla principal del Toolbox.

Los pasos para generar las gráficas son los siguientes:

1. Se escribirán las ecuaciones. Si existen ecuaciones auxiliares también se ingresarán, en caso contrario, se deja el espacio en blanco.
2. Presionar el botón de Analizar.
    a. El sistema verifica que las ecuaciones ingresadas estén correctamente escritas y no permite avanzar en coso contario.
    b. EL sistema genera un archivo .m (archivo de Matlab) donde contiene las ecuaciones ingresadas.
3. Se inicializarán las condiciones iniciales y parámetros, así como variable independiente.

4. Se elige el ODE con el que se quiere trabajar, el seleccionado es el ODE45, al igual que la salida, si se quiere en 3D, ya que la predeterminada es en 2D.
5. Por último presionar el botón de “Graficar” para mostrar el resultado (gráficas).
    a. Se genera otro archivo .m, que contiene los valores de las condiciones iniciales y parámetros ingresados. También incluye el ODE con el que va a trabajar y los puntos con los que va a trabajar la gráfica.
    b. Manda llamar las gráficas de Matlab, sin la necesidad de tenerlo instalado.

El resultado es plasmado en una pantalla generada por Matlab como se observa en la figura 19.

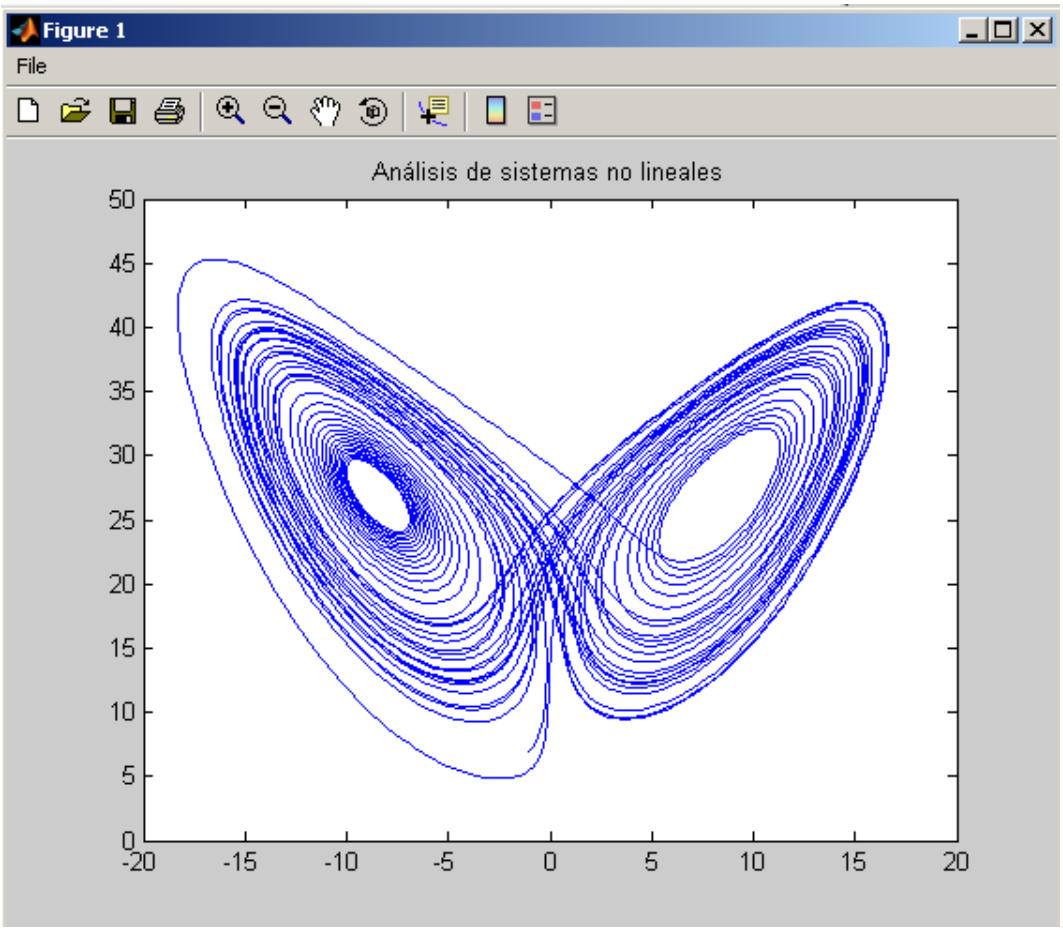


Figura 19. Pantalla generada por Matlab en el Toolbox.

La figura 20 contiende un menú File de la pantalla generada.

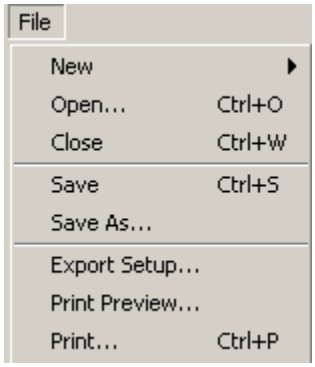


Figura 20. Herramientas del Menú File de la pantalla generada.

El menú contiene las siguientes herramientas:

New: abre un nueva pantalla, pero en este caso no sirve para esta tesis.

Open: abre una figura guardada anteriormente o una que ya disponga el usuario.

Close: cierra la pantalla.

Save: salva la figura o algún cambio que se le haya hecho.

Save as: guarda la figura.

Print: imprime la figura.

La pantalla generada también cuenta con una serie de botones (figura 21), los cuales hacen lo siguiente:

- Acerca la figura.
- Aleja la figura.
- Mano para manejar la figura.
- Rotación en 3D.
- Posicionamiento del cursor para darte la coordenada.
- Activa una barra de colores.
- Inserta la legenda.

Figura 21. Opciones que cuenta la barra de botones.

3.7.4.8. Gráficas generadas con Matlab

Matlab es actualmente el Sistema matemático más utilizado no sólo por sus cálculos matemáticos, además porque tiene un gran potencial de herramientas gráficas y cuenta con una serie de herramientas para poder utilizar sus aplicaciones si la necesidad de tenerlo instalado.

Estas herramientas son:

- Matlab Compiler.
- Matlab Builder Java.

Matlab Compiler

Permite compartir las aplicaciones de Matlab como un ejecutable o una biblioteca compartida.

Sus principales características son:

- Distribuye ejecutables independientes y componentes de software libre.
- Permite incorporar los algoritmos basados en aplicaciones desarrollados con otros lenguajes y tecnologías.
- Encripta los códigos de Matlab para que no se puedan ver o modificar.

Matlab Builder Java

Hace clases Java, desde los programas de Matlab. Estas clases que se generan son integradas en el programa principal que se está realizado en Java.

Principales características:

- Es de software libre.
- Permite hacer zoom en la figura, rotación a través de la interfaz.
- Conversión automática entre Java y Matlab.

La figura 22 muestra las variantes de Matlab Compiler:

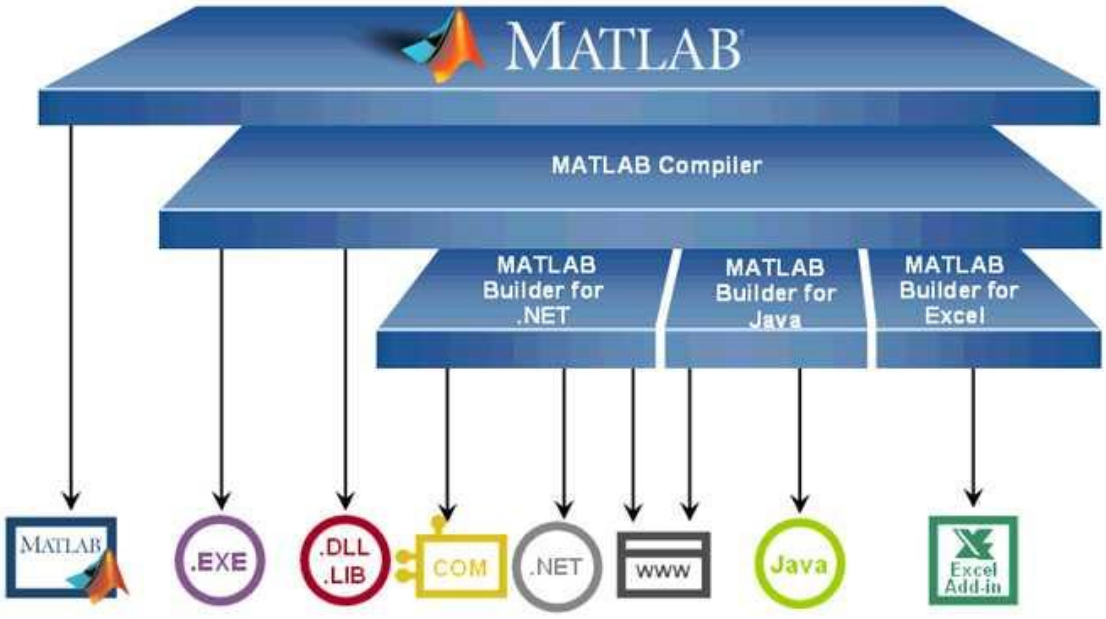


Figura 22. Diagrama de las variantes de Matlab Compiler y sus herramientas.

Para la realización de esta tesis como se menciónó anteriormente se utiliza Matlab Builder para Java, en la modalidad para escritorio.

Deploytool

Es el comando de Matlab con el cual abre la interfaz gráfica para trabajar con Matlab Builder Java y Matlab Compiler. Las siguiente serie de figuras (15-19) muestran la forma de realizar las clases con el cuales se trabajará en Java.

Hay dos formas de ingresar a Deployment Tool:

1. Escribir directamente en la ventana de trabajo de Matlab la sentencia deploytool.
2. Desde inicio (start), Matlab, Matlab Builder JA y Deployment Tool.

La figura 23 muestra la forma 1 de abrir deploytool.

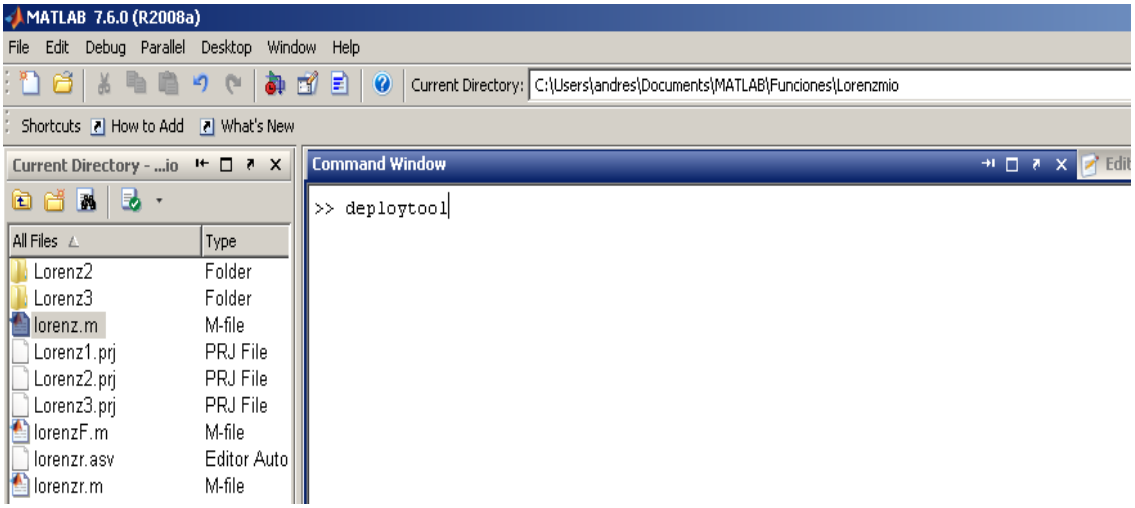


Figura 23. Comando Deploytool en Matlab.

En la figura 24 muestra la ventana principal de Deployment tool, donde se puede abrir un proyecto realizado o hacer uno nuevo.

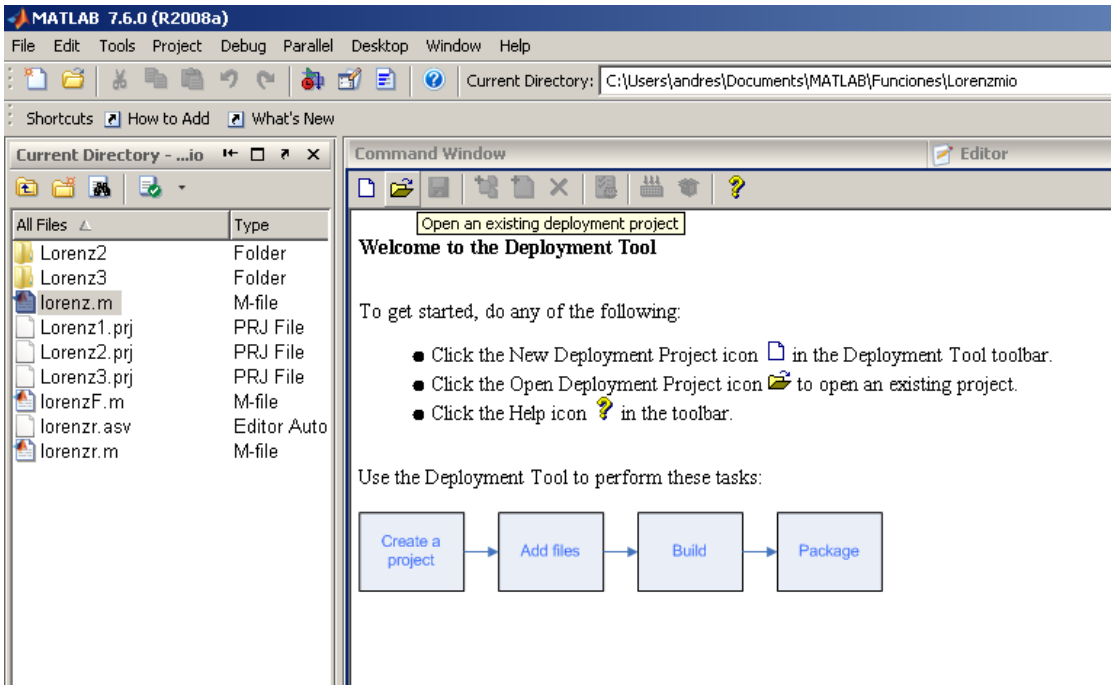


Figura 24. Ventana principal de la herramienta de Deployment.

En la figura 25 se elige el proyecto con el cual se va a trabajar, como se mencionó anteriormente, Matlab cuenta con diferentes herramientas para trabajar como Excel, .NET o Java, y se elige esta última con la que se trabaja en este proyecto.

Una vez que se eligió se le da el nombre al proyecto.

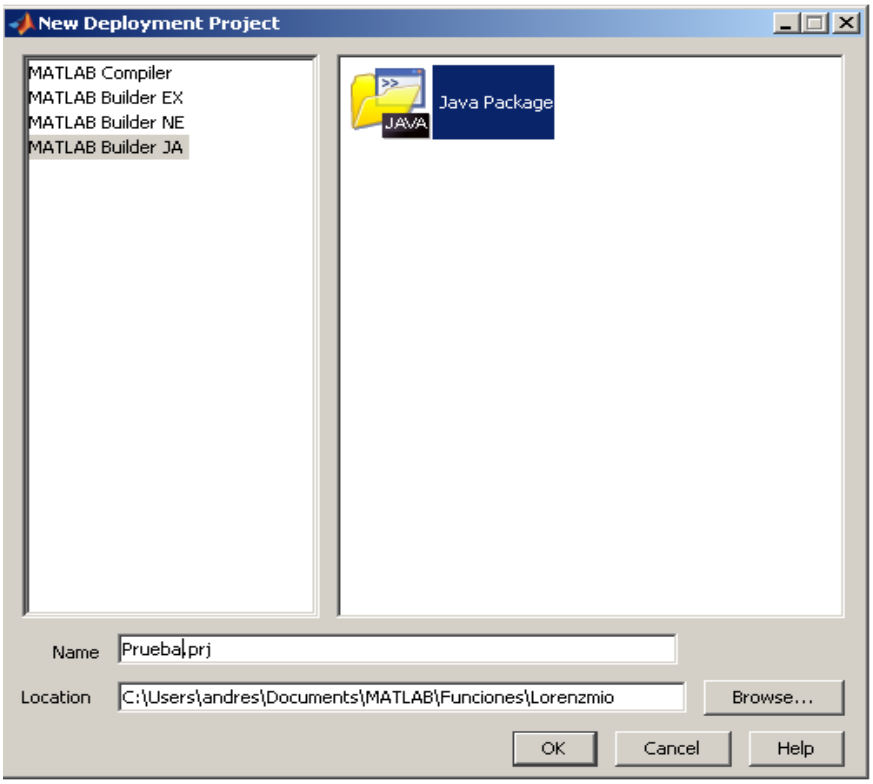


Figura 25. Muestra los diferentes proyectos que se pueden realizar.

Después se eligen las funciones con extensión ".m" de Matlab previamente programadas y se insertan en el proyecto para compilarse, como lo muestra la figura 26.

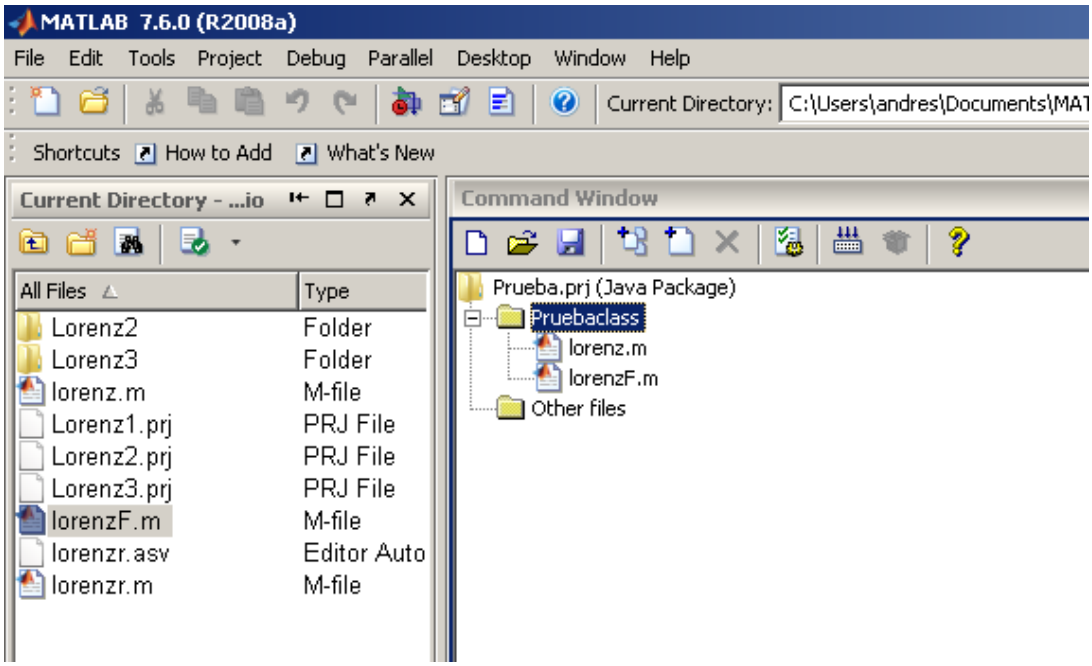


Figura 26. Se da el nombre a la clase y se agregan los archivos de Matlab.

Una vez que se agregaron las funciones se compila el proyecto en el ícono que muestra la figura 27.

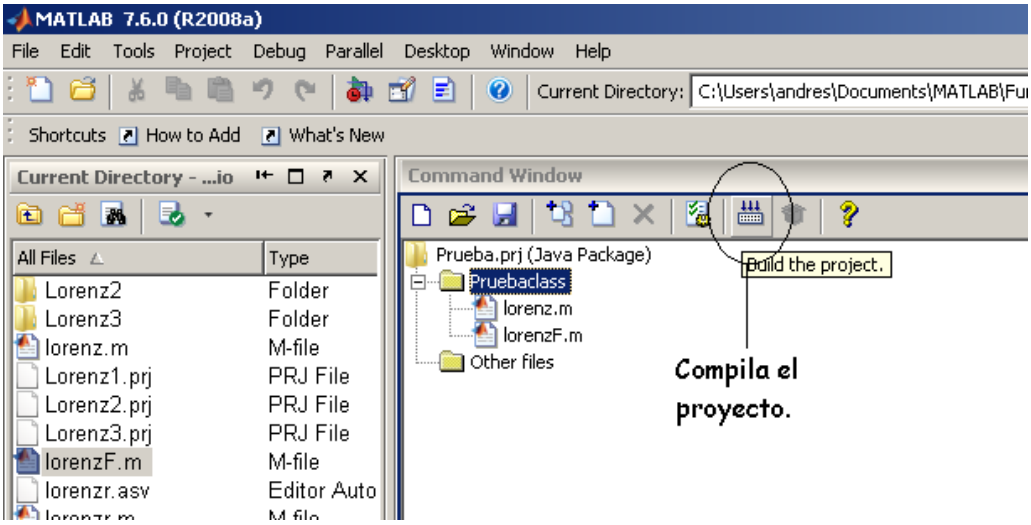


Figura 27. Muestra el botón para la compilación del proyecto.

Una vez que se compiló con éxito el proyecto, se exportan las librerías generadas por Matlab Builder Java al sistema.

3.8. PROBLEMÁTICA CON EL ODE FILE

El ODE File es la función en Matlab que contiene las ecuaciones diferenciales con las cuales se va a generar la gráfica.

Matlab no puede definir funciones completas dentro de un script, así que para escribir cualquier expresión matemática no básica es necesario realizar un archivo.

El problema surge al generar el archivo del Toolbox, éste no es reconocido por las clases de Matlab, ya que en Matlab Builder, primero es compilado el archivo ODE File y luego es llamado del Toolbox.

La solución a este problema es una sentencia de Matlab llamada Inline. Lo que hace es interpretar una cadena de texto en la que se escriben las funciones y las conecta a una variable, como si de un puntero a una función se tratara. Su sintaxis es la siguiente:

```
inline(expr);
inline(expr,arg1,arg2,...);
inline(expr,n);
```

Ejemplo:

*g = inline('sin(alpha*x)','x','alpha');*

Con esta sentencia se puede compilar un sólo archivo en Matlab Builder como lo muestra la Figura 28, que a diferencia de la figura 27 compilaban dos archivos: el ODE File y la que contiene el valor de los parámetros. Así sólo se envía la ecuación como cadena a esta sentencia y el problema es resuelto.

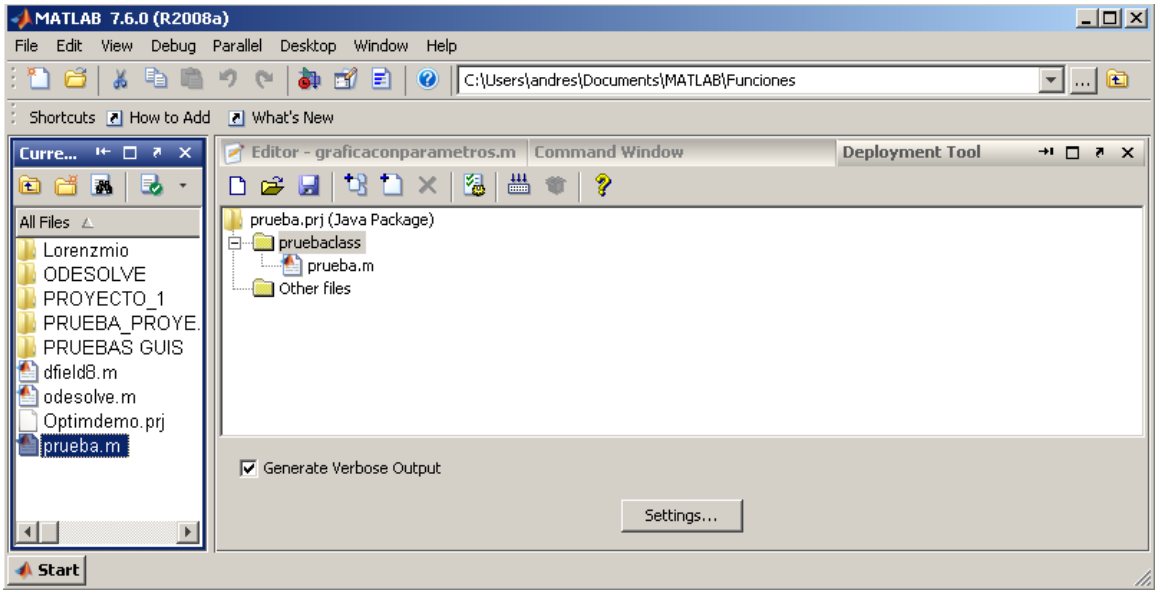


Figura 28. Compilación de un sólo archivo.

CAPÍTULO IV

RESULTADOS Y DISCUSIÓN

4.1. Introducción

Para obtener buenos resultados es necesario hacer las pruebas precisas, es decir encontrar diferencias entre el comportamiento esperado, especificado por los modelos del sistema, y el comportamiento observado del sistema.

El objetivo es diseñar pruebas que manifiesten los defectos que hay en el sistema y que revelen los problemas, por lo tanto, se realizan las pruebas para quebrantar al sistema.

En el caso de esta tesis hay tres puntos importantes en los que se realizaron las pruebas:

1. Verificar que las ecuaciones estén escritas correctamente.
2. Desglose de condiciones iniciales y parámetros para inicializar.
3. Que la gráfica resultante este correcta.

Las pruebas se hicieron con las ecuaciones de Lorenz. En 1963 Edward Lorenz trabajaba en unas ecuaciones, las ecuaciones mundialmente conocidas como

ecuaciones de Lorenz, que esperaba predijeran el tiempo en la atmósfera, y trató mediante los ordenadores ver gráficamente el comportamiento de sus ecuaciones. Aunque se les llamaran ordenadores de alta velocidad, los ordenadores por aquella época eran muy lentos, por lo que Lorenz se fue a tomar un té mientras el ordenador hacía los cálculos, y al volver se encontró con una figura que ahora se conoce como atractor de Lorenz.

4.2. Pruebas con el analizador de expresiones

Las ecuaciones 3, 4 y 5, ingresadas son el sistema de Lorenz:

$$x = -a * x + a * y; \quad (3)$$

$$y = b * x - y - x * z; \quad (4)$$

$$z = -r * z + x * y; \quad (5)$$

Como se mencionó en el capítulo III, el sistema empieza con un análisis léxico y pasa a lo que es el análisis sintáctico donde verifica que estén correctamente escritas las ecuaciones.

En la figura 29 se puede apreciar el resultado de evaluar las ecuaciones con análisis exitoso.

Figura 29. Análisis exitoso.

Si el sistema encuentra algún error durante el proceso del análisis de expresiones, arroja un aviso de alerta que en alguna de las ecuaciones ingresadas hay un error. La figura 30 muestra un análisis no exitoso y menciona el número de la ecuación que está ingresada incorrectamente.

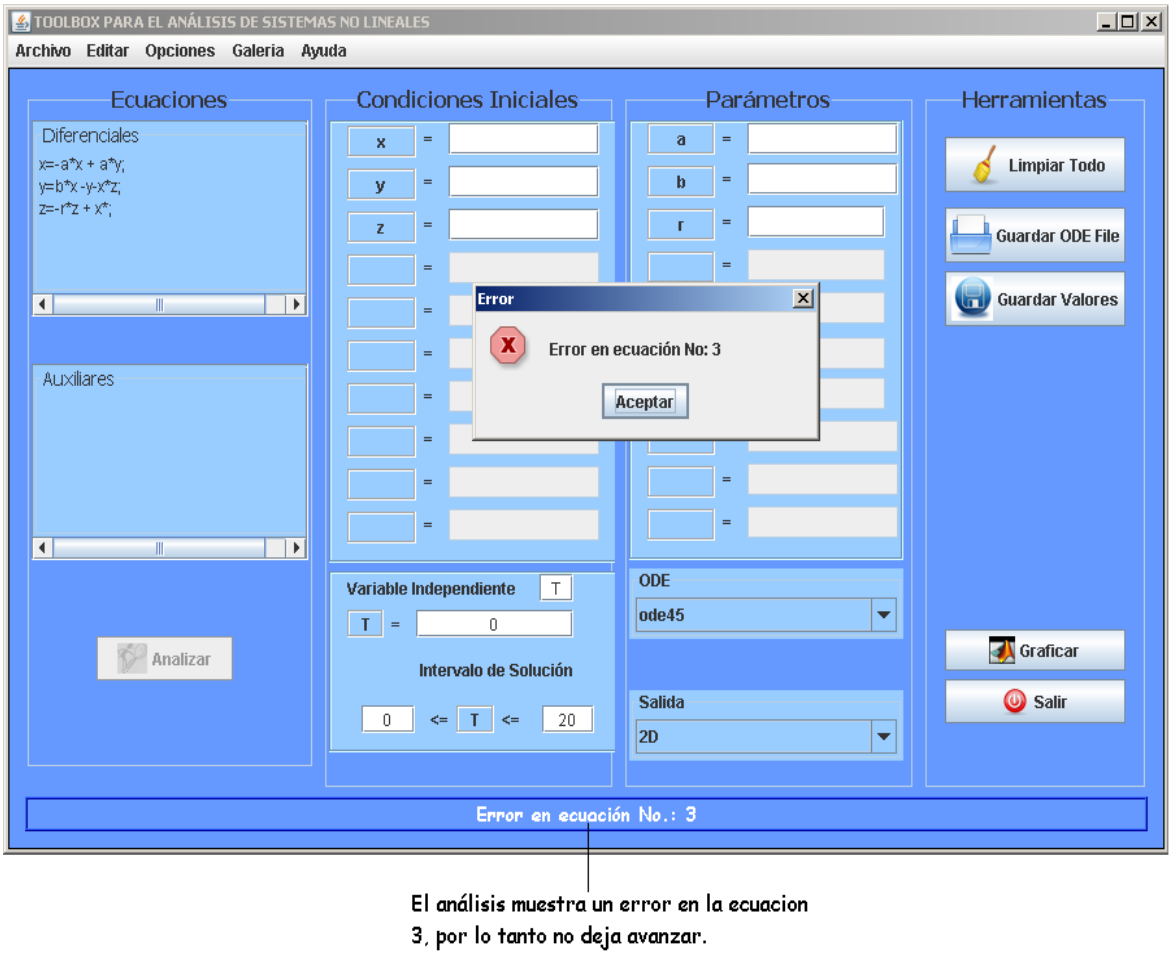


Figura 30. Análisis no exitoso.

En caso de que el sistema encuentre un error, el usuario deberá escribir correctamente la ecuación y presionar de nuevo el botón de “Analizar”.

4.3. Condiciones y parámetros a inicializar

Se observa que en las ecuaciones de Lorenz mencionadas en el capítulo anterior (3), (4) y (5), cuentan con las siguientes condiciones iniciales:

x, y, z.

Los parámetros:

a, b, r.

La figura 31 muestra las condiciones iniciales y parámetros listos para inicializarse una vez que estén bien escritas las ecuaciones.

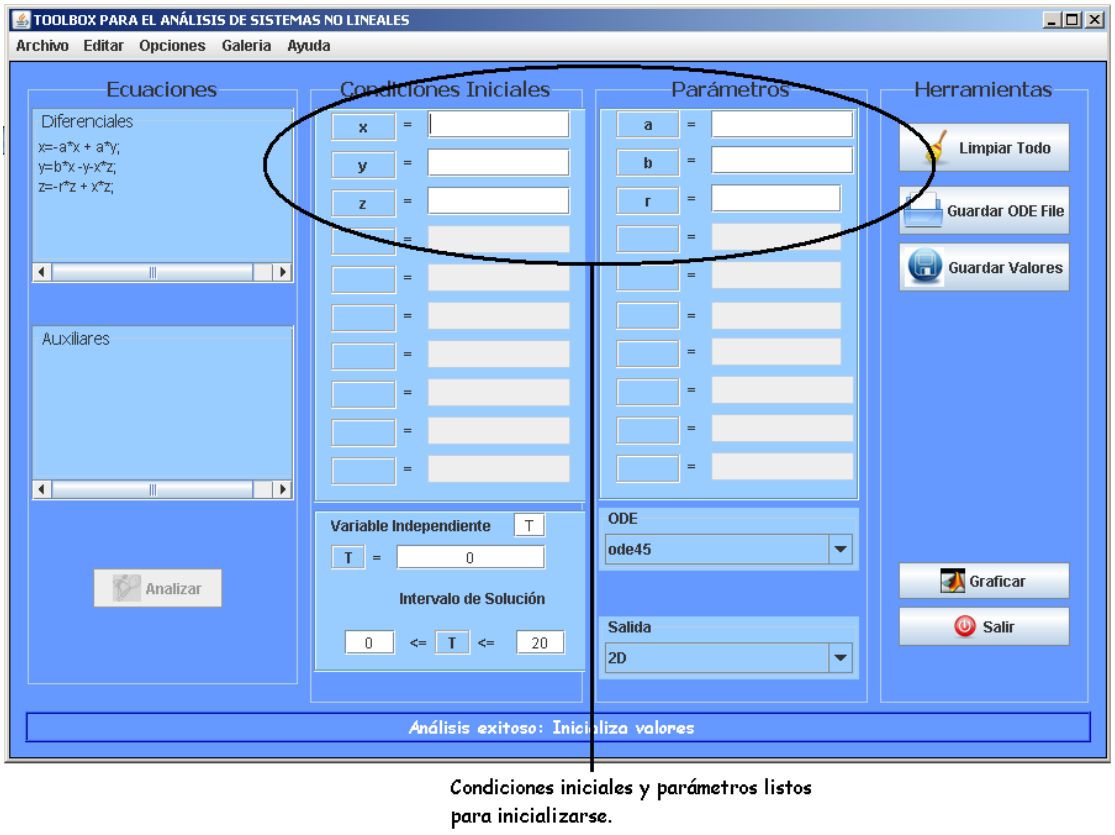


Figura 31. Condiciones iniciales y parámetros listos para su inicialización.

Estos apartados para inicialización de variables, no permite ingresar letras u otros dígitos, solo acepta números y guiones (-) para negar las variables.

4.4. Pruebas con las gráficas generadas

Los valores con los que se realizan las pruebas son los mismos en todos los programas con los que se trabaja, estos valores se encuentran en la tabla 2:

Tabla 2. Valores para realizar las pruebas.

Condiciones Iniciales:	**Parámetros:**	**Tolerancia:**	**Intervalo de Solución:**	**ODE:**	**Salida:**
x=-8 y=-8 z=24	a=10 b=28 c=2.6666667	1e -6	De 0 a 20	ode 45	En 2D variable x contra la variable z.

La gráfica que se genera con el Toolbox es comparada con los siguientes programas:

1. Matlab.
2. Dinamics Solver.
3. XPPAUT.
4. Simnom.

Lo que se observa en las siguientes figuras (32 - 36) son el resultado de la evolución en tiempo de las variables de estado de las ecuaciones, y se pueden graficar en tiempo o con respecto a una variable de elección del usuario.

La figura 32 muestra la gráfica generada con el Toolbox.

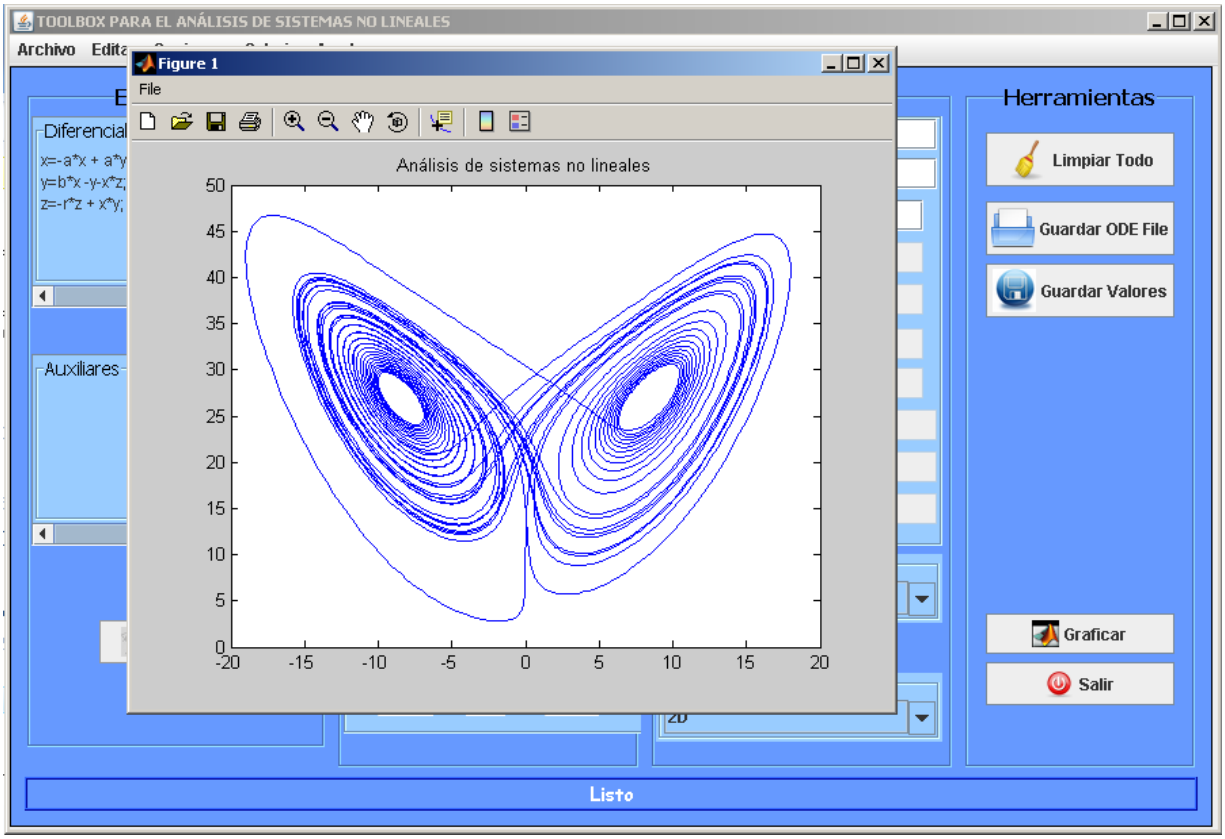


Figura 32. Gráfica generada con el Toolbox.

La figura 33 es generada desde Matlab.

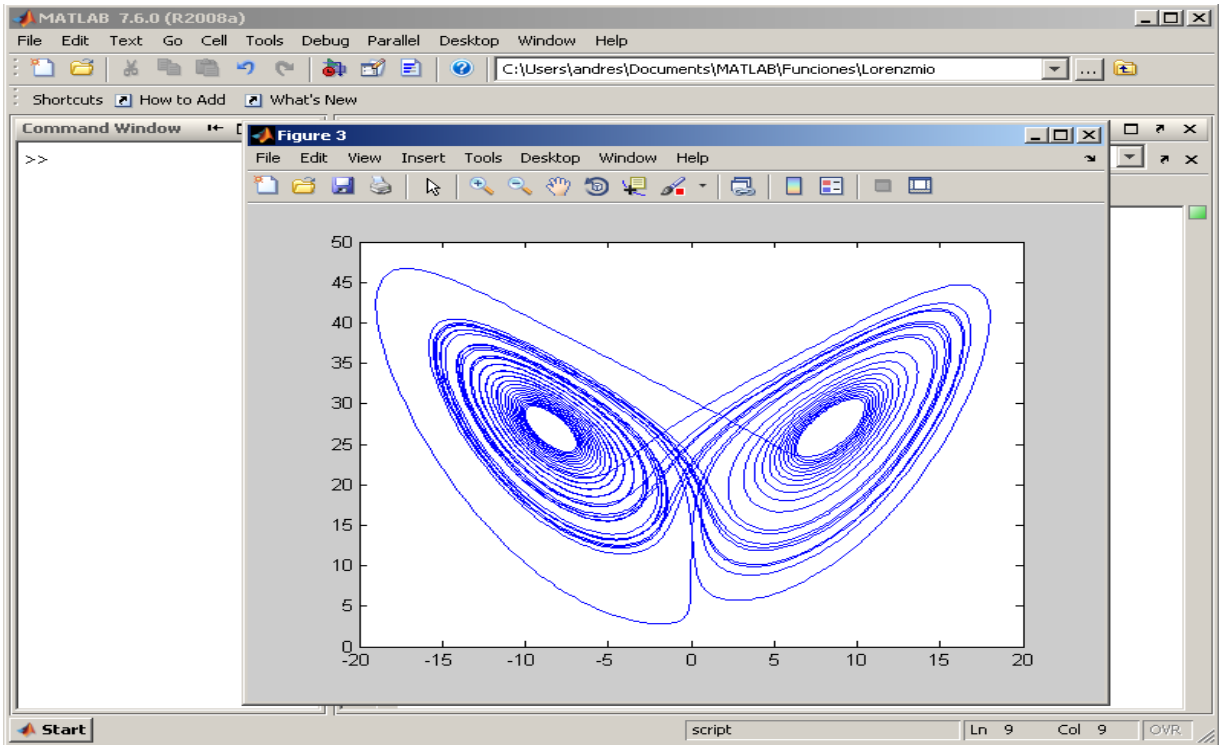


Figura 33. Gráfica generada desde Matlab.

La siguiente figura mostrada, número 34 es generada con Dinamics Solver.

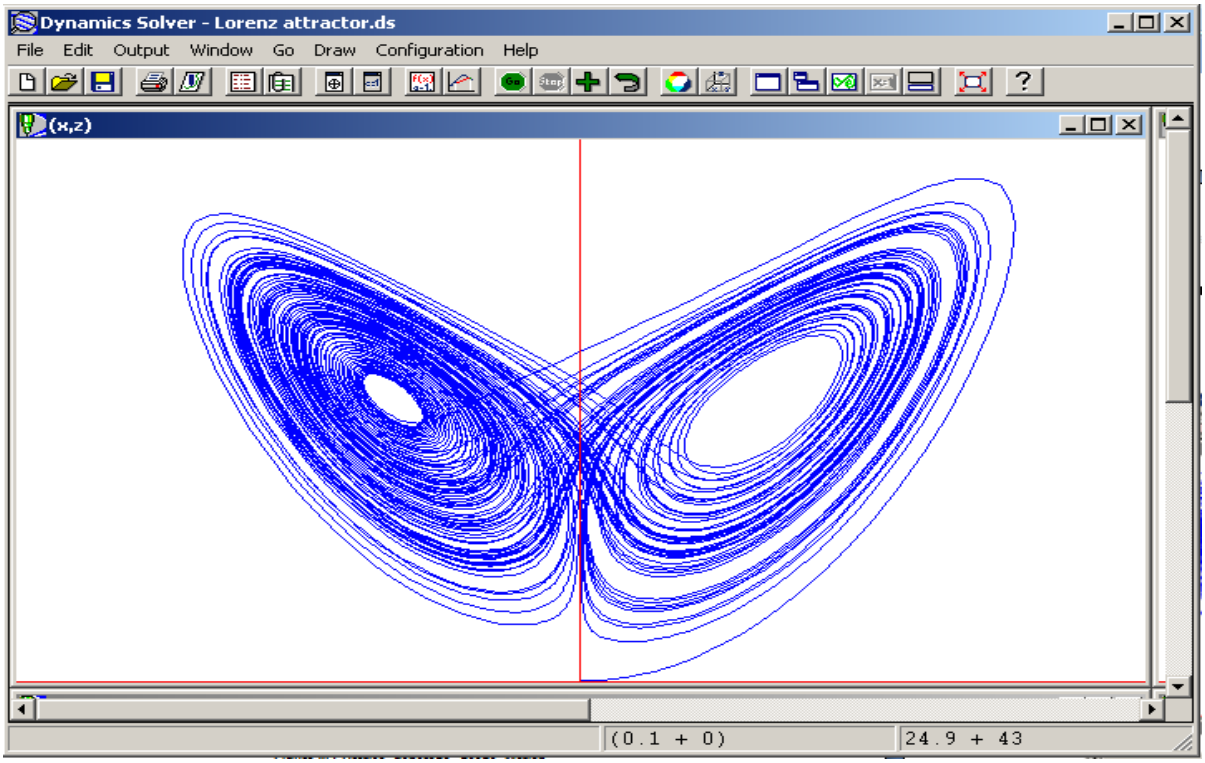


Figura 34. Gráfica generada con Dinamics Solver.

XPPAUT proporciona el resultado siguiente, figura 35.

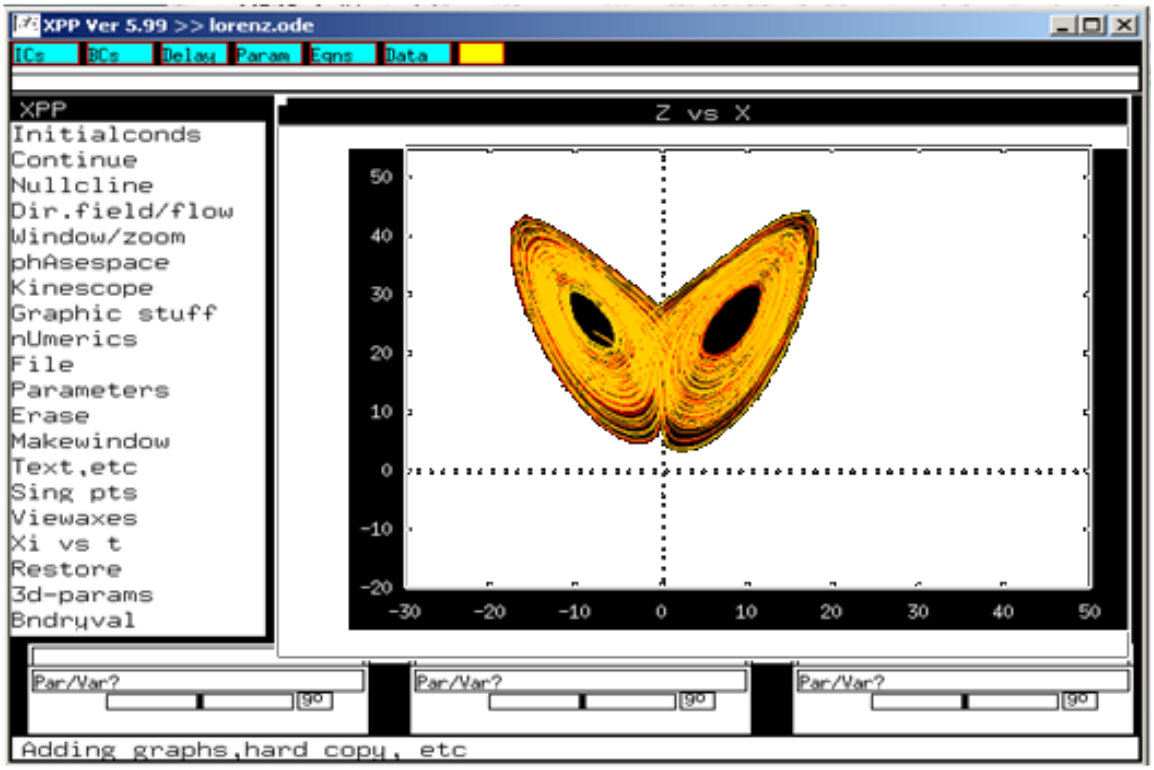


Figura 35. Gráfica generada con XPPAUT.

Por último, la gráfica con la que se compara es con el programa Simnon, mostrado en la figura 36.

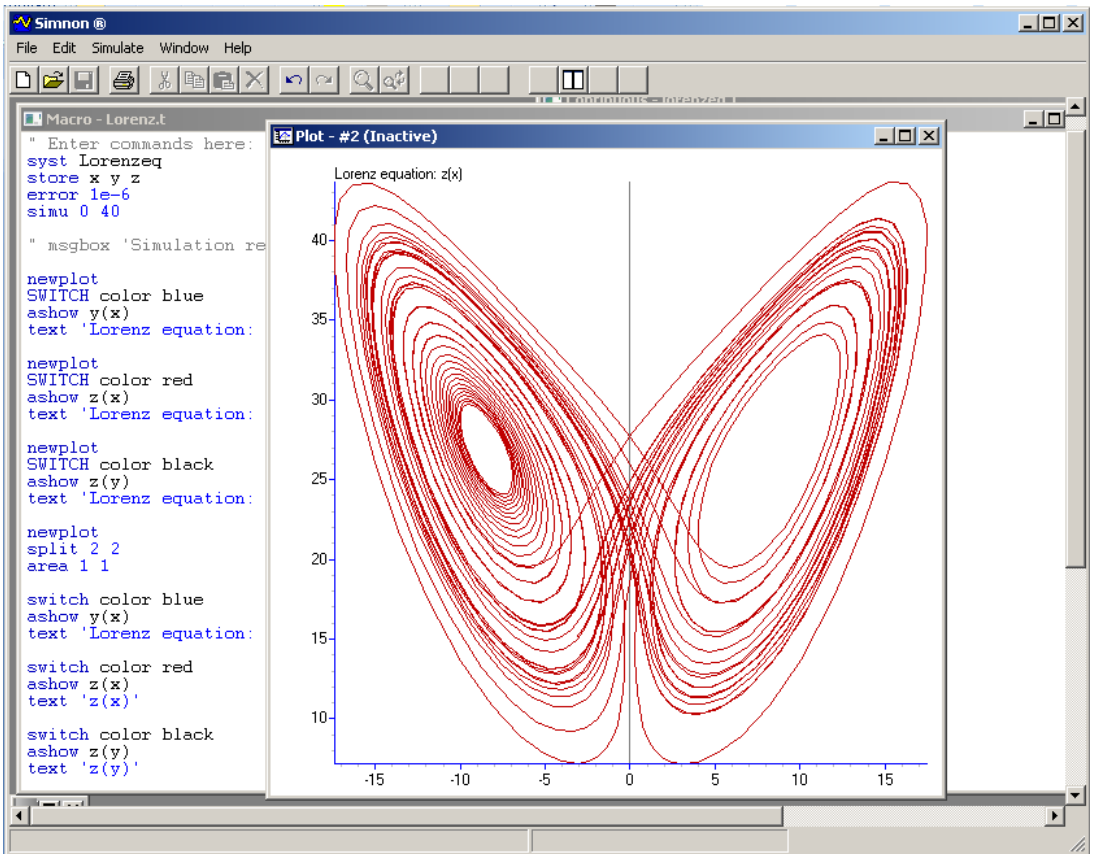
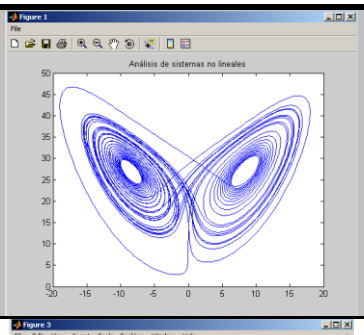
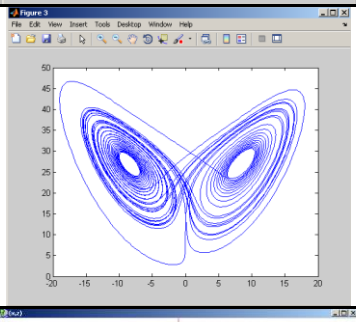
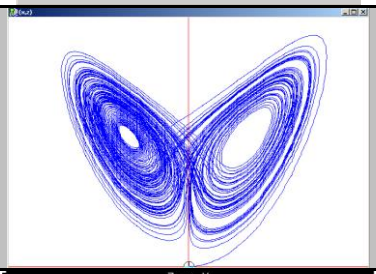
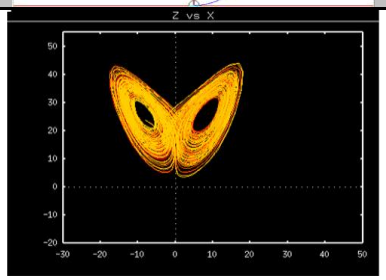
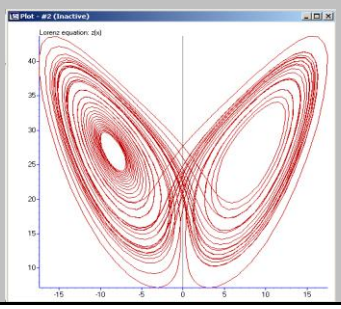


Figura 36. Gráfica generada con Simnon.

La tabla 3 muestra las gráficas generadas por los distintos programas para dar una mejor vista de las graficas resultantes.

Tabla 3. Concentrado de graficas generadas.

Programa:	**Gráfica:**
Toolbox para la solución de sistemas no lineales.	
Matlab.	
Dinamics Solver.	
XPPAUT.	
Simnon.	

Como se puede observar, no hay mucha diferencia entre las gráficas, en la grafica generada por Dinamics Solver la diferencia es por el intervalo de solución que no tiene tope, inicia y termina cuando el usuario presiona el botón de ”stop”.

Las gráficas generadas por los distintos programas, son cualitativamente iguales y sirven para ilustrar que el resultado es comparable con el Toolbox, no son iguales porque se simula un estado caótico, el cual tiene como una de las propiedades sensibilidad a las condiciones iniciales, por lo que al usar diferentes algoritmos numéricos producen diferentes resultados cuantitativos, sin embargo, cualitativamente representan la misma dinámica.

CAPÍTULO V

CONCLUSIONES

5.1. Introducción

En este trabajo se presenta una herramienta que muestra la solución de sistemas no lineales, que se puede instalar en cualquier equipo de escritorio con sistema operativo Windows y la posibilidad de migrarlo al sistema operativo Linux o Mac OS, ya que esta realizado con el lenguaje de programación Java, éste permite trabajar con cualquiera de los sistemas operativos e interactúa con el software científico llamado Matlab.

5.2. Conclusiones

Los resultados obtenidos en el capítulo IV son los esperados, ya que se analizó, diseño e implementó un programa que analiza los sistemas no lineales y genera las gráficas que el usuario necesita sin programar una sola línea. El programa cuenta con 3 módulos principales:

1. Inserción de ecuaciones.
2. Inicialización de parámetros, condiciones iniciales e intervalo de solución.
3. Generar gráfica.

Por lo anterior se concluye que se cumple satisfactoriamente con el objetivo general del trabajo:

- Objetivo general: realizar un sistema para análisis de sistemas no lineales que codifique las fórmulas, muestre parámetros para su inicialización y plasme en otra ventana la gráfica generada con Matlab.

Por lo tanto, es posible la solución de sistemas no lineales sin programar, sólo se ingresan las ecuaciones e inicializan y el sistema muestra la gráfica. Con la posibilidad de agregar más funciones al sistema por ser un sistema de software libre.

Al cumplir con el objetivo general del proyecto, se establece que la hipótesis planteada para el proyecto se cumple satisfactoriamente:

- Hipótesis: el análisis de sistemas no lineales es posible solucionarlo al utilizar un sistema que interactúe con Matlab para mostrar la solución, sin la necesidad de programar una sola línea.

Al haber validado la hipótesis del proyecto en forma positiva, se puede responder a las preguntas de investigación formuladas en el capítulo I:

- Preguntas de investigación:

1. ¿Cómo resuelve el usuario no experto en programación el análisis de los algoritmos de sistemas no lineales?

2. ¿Cómo se puede ahorrar tiempo el investigador o estudiante, en buscar una herramienta de análisis de sistema no lineales que se adapte a sus necesidades?

Como respuesta a las preguntas de investigación, es que el usuario que necesita resolver el análisis de sistemas no lineales sin tener conocimientos de programación y para ahorrarse tiempo en buscar un programa que contenga el algoritmo que esta buscado, es posible con el sistema desarrollado en este proyecto, ya que no es necesario programar una sola línea y puede ingresar las ecuaciones que desea para obtener el resultado esperado.

Por lo anterior mencionado, la problemática de tener un sistema donde el usuario necesita programar y muchas veces éste no contiene el algoritmo que esta buscado, se encuentra en la necesidad de buscar otro programa, y de esta forma se pierde tiempo, así que el problema se atiende con el desarrollo del Toolbox para el análisis de sistemas no lineales presentado en esta tesis.

5.3. Trabajos a futuro

Los trabajos a futuro son:

- Migrar el sistema a otros sistemas operativos como Linux y MAC OS.
- Ingresar al sistema los algoritmos conocidos, como Lorenz, Liapunov, etc.

- Pasar el sistema a la red, para no tener la necesidad de tener instalado el programa en la computadora, que el usuario con una conexión a internet, pueda utilizar la herramienta.

Al ser un programa de código libre, el sistema está abierto para hacerse cualquier modificación, según lo que el usuario necesite.

BIBLIOGRAFÍA

1. Kenneth C. Louden, Construcción de compiladores. Principios y prácticas. International Thompson ed., 2004.
2. Bruegge, Bernard & Dutoit, Allen H., Ingeniería de software orientado a objetos. Pearson educación, México, 2002.
3. Peckham y Scott J. Lloyd, Practicing Software Engineering in the 21st Century, Hershey, PA: IRMA Press, 2003.
4. Dr. Michael Small: Profesor de la Universidad de Politécnico de Hong Kong. Línea de investigación: Dinámica no lineal y sistema de modelado. (plática por correo electrónico: 29 /05 / 2009.)
5. Dr. Hugo G. González: Director del Depto. de Ing. Mecánica y Electrónica, Escuela de Ingeniería y Arquitectura, Tecnológico de Monterrey, Campus Puebla. (plática por correo electrónico: 01 /06 / 2009.)
6. J.C. Sprott, Chaos and Time-Series Analysis, London: Oxford Univ. Press, 2003.
7. Jacobson, I. The Unified Software Development Process., Addison-Wesley, Pearson Education, 2005.
8. Joyanes Aguilar Luis. Estructuras de datos en Java. McGraw-Hill, 2008.
9. Keegan, Patrick. NetBeans IDE field guide: developing desktop, Web, enterprise, and mobile applications**.** Prentice Hall, 2006.
10. Myatt, Adam**.** Pro NetBeans IDE 6 rich client platform edition, 2008.

11. Michael Small, Applied nonlinear time series analysis: applications in physics, physiology and finance, Edition: illustrated, World Scientific, 2005.

MEDIOS ELECTRÓNICOS

12. http://www.worldlingo.com/ma/enwiki/es/Henri_Poincar%C3%A9. Consultada en noviembre de 2008.
13. http://dialnet.unirioja.es/servlet/articulo?codigo=2228041. Consultada en noviembre de 2008.
14. http://personales.unican.es/gutierjm/docs/tut_SistNoLin.pdf. Consultado en noviembre de 2008.
15. http://math.rice.edu/~dfield/. Consultado el 10 de noviembre de 2008.
16. http://www.sld.cu/sitios/complejidad/temas.php?idv=19342. Consultado en diciembre de 2008.
17. http://personales.ya.com/casanchi/ref/tcaos01.pdf. Consultado en diciembre de 2008.
18. http://www.ant4669.de/relatedprojetcs.html. Consultado en octubre de 2008.
19. http://amath.colorado.edu/faculty/jdm/faq-%5B5%5D.html. Consultado en octubre de 2008.
20. http://www.escritos.buap.mx/escri25/rlema.pdf. Consultado en octubre de 2008.
21. http://controlweb.cyc.ull.es. Consultada en marzo 2009
22. http://www.mathworks.com. Consultado desde enero de 2009.

23. http://www.oracle.com/technetwork/indexes/downloads/index.html. Consultado desde enero 2009.
24. http://ciencianet.com.ar/138/el-problema-de-fermi-pasta-y-ulam-un-peque-o-descubrimiento. Consultado en enero 2009.

APÉNDICE A

Permiso para poner una figura de Matlab en la tesis

RE: Consulta

De: **Rozzana Almaraz** (ralmaraz@multion.com.mx)
Enviado: miércoles, 05 de mayo de 2010 03:53:43 p.m.
Para: chano_q@hotmail.com

Distinguido Sr. Camacho,

Buen día,

Solo se le pide que al poner el logotipo haga referencia DERECHOS RESERVADOS DE LA MARCA MATHWORKS INC.

Saludos cordiales

Rozzana Almaraz
Grupo de Ventas
ralmaraz@multion.com.mx
Tel: (55) 5559-4050 **Ext. 189**
Fax: (55) 5559-4048 Country Code: 52
SIN COSTO 01-800-MULTION (solo México)

MultiON Consulting S.A. de C.V.
Insurgentes Sur 1236 Int. 301 México D.F., 03200
Software Científico y Técnico. **http://www.multion.com.mx**
MATLAB&Simulink, MAPLE, STATA, STATISTICA, MINITAB, XLSTAT, DesignExpert, EViews,
EndNote, ChemOffice, Spartan, Origin, INTEL Fortran y C/C++,
QUANSER, Atlas.ti, NVivo, LINDO . . . para mas productos ver en www.multion.com.mx

APÉNDICE B

Clases del analizador léxico

```
public class Lexico {

    final int TOKREC = 7;
    final int MAXTOKENS = 50;
    private String[] _lexemas;
    private String[] _tokens;
    private String _lexema;
    private int _noTokens;
    private int _i;
    private int _iniToken;
    private Automata oAFD;
    public int comas;



    public Lexico() // constructor por defecto
    {
    _lexemas = new String[MAXTOKENS];
    _tokens = new String[MAXTOKENS];


    oAFD = new Automata();
    _i = 0;
    _iniToken = 0;
    _noTokens = 0;
    }
    public void Inicia()
    {
        comas=0;
        _i = 0;
        _iniToken = 0;
        _noTokens = 0;
        _lexemas = new String[MAXTOKENS];
        _tokens = new String[MAXTOKENS];
```

```
}
public void Analiza(String texto)
{
    Boolean recAuto;
    int noAuto;
    while (_i < texto.length())
    {
    recAuto=false;
    noAuto=0;
    for(;noAuto<TOKREC&&!recAuto;)
    if(oAFD.Reconoce(texto,this,noAuto))
    recAuto=true;
    else
    noAuto++;
    if (recAuto)
    {
    _lexema = texto.substring(_iniToken, _i);
    switch (noAuto)
    {
    //-------------- Automata delim--------------
    case 0 :// _tokens[_noTokens] = "delim";
    break;
    //-------------- Automata Id--------------
    case 1 :if(EsId())  _tokens[_noTokens] = "id";
           else   _tokens[_noTokens] = _lexema;
    break;
    //-------------- Automata OpAsig--------------
    case 2 : _tokens[_noTokens] = _lexema;
    break;
    //-------------- Automata oparit--------------
    case 3 : _tokens[_noTokens] = _lexema;
    break;
    //-------------- Automata num--------------
    case 4 : _tokens[_noTokens] = "num";
    break;
    //-------------- Automata sep--------------
    case 5 : _tokens[_noTokens] = _lexema;
    break;
    //-------------- Automata termInst--------------
    case 6 : _tokens[_noTokens] = _lexema;
            comas++;

    break;
    }
    if (noAuto != 0)
    _lexemas[_noTokens++] = _lexema;
```

```
    }
    else
    _i++;
    _iniToken = _i;
    }
} // fin del método Analiza()


public int getI()
    {
    return _i;
    }
public void setI(int valor)
    {
    _i=valor;
    }
public int getIniToken()
    {
    return _iniToken;
    }
public int NoTokens()
    {
        return _noTokens;
    }
public String Token(int a)
    {
    return _tokens[a];
    }
public String[] Tokens()
    {
    return _tokens;
    }
public String[] Lexemas()
    {
    return _lexemas;
    }
public int Comas()
    {
        return comas;
    }
private boolean EsId()
     {
        String[] palres ={ "sin", "cos","tan","SIN","COS","TAN"};
        for (int i = 0; i < palres.length; i++)
          if (_lexema.equals(palres[i]))
```

```
            return false;
        return true;
    }
    public void Anade(String valTok, String valLex)
    {
        _tokens[_noTokens] = valTok;
        _lexemas[_noTokens++] = valLex;
    }
} // fin de la clase Lexico
public class Automata {

    String _textoIma;
    int _edoAct;
    private char SigCar(int i) {
        if (i == _textoIma.length())
        return 'i';
        else
        return _textoIma.charAt(i);
    }
    public Boolean Reconoce(String texto,Lexico oAnaLex,int noAuto){
        char c;
        _textoIma = texto;
        String lenguaje;
        switch (noAuto)
        {
        //-------------- Automata delim--------------
        case 0 : _edoAct = 0;   break;
        //-------------- Automata Id--------------
        case 1 : _edoAct = 3;   break;
        //-------------- Automata OpAsig--------------
        case 2 : _edoAct = 6;   break;
        //-------------- Automata oparit--------------
        case 3 : _edoAct = 9;   break;
        //-------------- Automata num--------------
        case 4 : _edoAct = 11;  break;
        //-------------- Automata sep--------------
        case 5 : _edoAct = 16;  break;
        //-------------- Automata termInst--------------
        case 6 : _edoAct = 18;  break;
        }
        while(oAnaLex.getI()<=_textoIma.length())
        switch (_edoAct)
        {
            //-------------- Automata delim--------------
            case 0 : c=SigCar(oAnaLex.getI());
            oAnaLex.setI(oAnaLex.getI()+1);
```

```
        if ((lenguaje=" \n\r\t").indexOf(c)>=0) _edoAct=1;
        else
        {
           oAnaLex.setI(oAnaLex.getIniToken());
            return false;
        }break;

        case 1 : c=SigCar(oAnaLex.getI());
                 oAnaLex.setI(oAnaLex.getI()+1);
                 if ((lenguaje=" \n\r\t").indexOf(c)>=0) _edoAct=1;
        else  if ((lenguaje="*+-/^.0123456789="
                         +"ABCDEFGHIJKLMNOPQRSTUVWXYZ()"
                         +"abcdefghijklmnopqrstuvwxyz;").indexOf(c)>=0) _edoAct=2;
        else
           {
              oAnaLex.setI(oAnaLex.getIniToken());
           return false;
           } break;

        case 2 : oAnaLex.setI(oAnaLex.getI()-1);
        return true;

        //-------------- Automata Id----------
        case 3 : c=SigCar(oAnaLex.getI());
        oAnaLex.setI(oAnaLex.getI()+1);

if((lenguaje="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy
z").indexOf(c)>=0) _edoAct=4; else
              if ((lenguaje="_").indexOf(c)>=0) _edoAct=4;
              else
              {
                 oAnaLex.setI(oAnaLex.getIniToken());
                 return false;
              }   break;

        case 4 : c=SigCar(oAnaLex.getI());
        oAnaLex.setI(oAnaLex.getI()+1);

if((lenguaje="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxy
z").indexOf(c)>=0) _edoAct=4; else
        if ((lenguaje="_").indexOf(c)>=0) _edoAct=4; else
        if ((lenguaje="0123456789").indexOf(c)>=0) _edoAct=4;
        else  if ((lenguaje=" *+-/^.0123456789="
                         +"ABCDEFGHIJKLMNOPQRSTUVWXYZ()"
                         +"abcdefghijklmnopqrstuvwxyz;\n\t\r\f").indexOf(c)>=0)
_edoAct=5;
```

```
else
{ oAnaLex.setI(oAnaLex.getIniToken());
return false; }
break;
case 5 : oAnaLex.setI(oAnaLex.getI()-1);
return true;
//---------- Automata OpAsig----------
case 6 : c=SigCar(oAnaLex.getI());
oAnaLex.setI(oAnaLex.getI()+1);
if ((lenguaje="=").indexOf(c)>=0) _edoAct=7; else
if ((lenguaje="+").indexOf(c)>=0) _edoAct=8; else
if ((lenguaje="-").indexOf(c)>=0) _edoAct=8; else
if ((lenguaje="*").indexOf(c)>=0) _edoAct=8; else
if ((lenguaje="/").indexOf(c)>=0) _edoAct=8; else
if ((lenguaje="^").indexOf(c)>=0) _edoAct=8; else
{ oAnaLex.setI(oAnaLex.getIniToken());
return false; }
break;
case 7 : return true;
case 8 : c=SigCar(oAnaLex.getI());
oAnaLex.setI(oAnaLex.getI()+1);
if ((lenguaje="=").indexOf(c)>=0) _edoAct=7; else
{ oAnaLex.setI(oAnaLex.getIniToken());
return false; }
break;
//---------- Automata oparit----------
case 9 : c=SigCar(oAnaLex.getI());
oAnaLex.setI(oAnaLex.getI()+1);
if ((lenguaje="+").indexOf(c)>=0) _edoAct=10; else
if ((lenguaje="-").indexOf(c)>=0) _edoAct=10; else
if ((lenguaje="*").indexOf(c)>=0) _edoAct=10; else
if ((lenguaje="/").indexOf(c)>=0) _edoAct=10; else
if ((lenguaje="^").indexOf(c)>=0) _edoAct=10; else
{ oAnaLex.setI(oAnaLex.getIniToken());
return false; }
break;
case 10 : return true;
//---------- Automata Num
case 11 : c=SigCar(oAnaLex.getI());
oAnaLex.setI(oAnaLex.getI()+1);
if ((lenguaje="0123456789").indexOf(c)>=0) _edoAct=12;
else
{ oAnaLex.setI(oAnaLex.getIniToken());
return false; }
break;
case 12 : c=SigCar(oAnaLex.getI());
```

```
            oAnaLex.setI(oAnaLex.getI()+1);
            if ((lenguaje="0123456789").indexOf(c)>=0) _edoAct=12;
            else  if ((lenguaje="*+-/^.0123456789="
                              +"ABCDEFGHIJKLMNOPQRSTUVWXYZ()"
                              +"abcdefghijklmnopqrstuvwxyz;\n\t\r\f").indexOf(c)>=0)
_edoAct=13; else
            if ((lenguaje=".").indexOf(c)>=0) _edoAct=14; else
            { oAnaLex.setI(oAnaLex.getIniToken());
            return false; }
            break;
            case 13 : oAnaLex.setI(oAnaLex.getI()-1);
            return true;
            case 14 : c=SigCar(oAnaLex.getI());
            oAnaLex.setI(oAnaLex.getI()+1);
            if ((lenguaje="0123456789").indexOf(c)>=0) _edoAct=15;
            else
            {
               oAnaLex.setI(oAnaLex.getIniToken());
               return false;
            }
            break;
            case 15 : c=SigCar(oAnaLex.getI());
            oAnaLex.setI(oAnaLex.getI()+1);
            if ((lenguaje="0123456789").indexOf(c)>=0) _edoAct=15;
            else if ((lenguaje="*+-/^.()="
                              +"0123456789"
                              +"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                              +"abcdefghijklmnopqrstuvwxyz;\n\t\r\f").indexOf(c)>=0)
_edoAct=13;
            else if ((lenguaje=".").indexOf(c)>=0) _edoAct=13;
            else
            { oAnaLex.setI(oAnaLex.getIniToken());
            return false; }
            break;
            //-------------- Automata sep--------------
            case 16 : c=SigCar(oAnaLex.getI());
            oAnaLex.setI(oAnaLex.getI()+1);
            if ((lenguaje="(").indexOf(c)>=0) _edoAct=17;
            else
            if ((lenguaje=")").indexOf(c)>=0) _edoAct=17;
            else
            { oAnaLex.setI(oAnaLex.getIniToken());
            return false; }
            break;
            case 17 : return true;
            //-------------- Automata termInst--------------
```

```
            case 18 : c=SigCar(oAnaLex.getI());
            oAnaLex.setI(oAnaLex.getI()+1);
            if ((lenguaje=";").indexOf(c)>=0) _edoAct=19; else
            { oAnaLex.setI(oAnaLex.getIniToken());
            return false; }
            break;
            case 19 : return true;
            }
            switch (_edoAct)
            {
            case 2 : // Autómata delim
            case 5 : // Autómata Id
            case 13 : // Autómata num
            oAnaLex.setI(oAnaLex.getI()-1);
            return true;
            }
         return false;
      }

}
```

APÉNDICE C

Clases del análisis sintáctico

class Sintactico

```
{
    public final int NODIS=5000;
    private Pila _pila;
  private String[] _vts={ "", "id", "=", ";", "-", "+", "*", "/", "^", "num", "(", ")", "tan",
"cos", "sin", "$" };
    private String[] _vns={ "", "A", "A'", "E", "E'", "E'", "T", "T'", "F", "G" };
    private int[][] _prod={{1,3,-1,-2,2,0},          // A->id = A'
                           {2,2,3,-3,0,0},           // A'->E ;
                           {2,3,-4,3,-3,0},          // A'->- E ;
                           {3,2,6,4,0,0},           // E->T E'
                           {4,2,-4,5,0,0},           // E'->- E''
                           {4,2,-5,5,0,0},           // E'->+ E''
                           {4,0,0,0,0,0},           // E'->£
                           {5,2,6,4,0,0},           // E''->T E'
                           {5,2,9,4,0,0},           // E''->G E'
                           {6,2,9,7,0,0},           // T->G T'
                           {6,2,8,7,0,0},           // T->F T'
                           {7,2,-8,6,0,0},           // T'->^ T
                           {7,2,-7,6,0,0},           // T'->/ T
                           {7,2,-6,6,0,0},           // T'->* T
                           {7,0,0,0,0,0},           // T'->£
                           {8,1,-1,0,0,0},           // F->id
                           {8,1,-9,0,0,0},           // F->num
                           {8,3,-10,3,-11,0},          // F->( E )
                           {9,4,-12,-10,3,-11},          // G->tan ( E )
                           {9,4,-13,-10,3,-11},          // G->cos ( E )
                           {9,4,-14,-10,3,-11}          // G->sin ( E )
                          };
     private int[][] _m={{1,1,0}, {2,12,1}, {2,13,1}, {2,14,1}, {2,1,1}, {2,9,1},
{2,10,1}, {2,4,2}, {3,12,3}, {3,13,3}, {3,14,3}, {3,1,3}, {3,9,3}, {3,10,3}, {4,4,4},
4,5,5}, {4,3,6}, {4,11,6}, {5,12,7},                {5,13,7},                {5,14,7},
{5,1,7},                {5,9,7},                {5,10,7},                {6,12,9},
{6,13,9},                {6,14,9},                {6,1,10},                {6,9,10},
{6,10,10},            {7,8,11},                {7,7,12},                {7,6,13},
{7,4,14},            {7,5,14},                {7,3,14},                {7,11,14},
{8,1,15},                {8,9,16},                {8,10,17},                {9,12,18},
{9,13,19},                {9,14,20}
```

```
                };

private int _noVts;
private int _noVns;
private int _noProd;
private int _noEnt;
private int[] _di;
private int _noDis;

// Metodos

public Sintactico() // Constructor ----------------------
{
    _pila=new Pila();
    _noVts = _vts.length;
    _noVns = _vns.length;
   _noProd = 21;
    _noEnt = 43;
    _di=new int[NODIS];
    _noDis=0;


}  // Fin del Constructor ---------------------------------------

public void Inicia() // Constructor ----------------------
{
    _pila.Inicia();
    _noDis=0;
}
public int Analiza(Lexico oAnaLex)
{
    SimbGram x=new SimbGram("");
    String a;
    int noProd;
                _pila.Inicia();
        _pila.Push(new SimbGram("$"));
        _pila.Push(new SimbGram(_vns[1]));
        oAnaLex.Anade("$", "$");
        int ae = 0;
     do {
        x = _pila.Tope();
        a = oAnaLex.Token(ae);
        if (EsTerminal(x.Elem()))
            if (x.Elem().equals(a))
            {
                _pila.Pop();
```

```
                ae++;
            }
            else
                return 1;
        else
        {
            if ((noProd = BusqProd(x.Elem(), a)) >= 0)
            {
                _pila.Pop();
                MeterYes(noProd);
                _di[_noDis++] = noProd;
            }
            else
                return 2;
        }
             }while(!x.Elem().equals("$"));
       return 0;
 }
 public boolean EsTerminal(String x)
 {
    for(int i=1;i<_noVts;i++)
        if(_vts[i].equals(x))
           return true;
    return false;
 }
 public int BusqProd(String x, String a)
 {
    int indiceX=0;
    for(int i=1;i<_noVns;i++)
        if (_vns[i].equals(x))
        {
            indiceX = i;
            break;
        }
    int indiceA=0;
    for(int i=1;i<_noVts;i++)
        if (_vts[i].equals(a))
        {
            indiceA = i;
            break;
        }
    for (int i = 0; i < _noEnt; i++)
         if (indiceX == _m[i][0] && indiceA == _m[i][1])
             return _m[i][2];
    return -1;
 }
```

```
public void MeterYes(int noProd)
{
    int noYes = _prod[noProd] [1];
    for (int i = 1; i <= noYes; i++)
        if (_prod[noProd] [noYes + 2 - i] < 0)
            _pila.Push(new SimbGram(_vts[-_prod[noProd] [noYes + 2 - i]]));
        else
            _pila.Push(new SimbGram(_vns[_prod[noProd][noYes + 2 - i]]));
}

public String[] Vts()
{
    return _vts;
}

public String[] Vns()
{
   return _vns;
}

public int[][] M()
{
    return _m;
}

public int NoDis()
{
    return _noDis;
}

public int[] Di()
{
   return _di;
}

public int IndiceVn(String vn)
{
   for (int i = 1; i < _noVns; i++)
       if (_vts[i].equals(vn))
           return i;
   return 0;
}

public int IndiceVt(String vt)
{
```

```
        for (int i = 1; i < _noVts; i++)
            if (_vts[i].equals(vt))
               return i;
        return 0;
    }

    public int NoProd()
    {
        return _noProd;
    }

} // fin de la clase sintáctico

class SimbGram
   {
     String _elem;

        public SimbGram(String sValor)
        {
            _elem = sValor;
        }

        public String Elem()
        {
            return _elem;
        }

   }  // Fin de la clase SimbGram

public class Pila
{
        final int MAX = 5000;
        SimbGram[] _elems;
        int _tope;
        int a=0;
        public Pila()
        {
            _elems=new SimbGram[MAX];
            for (int i = 0; i < _elems.length; i++)
                _elems[i] = new SimbGram("");
            _tope = 0;
        }
        public boolean Empty()
        {
            return _tope==0? true : false;
        }
```

```
    public boolean Full()
    {
       return _tope==_elems.length? true : false;
    }
    public void Push(SimbGram oElem)
    {
       _elems[_tope++]=oElem;
    }
    public int Length()
    {
       return _tope;
    }
    public SimbGram Pop()
    {
       return _elems[--_tope];
    }

    public void Inicia()
    {
       _tope = 0;
    }

    public SimbGram Tope()
    {
       return _elems[_tope - 1];
    }

}  // Fin de la clase Pila-
```

APÉNDICE D

Programa principal de Matlab

```
function x = grafica(ecuacion,ti,tf,x,tol,odesel,x1,x2,x3,tipoGrafica)
t=[ti tf];
graficar=inline(ecuacion,'t','x');     %genera el ode file con las ecuaciones
options=odeset('RelTol',tol);
%el siguiente switch genera la grafica dependiendo del ode seleccionado
switch odesel
   case {0}
      [t,x]=ode45(graficar,t,x,options);
   case {1}
      [t,x]=ode23(graficar,t,x,options);
   case {2}
      [t,x]=ode113(graficar,t,x,options);
   case {3}
      [t,x]=ode23s(graficar,t,x,options);
   case {4}
      [t,x]=ode23t(graficar,t,x,options);
end

switch tipoGrafica
   case {1}
      figure
      plot(x(:,x1),x(:,x2))
      title('Análisis de sistemas no lineales');
   case {2}
      figure
      plot3(x(:,x1),x(:,x2),x(:,x3))
      title('Análisis de sistemas no lineales');
end
```