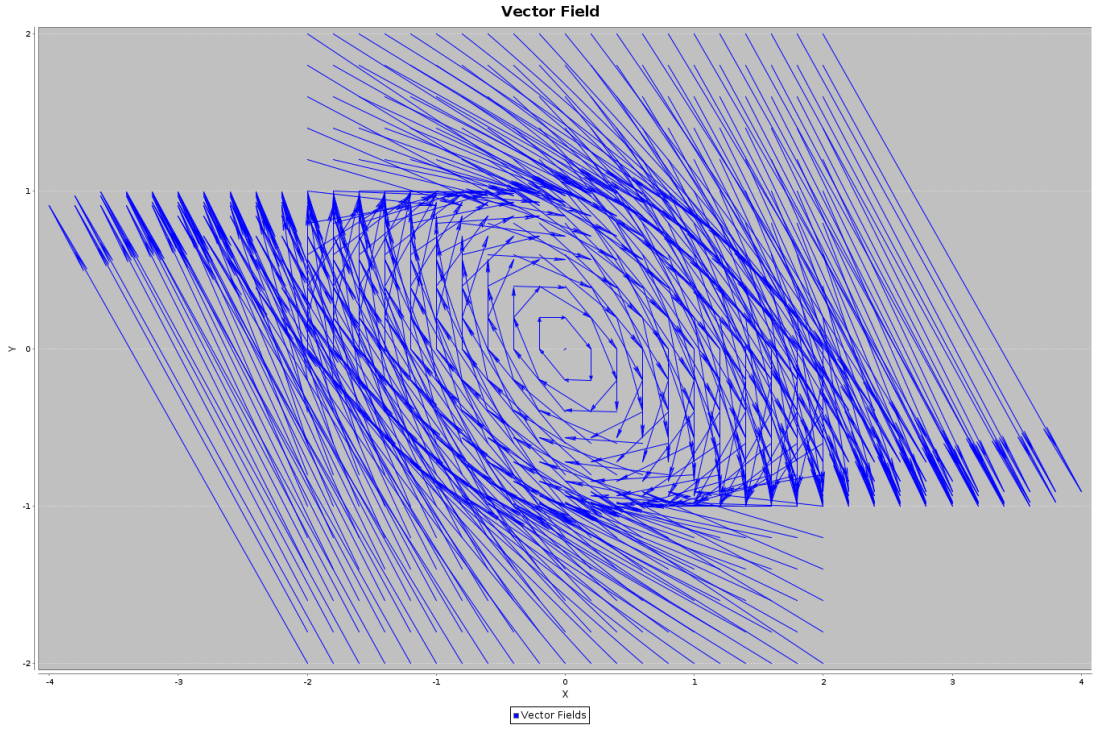
DYNAMICAL SYSTEMS SIMULATOR

DS SIMULATOR

USER MANUAL



ANGÉLICA MARÍA ATEHORTÚA LABRADOR[1]

EDISON IVANÚ SABOGAL PÉREZ[2]

[1] Systems engineer, Universidad de los Llanos. Research Group: Dynamical Systems. Young researcher of Colciencias 2010, amatehortual@gmail.com

[2] Mathematician, Universidad Nacional. Teacher from Universidad de los Llanos and director of Dynamical Systems research group, eisabogal@gmail.com

DYNAMICAL SYSTEMS SIMULATOR
DS SIMULATOR

USER MANUAL

Angélica María Atehortúa Labrador
Edison Ivanú Sabogal Pérez

August 8, 2011

INTRODUCTION

DS SIMULATOR is an open source tool developed to analyze and simulate discrete, continuous and stochastic dynamical systems in real variable. DS SIMULATOR is designed for students, teachers and researchers whose field of activity is broad, presented with a graphical user interface very simple and easy to use. DS Simulator is presented as a tool multiplatform that can be used on any operating system. You can generate efficient programs and numerical solutions is also possible to export the results of numerical experiments to Octave, gnuplot and Matlab.

The simulation of dynamical systems is useful to understand the theory of dynamical systems, in the case of students, and support research in the various applications of them, as in biology at population model; in physics, chemistry, economics, and engineering, among others. This software is intended for the simulation of discrete and continuous dynamical systems in real variable. Discrete systems, which are in terms of difference equations, continuous system, represented by ordinary differential equations, only two-dimensional systems and stochastic systems, which are written by stochastic differential equations in one and two dimensions.

Very few developed software for analysis and simulation of dynamical systems, some are complex to use and business license, which requires a value for purchase, and generally softwares are very specific to an area of dynamic systems, such as bifurcations. The main reason why the DS Simulator was developed to satisfy a specific academic need for teaching and research of the theory and applications of dynamical systems in the population of Unillanos and others interested in the area.

This software and documentation is also available in pdf format to the user through the site https://sites.google.com/site/angelicamariaatehortualabrador

Contents

Chapter 1

PREPARATION

1.1 System Requirements

DS Simulator can be run from any operating system with Java version 6. This is the download link for java: http://www.java.com/es/download/. The computer must have a pdf reader, that to see the user manual.

1.2 Installation

Simply run the DS Simulator with double click on it.

Chapter 2

GRAPHICAL USER INTERFACE (GUI)

DS Simulator starts with the application of graphical iteration of Discretes menu. Figure 2.1. The main window of each menu has 4 panels, Data entry, Graph, Status and Analysis, the latter is not present at all. in Data entry it introduce the data to be plotted and displayed in the Graph menu , Status displays the status of the current process to complete 100%, which displays the results in the Graph Panel, and Analysis introduces a data to be analyzed in relation to data of Data entry panel and display the results right there.

2.1 Menus

The program menu is composed of File, Discretes, Continuous, Stochastic and Help menus, corresponding to Figures 2.3a, 2.3b, 2.3c, 2.3d y 2.3e respectively. The figure 2.2 shows the main menu. The file menu can be run from any open window.

2.2 Graph panel

To plot were used jfreechart libraries [2]. Figure 2.4 shows the graph obtained by the simulation of dynamical system entered. To select a portion of the graph, this part is maximized, as shown in Figure 2.5. Clicking on the graph shows the menu and also shows a submenu of "Acercar", which is the same for menus "Alejar" and "Escala Automática", displayed in Figure 2.6. The properties of the graph are displayed in the Figure 2.7.

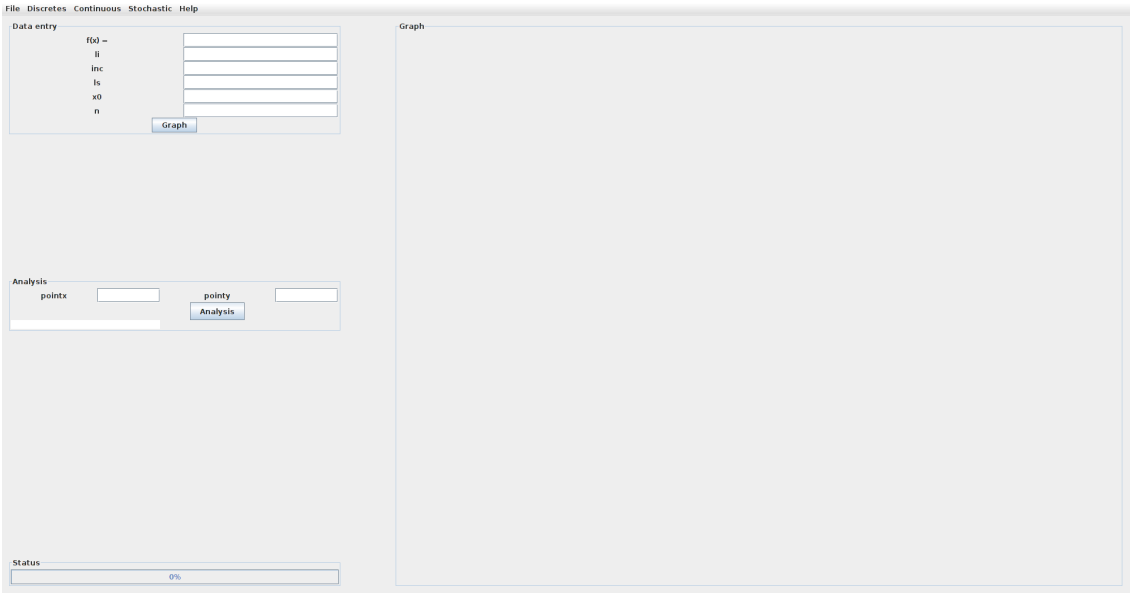


Figure 2.1: Main window

Figure 2.2: Program menu

2.3 Data entry panel

To evaluate the functions are used library jep [1]. The syntax for the input data is:

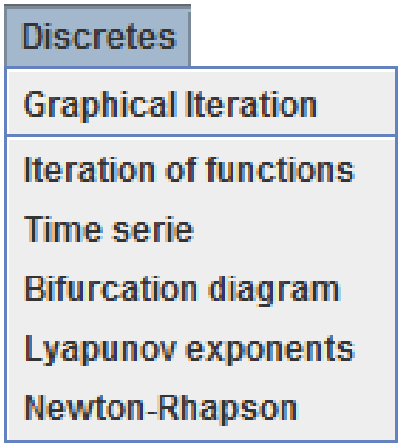
Basic Operations

Sum +
Subtraction -
Division /
Potentiation ^
multiplication *
Mathematical Functions

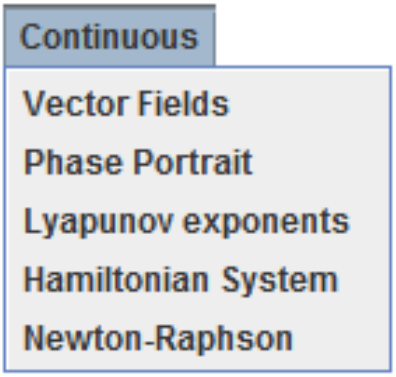
Functions

Support for implicit multiplication use of expression "3*x". The syntax for functions are displayed in tables 2.1, 2.2 and 2.3.

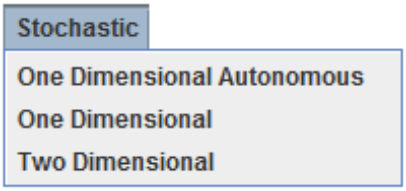
(a) File menu



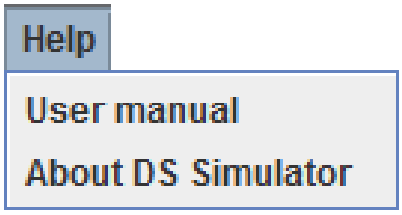
(b) Discrete menu



(c) Continuous menu



(d) Stochastic menu



(e) Help menu

Figure 2.3: Menus

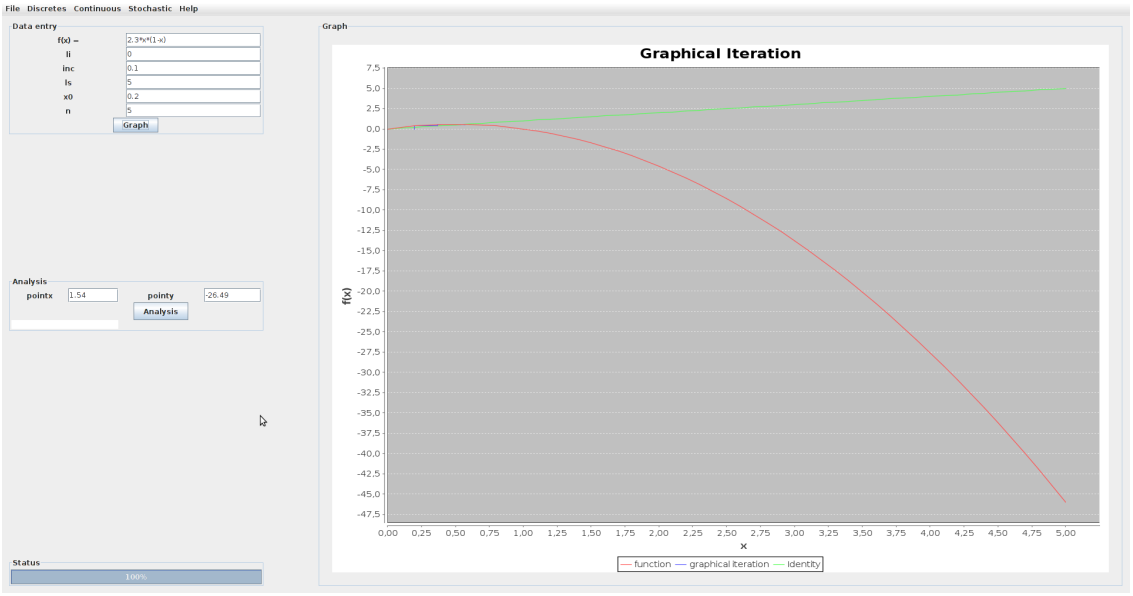


Figura 2.4: Graph panel

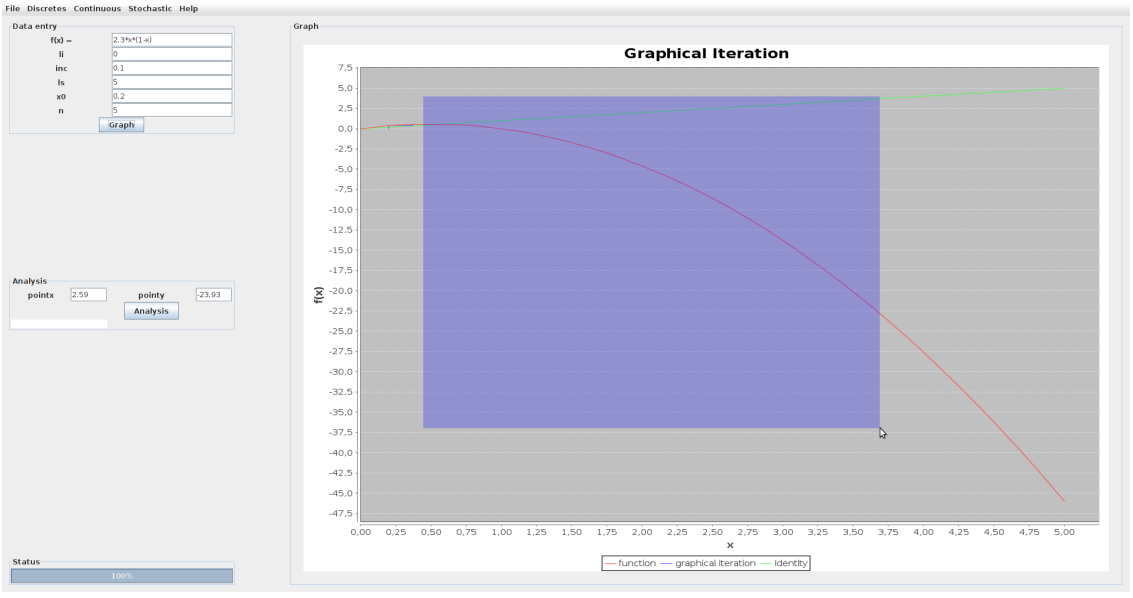


Figura 2.5: Graphic maximize

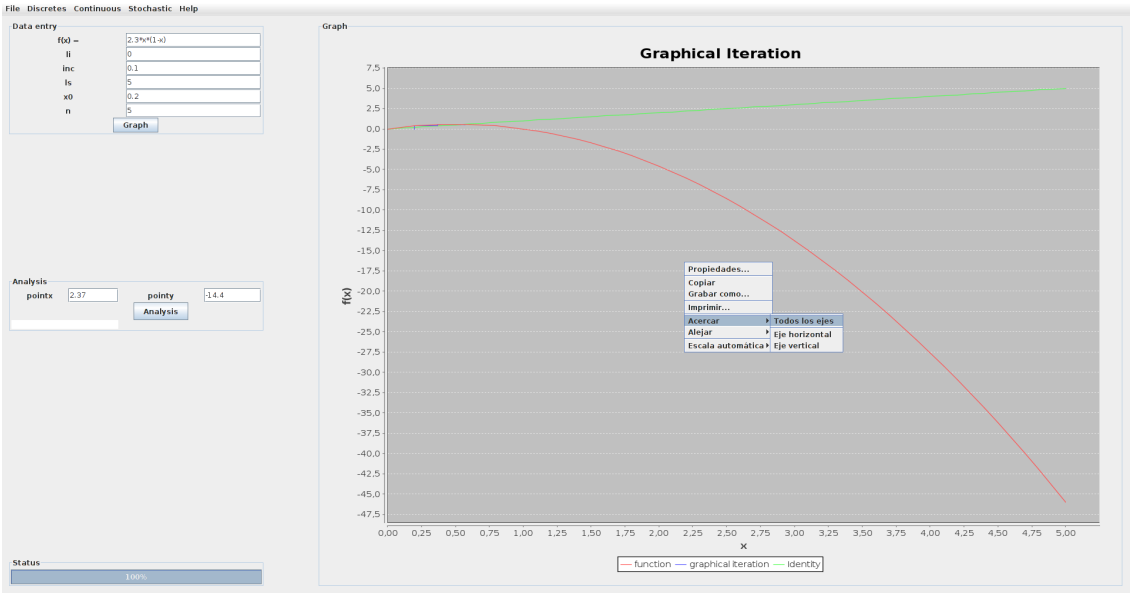


Figura 2.6: Graph menu

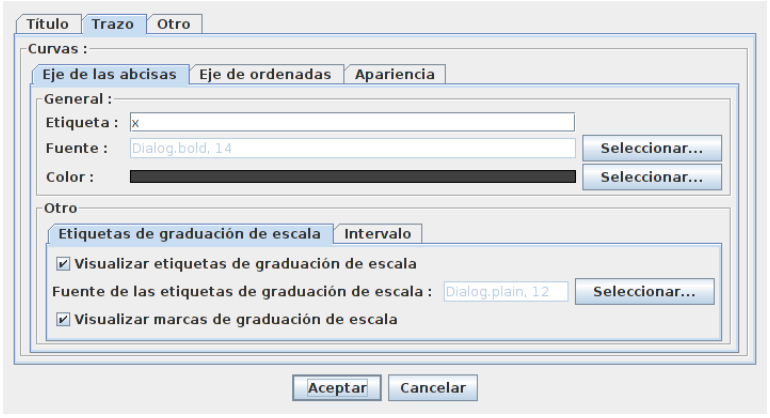


Figura 2.7: Graphic properties

Table 2.1: Trigonometric Functions

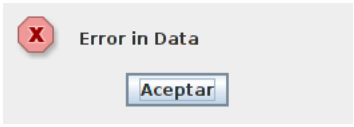
Description	**Function name**
Sine	sin(x)
Cosine	cos(x)
Tangent	tan(x)
Arc Sine2	asin(x)
Arc Cosine	acos(x)
Arc Tan	atan(x)
Arc Tan with 2 parameters	atan2(y, x)
Secant	sec(x)
Cosecant	cosec(x)
Co-tangent	cot(x)
Hyperbolic Sine	sinh(x)
Hyperbolic Cosine	cosh(x)
Hyperbolic Tangent	tanh(x)
Inverse Hyperbolic Sine	asinh(x)
Inverse Hyperbolic Cosine	acosh(x)
Inverse Hyperbolic Tangent	atanh(x)

Table 2.2: Log and Exponential Functions

Description	**Function name**
Natural Logarithm	ln(x)
Natural Logarithm base 10	log(x)
Logarithm base 2	lg(x)
Exponential	exp(x)
Power	pow(x)

Table 2.3: Miscellaneous Functions

Description	**Function name**
Absolute value / Magnitude	abs(x)
Random number (between 0 and 1)	rand()
Modulus	mod(x,y)
Square Root	sqrt(x)
Sum	sum(x,y)
Binomial coefficients	binom(n,i)
Signum (-1,0,1 depending on sign of argument)	signum(x)



(a) Error message

(b) Empty function

(c) First graph

Figure 2.8: Messages

2.4 Messages

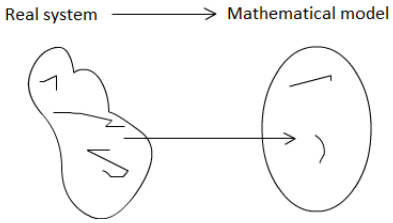
When an error occurs is displayed a window showing "Error in data", this is due to incorrect data or that have not entered any, as showed in figure 2.8a, and if when the function field is empty shows “Empty function”, figure 2.8b. The message in figure 2.8c indicates that you must first run the graph and then the action to take.

Chapter 3

DYNAMICAL SYSTEMS SIMULATOR "DS SIMULATOR"

In this software are presented discrete, continuous and stochastic dynamical systems using ordinary differential equations and stochastic equations.

A dynamical system is a mathematical model represented by a set of differential equations, difference equations or partial differential equations that describe how the state of a system changes over time. These models can be of different types such as linear or nonlinear, discrete or continuous, spatially homogeneous or heterogeneous, deterministic or stochastic, dynamical systems have different application areas such as mechanics, electronics and electricity, elementary particle physics, fluid , thermodynamics, chemistry, population dynamics, cell biology, biochemistry, genetics, epidemiology, physiology, economics, sociology and psychology.



$$\frac{d\mathbf{x}}{dt} = \mathbf{F}\left(x; \boldsymbol{\mu}, t\right)$$

Figura 3.1: Definition of a dinamycal System

Chapter 4

DISCRETE DYNAMICAL SYSTEMS

Are mathematical models of difference equations for processes that evolve in discrete time, that system changes occur at specific times rather than continuously. For example, they often measure growth of a population at specific times, and the end of a production cycle. The study of dynamical systems often involves discrete iteration process.

It work with real functions f : $\mathbb{R} \to \mathbb{R}$. As usual, assumed throughout that f is C^∞ , although there will be several special examples where this is not the case.

Let be f^n denote the nth iterate of f . That is, f^n is the n-fold composition of f with itself. Given $x_0 \in \mathbb{R}$, the orbit of x_0 is the sequence $x_0, x_1 = f(x_0), x_2 = f^2(x_0), ..., x_n = f^n(x_0), ...$

The point x_0 is called the seed of the orbit [3].

4.1 Graphical iteration

A useful way to visualize orbits of one-dimensional discrete dynamical systems is via *graphical iteration.* In this picture, it overlaps the curve $y = f(x)$ and the diagonal line $y = x$ on the same graph. it displayed the orbit of x_0 as follows: Begin at the point (x_0, x_0) on the diagonal and draw a vertical line to the graph of f , reaching the graph at $(x_0, f(x_0)) = (x_0, x_1)$. Then draw a horizontal line back to the diagonal, ending at (x_1, x_1). This procedure moves us from a point on the diagonal directly over the seed x_0 to a point directly over the next point on the orbit, x_1 . Then it continued from (x_1, x_1): First go vertically to the graph to the point (x_1, x_2), then horizontally back to the diagonal at (x_2, x_2). On the x-axis this moves us from x_1 to the next point on the orbit, x_2. Continuing, it produced a sequence of pairs of lines, each of which terminates on the diagonal at a point of the form (x_n, x_n) [3].

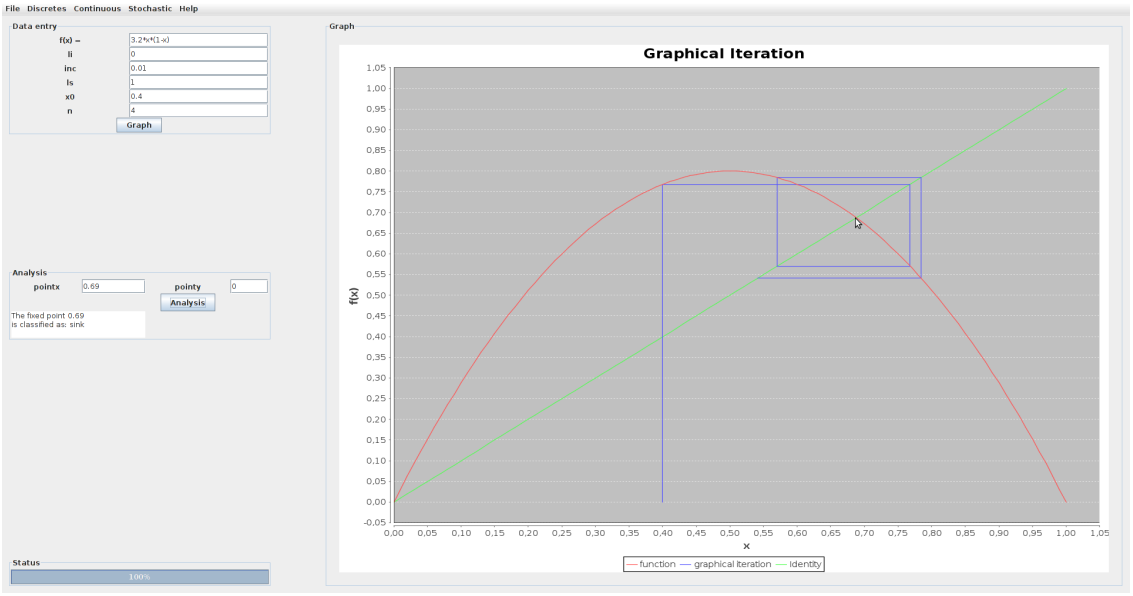


Figure 4.1: Graphical iteration of $3.2x(1-x)$ under orbite 0.4

Theorem 4.1.1 *Let I be $I=[a,b]$ a closed interval and a $f: I \to I$ continuous function. Then f has a fixed point in I.*

[4]
A fixed point x_0 is one that satisfies the equality $f(x) = x$.

Proposition 4.1.1 *Suppose f has a fixed point at x_0 . Then*

1. *x_0 is a sink if $|f(x_0)| < 1$;*
2. *x_0 is a source if $|f(x_0)| > 1$;*
3. *It get no information about the type of x_0 if $f(x0) = \pm 1$.*

[3]
The figure 4.1 shows the graph of $f(x) = 3.2x(1-x)$ describing the orbit 0.4 around 0.687 fixed point of the function.

4.1.1 Gui

Figure 4.1 is explained:

- **Data entry panel:**
  f(x): is the function.
  $I = [li, ls]$ is the interval function f on the which is observed its behavior.

inc: is the difference in the arithmetic progression which is the interval I. Take a very small increment value for that the graph more accurate.

x0: is the orbit.

n: is the number of iterations.

- **Graph panel:**

  Is where it looks the graph of the input data, can be selected a point on this graph and it displayed in the panel analysis.

- **Analysis panel:**

  Point x shows the point of the x axis that was selected from the graph panel, or also can be entered by the user. By analyzing shows the classification of fixed point according to the proposition 4.1.1.

4.1.2 File menu: Save

To save must first to graph. It is saved in a flat file, TXT format, where the first two columns represent data function, the next two columns represent the graphic iteration data and the latest columns are the identity function data. They are organized first x-axis data followed by the y-axis data.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where the first replaces the comma by point, and load file gnuplot.

All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done as follows in linux: perl -p -i -e 's/,/./g' graph.txt

Script file:

```
#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot
load 'graph.gnuplot'
EOF
```

Gnuplot file:

```
set term postscript eps enhanced
set output "GraphicalIteration.eps"
set size square
set xlabel "x"
set ylabel "y"
plot "<awk '{print $1, $2}' graph.txt" ti "Graphical Iteration" with lines, "<awk '{print $3, $4}' graph.txt" ti "Function" with lines, "<awk '{print $5, $5}' graph.txt" ti "Identity" with lines
```

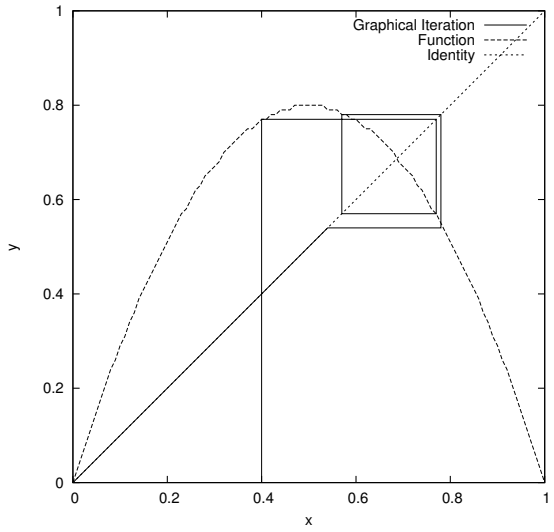


Figure 4.2: Graphical iteration using GNUPLOT

Once you save the file gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as showed in figure 4.2. The data are the same as those used in Figure 4.1.

In Matlab and Octave can plot the data from txt file, saving a file m format the following code:

```
clear all
close all
clc
X=load('graph.txt');
hold on
plot(X(:,1),X(:,2),'b')
plot(X(:,3),X(:,4),'g')
plot(X(:,5),X(:,5),'r')
xlabel('x'); ylabel('f(x)')
legend('function','graphical iteration','identity')
```

The figure 4.3 displays the graphs obtained in Matlab and Octave respectively. To save in Octave is used:

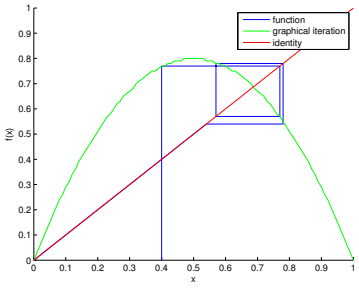
```
octave:5> print -color filename.eps
```

4.1.3 File menu: Export to Matlab

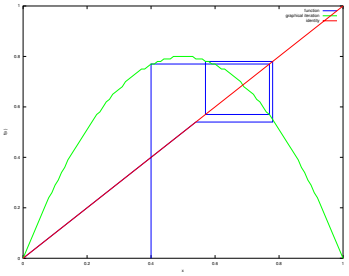
To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.

This is the code that is generated with the data showed in figure 4.1.

```
% DS Simulator
```



(a) Graphic iteration in MATLAB loading data



(b) Graphic iteration in Octave loading data

Figure 4.3: Graphical iteration loading data in MATLAB and Octave

```
% Graphical Iteration
clear all
close all
clc
li=0.0;
inc=0.01;
ls=1.0;
Xo=0.4;
n=4.0;
xx(1)=0.4;
yy(1)=0;
i=1;
for j=1:n
xx(i+1)=xx(i);
x=xx(i);
yy(i+1)=3.2*x*(1-x);
xx(i+2)=yy(i+1);
yy(i+2)=xx(i+2);
```

```
i=i+2; end
ki=li:inc:ls;
for i=1:length(ki)
x=ki(i);
f(i)=3.2*x*(1-x);
end
hold on
plot(xx,yy,'r')
plot(ki,f,'b')
plot(ki,ki,'g')
grid on
xlabel('x')
ylabel('y')
legend('Graphic Iteration','Function','Identity')
```

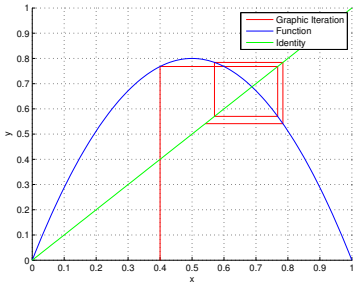
Figure 4.4a shows the result.

4.1.4 File menu: Export to Octave

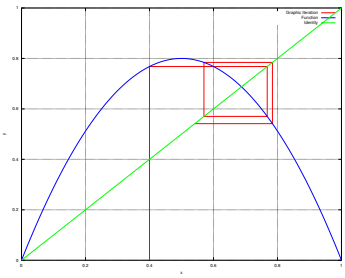
To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

This is the code that is generated with the data showed in figure 4.1.

```
% DS Simulator
% Graphical Iteration
clear all
close all
clc
li=0.0;
inc=0.01;
ls=1.0;
Xo=0.4;
n=4.0;
xx(1)=0.4;
yy(1)=0;
i=1;
for j=1:n
xx(i+1)=xx(i);
x=xx(i);
yy(i+1)=3.2*x*(1-x);
xx(i+2)=yy(i+1);
yy(i+2)=xx(i+2);
i=i+2;
endfor
ki=li:inc:ls;
for i=1:length(ki)
x=ki(i);
f(i)=3.2*x*(1-x);
```



(a) Graphic iteration exporting code to MATLAB



(b) Graphic iteration exporting code to Octave

Figure 4.4: Code exported to Matlab and Octave from graphic iteration

```
endfor
hold on
plot(xx,yy,'r')
plot(ki,f,'b')
plot(ki,ki,'g')
grid on
xlabel('x')
ylabel('y')
legend('Graphic Iteration','Function','Identity')
```

Figure 4.4b shows the result.

4.2 Iteration of functions

To find the periodic points is used the iteration of functions, for example, the second iterated are points of period two, and so on. The point x is a periodic point of f with period k if $f^k(x) = x$. In other words, x is a periodic point of f if x is a fixed point of f^k. Also, x has prime period k_0 if x return to x just

after k_0 iterations of f [4].

To simplify the notation of the iteration, it introduced the expression f^n to denote the nth iteration of f. For example, if $f(x) = x^4$ then $f^2(x) = f(f(x)) = (x^4)^4 = x^{16}$, and similarly $f^3(x) = 64$.

Definition 4.2.1 *the point x is a periodic point of f with period k if $f^k(x) = x$. In other words, x is a periodic point of f with period k if x is a fixed point of f^k. The point x has prime period k_0 if $f^{x_0}(x) = x$ and $f^n(x) \neq x$ whenever $0 < n < k_0$. This is, x has prime period k_0 if x return to x just after k_0 iterations of f.*

The set of all iterations of a point x is called the orbit of x, and if x is a periodic point, then it and iterated is called a periodic orbit or periodic cycle.

Definition 4.2.2 *the point x is a fixed finally point of the function f, if exist N, so that $f^{n+1}(x) = f^n(x)$ whenever $n \geq N$. The point x is periodic finally point with period k, if exist N so that $fn + k(x) = f^n(x)$ whenever $n \geq N$.*

Definition 4.2.3 *Let be f a function and p a periodic point of f with prime period k. Then x is asymptotic to p if sequence $x, f^k(x), f^{2k}(x), f^{3k}, ...$ converges to p, in other words, $\lim_{n\to\infty} f^{nk}(x) = p$. The stable set of p denoted by $W^s(p)$, consists of all points which are asymptotic to p. If sequence $|x|, |f(x)|, |f^2(x)|, |f^3(x)|, ...$ grows without limits, then x is asymptotic to ∞. The stable set of ∞, denoted by $W^s(\infty)$, consists of all points which are asymptotic to ∞.*

The figure 4.5 shows the graph of second iterate of $f(x) = 3.2x(1 - x)$ describing the orbit 0.4 around 0.8 fixed point of the function.

4.2.1 Gui

Figure 4.5 is explained:

- **Data entry panel:**

  f(x): is the function.

  iterated: is the number of times it will iterate the function *f(x)*

  $I = [li, ls]$ is the interval function f on the which is observed its behavior.

  inc: is the difference in the arithmetic progression which is the interval I. Take a very small increment value for the graph more accurate.

  x0: is the orbit.

  n: is the number of iterations.

- **Graph panel:**

  Is where it looks the graph of the input data, can be selected a point on this graph and it displayed in the panel analysis.

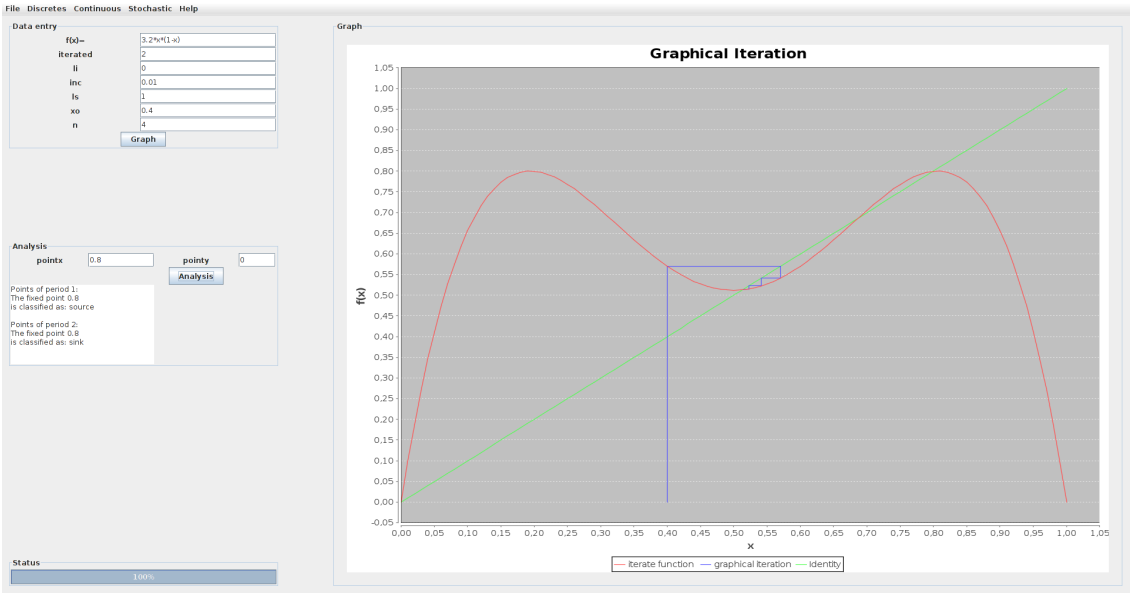


Figure 4.5: Second iterate of $f(x) = 3.2x(1-x)$

- **Analysis panel:**

  Point x shows the point of the x axis that was selected from the graph panel, or also can be entered by the user. By analyzing shows the classification of periodic points according to the proposition 4.1.1.

4.2.2 File menu: Save

To save must first to graph. It is saved in a flat file, TXT format, where the first two columns represent data function, the next two columns represent the graphic iteration data and the latest columns are the identity function data. They are organized first x-axis data followed by the y-axis data.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where the first replaces the comma by point, and load file gnuplot.

All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done as follows in linux: perl -p -i -e 's/,/./g' graph.txt

Script file:
#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot

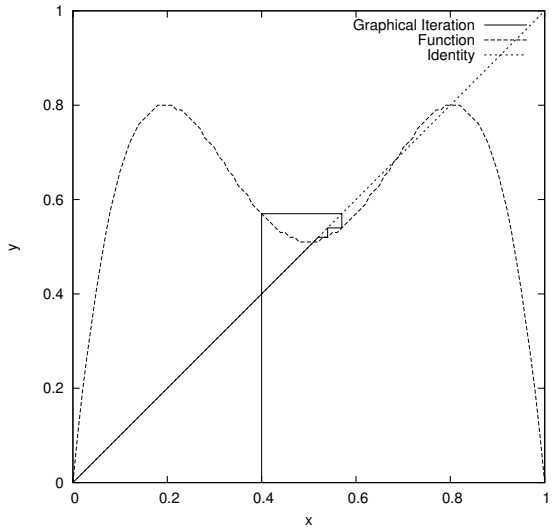


Figure 4.6: Iteration of functions using GNUPLOT

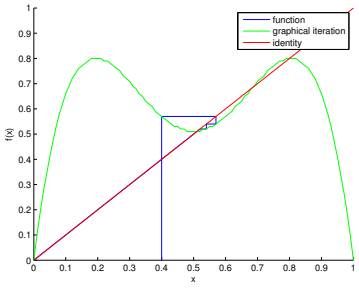
load 'graph.gnuplot'
EOF

Gnuplot file:
set term postscript eps enhanced
set output "IterationOfFunctions.eps"
set size square
set xlabel "x"
set ylabel "y"
plot "<awk '{print $1, $2}' graph.txt" ti "Graphical Iteration" with lines, "<awk '{print $3, $4}' graph.txt" ti "Function" with lines, "<awk '{print $5, $5}' graph.txt" ti "Identity" with lines

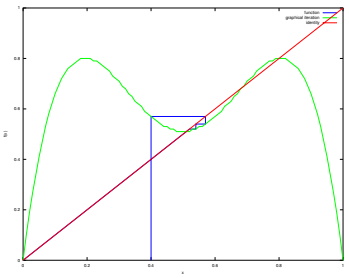
Once you save the file gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as showed in figure 4.6. The data are the same as those used in figure 4.5.

In Matlab and Octave can plot the data from txt file, saving a file m format the following code:

```
clear all
close all
clc
X=load('graph.txt');
hold on
plot(X(:,1),X(:,2),'b')
plot(X(:,3),X(:,4),'g')
```



(a) Iteration of functions in MATLAB loading data



(b) Iteration of functions in Octave loading data

Figure 4.7: Iteration of functions in MATLAB and Octave loading data

```
plot(X(:,5),X(:,5),'r')
xlabel('x'); ylabel('f(x)')
legend('function','graphical iteration','identity')
```

The figure 4.7 displays the graphs obtained in Matlab and Mctave respectively. To save in Octave is used:

octave:5> print -color filename.eps

4.2.3 File menu: Export to Matlab

To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.

This is the code that is generated with the data showed in figure 4.5.

```
% DS Simulator
% Iteration of functions
clear all
close all
clc
li=0.0;
```

```
inc=0.01;
ls=1.0;
Xo=0.4;
n=4.0;
xx(1)=0.4;
yy(1)=0; i=1;
for j=1:n
xx(i+1)=xx(i);
x=xx(i);
yy(i+1)=3.2*(3.2*x*(1-x))*(1-(3.2*x*(1-x)));
xx(i+2)=yy(i+1);
yy(i+2)=xx(i+2);
i=i+2;
end
ki=li:inc:ls;
for i=1:length(ki)
x=ki(i);
f(i)=3.2*(3.2*x*(1-x))*(1-(3.2*x*(1-x)));
end
hold on
plot(xx,yy,'r')
plot(ki,f,'b')
plot(ki,ki,'g')
grid on
xlabel('x')
ylabel('y')
legend('Graphic Iteration','Function','Identity')
```

Figure 4.8a shows the result.

4.2.4 File menu: Export to Octave

To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

This is the code that is generated with the data showed in figure 4.5.

```
% DS Simulator
% Iteration of functions
clear all
close all
clc
li=0.0;
inc=0.01;
ls=1.0;
Xo=0.4;
n=4.0;
xx(1)=0.4;
yy(1)=0;
```

```
i=1;
for j=1:n
xx(i+1)=xx(i);
x=xx(i);
yy(i+1)=3.2*(3.2*x*(1-x))*(1-(3.2*x*(1-x)));
xx(i+2)=yy(i+1);
yy(i+2)=xx(i+2);
i=i+2;
endfor
ki=li:inc:ls;
for i=1:length(ki)
x=ki(i);
f(i)=3.2*(3.2*x*(1-x))*(1-(3.2*x*(1-x)));
endfor
hold on
plot(xx,yy,'r')
plot(ki,f,'b')
plot(ki,ki,'g')
grid on
xlabel('x')
ylabel('y')
legend('Graphic Iteration','Function','Identity')
```

Figure 4.8b shows the result.

4.3 Time serie

It is a convenient method to describe the orbits geometrically. In these diagrams it drawn the iteration count on the horizontal axis and the numerical values of the orbit on the vertical axis. To get a better view, it can to draw straight lines connecting successive points of time series [5]. Figure 4.9 displays the time series $f(x) = 3.2x(1-x)$ and a periodic behavior is observed.

4.3.1 Gui

Figure 4.9 is explained:

- **Data entry panel:**

  f(x): is the function.

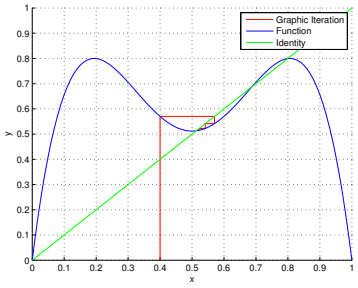
  initial point: is the initial value of the variable x, the default is 0.

  x0: is the orbit.

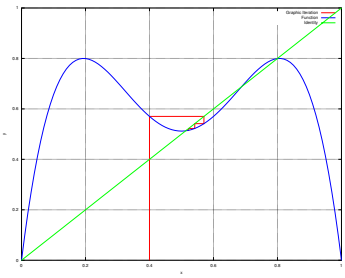
  n: is the number of iterations.

- **Graph panel:**

  Is where it looks the graph of the input data.



(a) Iteration of functions exporting code to MATLAB



(b) Iteration of functions exporting code to Octave

Figure 4.8: Code exported to Matlab and Octave from iteration of functions

4.3.2 File menu: Save

To save must first to graph. It is saved in a flat file, TXT format, where the first column represent the iterations and second column represent the time serie.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where the first replaces the comma by point, and load file gnuplot.

All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done as follows in linux: perl -p -i -e 's/,/./g' graph.txt

Script file:

```
#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot
load 'graph.gnuplot'
EOF
```

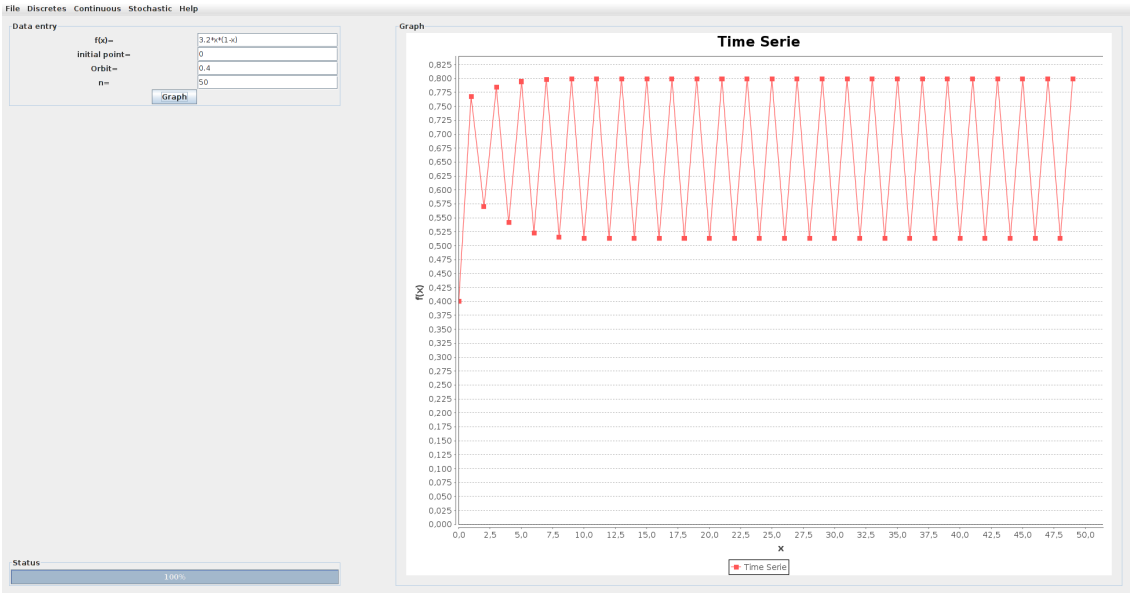


Figure 4.9: Time serie of $f(x) = 3.2x(1-x)$ under orbite 0.4

Gnuplot file:

```
set term postscript eps enhanced
set output "TimeSerie.eps"
set size square
set xlabel "n"
set ylabel "x"
plot "graph.txt" with lines
```

Once you save the file gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as showed in figure 4.10. The data are the same as those used in figure 4.9.

In Matlab and Mctave can plot the data from txt file, saving a file m format the following code:

```
clear all
close all
clc
X=load('graph.txt');
hold on
plot(X(:,1),X(:,2),'b-o')
xlabel('n')
ylabel('x(n)')
```

The figure 4.11 displays the graphs obtained in Matlab and Octave respectively. To save in Octave is used:

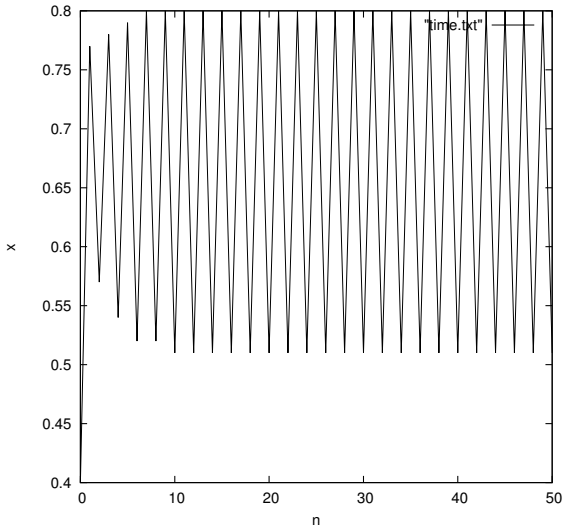


Figure 4.10: Time serie using GNUPLOT

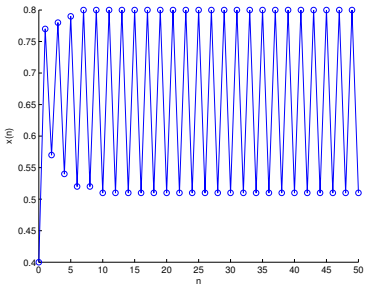
octave:5> print -color filename.eps

4.3.3 File menu: Export to Matlab

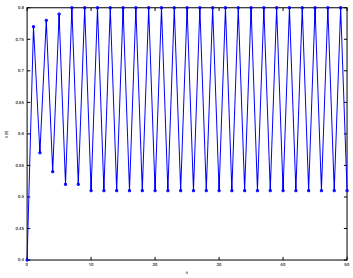
To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.

This is the code that is generated with the data showed in figure 4.9.

```
% DS Simulator
% Time Serie
clear all
close all
clc
nn(1)=0.0;
xx(1)=0.4;
n=50.0;
for i=1:n
x=xx(i);
nn(i+1)=nn(i)+1;
xx(i+1)=3.2*x*(1-x);
end
hold on
plot(nn,xx,'b-o')
grid on
xlabel('n')
ylabel('xn')
```



(a) Time serie in MATLAB loading data



(b) Time serie in Octave loading data

Figure 4.11: Time serie in MATLAB and Octave loading data

```
title('Time serie')
```

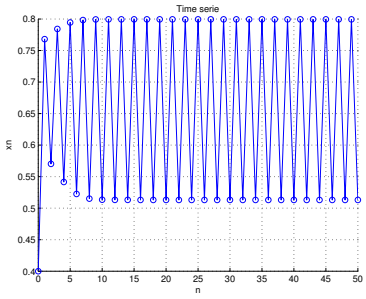
Figure 4.12a shows the result.

4.3.4 File menu: Export to Octave

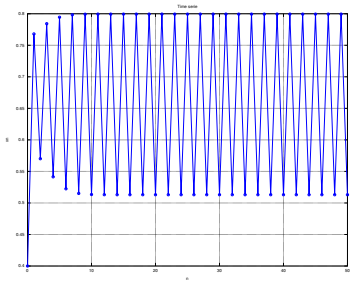
To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

This is the code that is generated with the data showed in figure 4.9.

```
% DS Simulator
% Time serie
clear all
close all
clc
nn(1)=0.0;
xx(1)=0.4;
n=50.0;
for i=1:n
x=xx(i);
nn(i+1)=nn(i)+1;
xx(i+1)=3.2*x*(1-x);
```



(a) Time serie exporting code to MATLAB



(b) Time serie exporting code to Octave

Figure 4.12: Code exported to Matlab and Octave from Time serie

```
endfor
hold on
plot(nn,xx,'b-o')
grid on
xlabel('n')
ylabel('xn')
title('Time serie')
```

Figure 4.12b shows the result.

4.4 Bifurcation diagram

Discrete dynamical systems may change in the structure of orbits when parameters vary. Such modifications may include the birth or death of fixed points and cycles or changes in the type of these orbits. Such changes are know as bifurcations.

Theorem 4.4.1 *(The Bifurcation Criterion) Let f_r be a family of functions depending smoothly on the parameter r. Suppose that $f_{r_0}(x_0) = x_0$ and $f_{r_0}(x_0) = 1$. Then there are intervals I about x_0 and J about r_0 and a smooth function p*

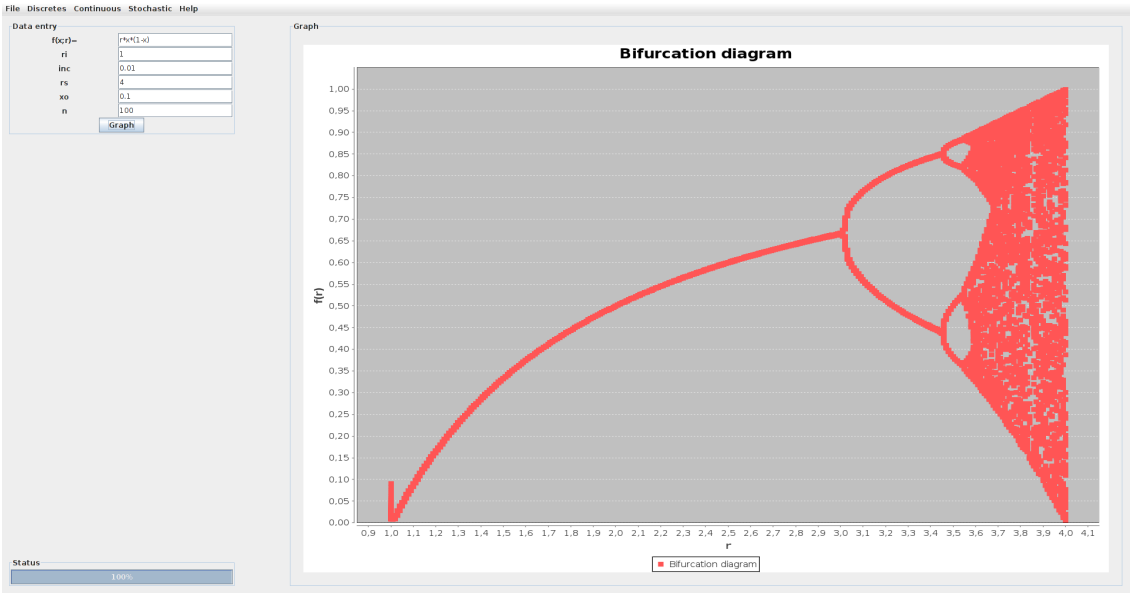


Figure 4.13: Bifurcation diagram of $rx(1-x)$

: $J \to I$ such that $p(r_0) = x_0$ and $f_r(p(r)) = p(r)$. Moreover, f_r has no other fixed points in I [3].

The bifurcation diagram is an extremely useful to understand the qualitative behavior of solutions. On the x axis is plotted the parameters on the axis r and the corresponding values taken by the fixed points in each parameter r. In the figure 4.13 is showed bifurcation diagram of $f_r(x) = rx(1-x)$.

4.4.1 Gui

Figure 4.13 is explained:

- **Data entry panel:**

  f(x;r): is the function with parameter r.

  $I = [ri, rs]$ is the interval function f on the which is observed its behavior.

  inc: is the difference in the arithmetic progression which is the interval I. Take a very small increment value for the graph more accurate.

  x0: is the initial value of the variable x.

  n: is the number of iterations.

- **Graph panel:**

  Is where it looks the graph of the input data.

4.4.2 File menu: Save

To save must first to graph. It is saved in a flat file, TXT format, where the first column represent the parameter and the other columns represent the fixed points.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where the first replaces the comma by point, and load file gnuplot.

All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done as follows in linux: perl -p -i -e 's/,/./g' graph.txt

Script file:

#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot
load 'graph.gnuplot'
EOF

Gnuplot file:

set term postscript eps enhanced
set output "Bifurcation.eps"
set size square
set xlabel "r"
set ylabel "fr(x)"
plot "<awk '{for(i=2; i<=NF;i++){print $1, $i} }' graph.txt" ti "Bifurcation"

Once you save the file gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as shown in Figure 4.14. The data are the same as those used in figure 4.13.

In Matlab it can plot the data from txt file, saving a file m format the following code:

```
clear all
close all
clc
X=load('graph.txt');
[m,n]=size(X);
for i=2:n
hold on
plot(X(:,1),X(:,i))
end
xlabel('r')
```

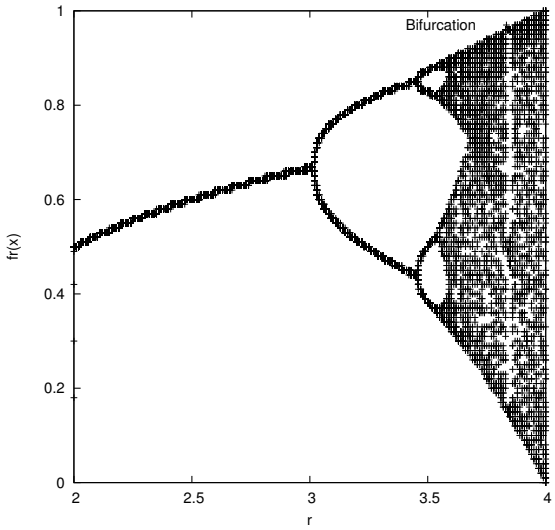


Figure 4.14: Bifurcation diagram using GNUPLOT

```
ylabel('f(r;x)')
```

And in Octave:

```
clear all
close all
clc
X=load('graph.txt');
[m,n]=size(X);
for i=2:n
hold on
plot(X(:,1),X(:,i))
endfor
xlabel('r')
ylabel('f(r;x)')
```

The figure 4.15 displays the graphs obtained in Matlab and Octave respectively. To save in Octave is used:

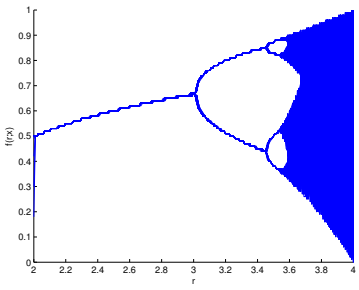
octave:5> print -color filename.eps

4.4.3 File menu: Export to Matlab

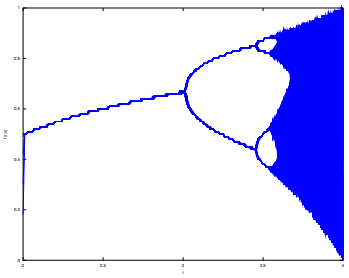
To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.

This is the code that is generated with the data showed in figure 4.13.

```
% DS Simulator
```



(a) Bifurcation diagram in MATLAB loading data



(b) Bifurcation diagram in Octave loading data

Figure 4.15: Bifurcation diagram in MATLAB and Octave loading data

```
% Bifurcation Diagram
clear all
close all
clc
x=0.1;
rr=1.0:0.01:4.0;
n=length(rr);
nn=100;
g=waitbar(0,'Processing...');
for i=1:n
r=rr(i);
for j=1:nn
xx(i,j)=r*x*(1-x);
x=xx(i,j);
end
waitbar(i/length(rr))
end
close(g)
```

```
hold on
for j=1:nn
plot(rr,xx(:,j),'r')
end
hold off
title('Bifurcation Diagram')
grid on
xlabel('r')
ylabel('p(r)')
```
Figure 4.16a shows the result.

4.4.4 File menu: Export to Octave

To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

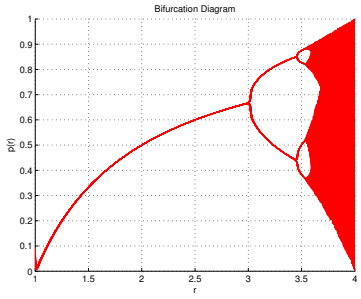
This is the code that is generated with the data showed in figure 4.13.

```
% DS Simulator
% Bifurcation diagram
clear all
close all
clc
nn(1)=0.0;
xx(1)=0.4;
n=50.0;
for i=1:n
x=xx(i);
nn(i+1)=nn(i)+1;
xx(i+1)=3.2*x*(1-x);
endfor
hold on
plot(nn,xx,'b-o')
grid on
xlabel('n')
ylabel('xn')
title('Bifurcation Diagram')
```
Figure 4.16b shows the result.

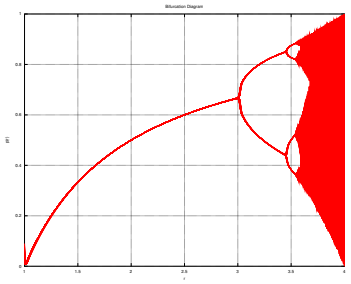
4.5 Lyapunov exponents

The Lyapunov exponent is defined as:

$$\lambda(x_0; r) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} ln \left| \frac{\partial f(x; r)}{\partial x} \right| \tag{4.1}$$



(a) Bifurcation diagram exporting code to MATLAB



(b) Bifurcation diagram exporting code to Octave

Figure 4.16: Code exported to Matlab and Octave from bifurcation diagram

where $f(x;r)$ is a function in discrete time that depends on the parameters.

If it had that$\lambda(x_0;r) > 0$, then the trajectory is chaotic; if $\lambda(x_0;r) < 0$, the trajectorie has regular behavior. The values of r for which $\lambda(x_0;r) = 0$ correspond to bifurcation points where the behavior regular changes to chaotic or vice versa [6].

Proposition 4.5.1 *Behaviors:*

- *The regular behavior occurs when a small change initial condition generates a small change in the distant future, the trajectory converges to a stable periodic orbit and the Lyapunov exponents is less than or equal to zero,* $\lambda \leq 0$.
- *The chaotic behavior occurs when the system has sensitivity to small changes in initial conditions, so in the distant future the path has an aperiodic behavior; Lyapunov exponent is positive* ($\lambda \geq 0$)*, indicating an exponential divergence of trajectories initially neighboring.*

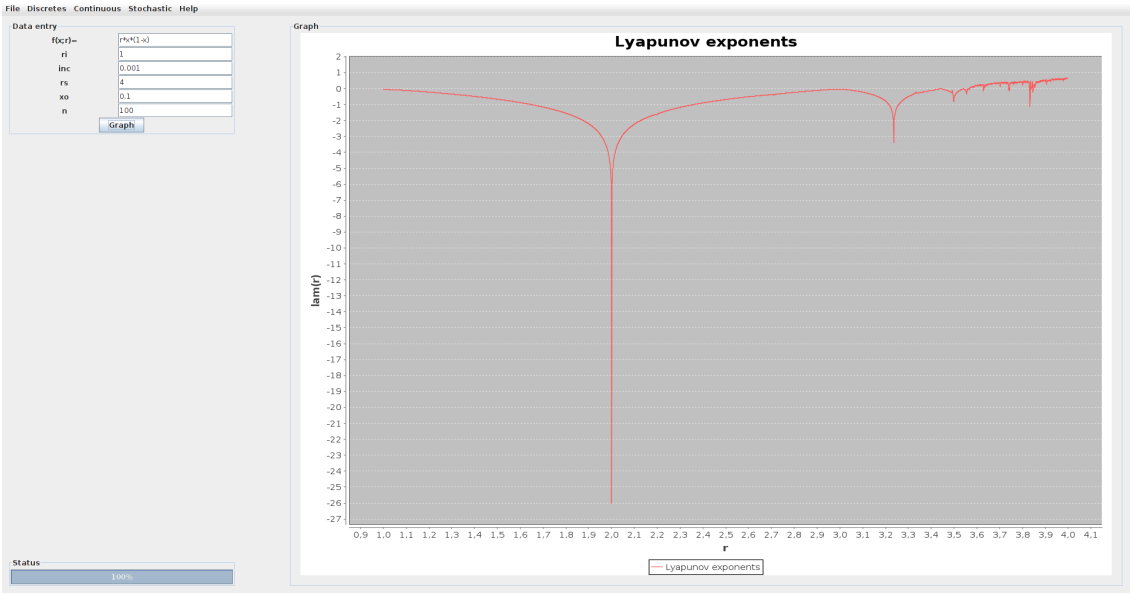


Figure 4.17: Lyapunov exponents $\lambda(r)$ as a function of the parameter r, for the function $f(x;r) = rx(1-x)$. In the interval $1 \leq r \leq 4$

The Lyapunov exponent vanishes ($\lambda(r_k) = 0$) for certain values r_k of the parameter r. In these it presented a bifurcation, as the system changes it behavior; for example, regular to chaotic behavior or vice versa.

The Lyapunov exponents are used to determine if a system is chaotic or not, allowing to better analyze the behavior of a discrete system. In the figure 4.17 is showed Lyapunov exponents of $f_r(x) = rx(1-x)$.

4.5.1 Gui

Figure 4.17 is explained:

- **Data entry panel:**

  f(x;r): is the function with parameter r.

  $I = [ri, rs]$ is the interval function f on the which is observed its behavior.

  inc: is the difference in the arithmetic progression which is the interval I. Take a very small increment value for the graph more accurate.

  x0: is the initial value of the variable x.

  n: is the number of iterations.

- **Graph panel:**

  Is where it looks the graph of the input data.

4.5.2 File menu: Save

To save must first to graph. It is saved in a flat file, TXT format, where the first column represent the parameter and the second column represent the values of lyapunov exponents.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where the first replaces the comma by point, and load file gnuplot.

All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done as follows in linux: perl -p -i -e 's/,/./g' graph.txt

Script file:
#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot
load 'graph.gnuplot'
EOF

Gnuplot file:
set term postscript eps enhanced
set output "LyapunovExponents.eps"
set size square
set xlabel "r"
set ylabel "{/Symbol l}(r)"
plot "graph.txt" ti "Lyapunov exponents" with lines

Once you save the file Gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as showed in figure 4.18. The data are the same as those used in figure 4.17.

In Matlab and Octave it can plot the data from txt file, saving a file m format the following code:

```
clear all
close all
clc
X=load('graph.txt');
[m,n]=size(X);
plot(X(:,1),X(:,2))
xlabel('r')
ylabel('\lambda(r)')
grid on
```

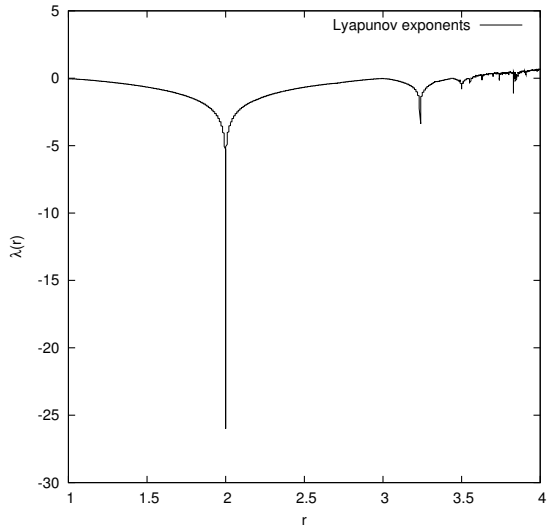


Figure 4.18: Lyapunov exponents using GNUPLOT

The figure 4.19 displays the graphs obtained in Matlab and Octave respectively. To save in Octave is used:

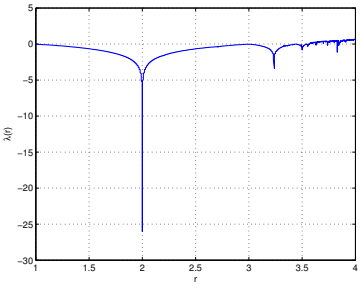
octave:5> print -color filename.eps

4.5.3 File menu: Export to Matlab

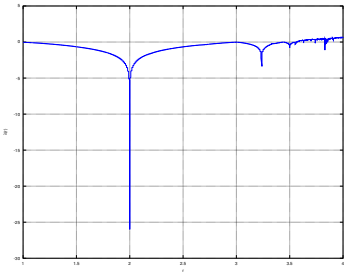
To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.

This is the code that is generated with the data showed in figure 4.17.

```
% DS Simulator
% Lyapunovs exponents
clear all
close all
clc
xx(1)=0.1;
rr=1.0:0.001:4.0;
nn=1000;
y=0;
for i=1:length(rr)
r=rr(i);
for j=1:nn
x=xx(j);
xx(j+1)=r*x*(1-x);
y=log(abs(- r*x - r*(x - 1)))+y;
end lam(i)=(1/nn)*y;
```



(a) Lyapunov exponents in MATLAB loading data



(b) Lyapunov exponents in Octave loading data

Figure 4.19: Lyapunov exponents in MATLAB and Octave loading data

```
y=0;
end
plot(rr,lam)
grid on
xlabel('r')
ylabel('\lambda(r)')
title('Lyapunovs Exponents')
```

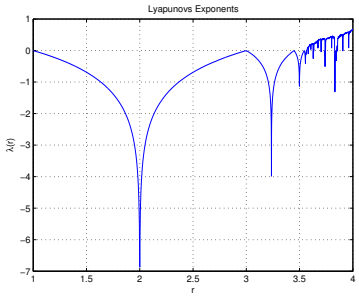
Figure 4.20a shows the result.

4.5.4 File menu: Export to Octave

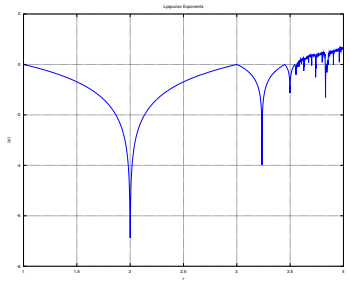
To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

This is the code that is generated with the data showed in figure 4.17.

```
% DS Simulator
% Lyapunov exponents
clear all
close all
```



(a) Lyapunov exponents exporting code to MATLAB



(b) Lyapunov exponents exporting code to Octave

Figure 4.20: Code exported to Matlab and Octave from Lyapunov exponents

```
clc
nn(1)=0.0;
xx(1)=0.4;
n=50.0;
for i=1:n
x=xx(i);
nn(i+1)=nn(i)+1;
xx(i+1)=3.2*x*(1-x);
endfor
hold on
plot(nn,xx,'b-o')
grid on
xlabel('n')
ylabel('xn')
title('Time serie')
```

Figure 4.20b shows the result.

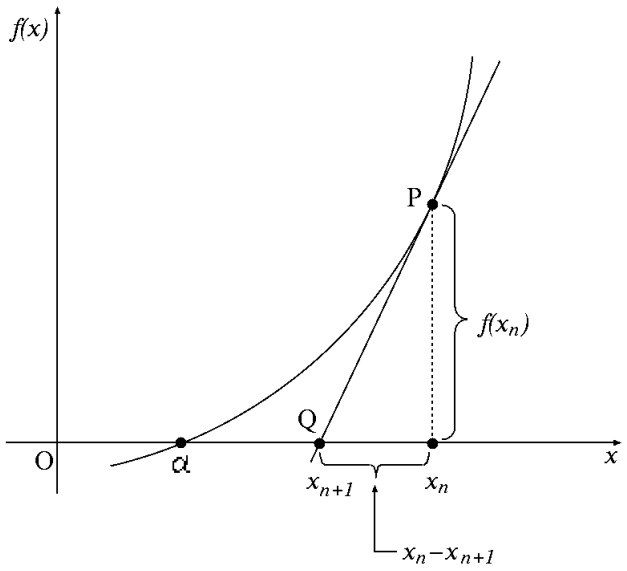


Figure 4.21: Graphic interpretation of Newton-Raphson formula

4.6 Fixed point

To find fixed points is used the Newton-Raphson method .

The Newton-Raphson method is a well-known algorithm to find roots of a function. It is very simple and very fast. The only disadvantage of method is using the derivative $f'(x)$ of the function and as the function $f(x)$. Therefore, the Newton-Raphson method is only usable on problems where $f'(x)$ is computable.

The Newton-Raphson formula that can be derived from the expansion the Taylor series of $f(x)$ on x:

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x_i) + O(x_{i+1} - x_i)^2 \tag{4.2}$$

If x_{i+1} is a root of $f(x) = 0$, then equation 4.2 become

$$0 = f(x_i) + f'(x_i)(x_{i+1} - x_i) + O(x_{i+1} - x_i)^2 \tag{4.3}$$

assuming that x_i to x_{i+1} is closed, it can delete the last term in equation 4.3 and solve for x_{i+1}. The result is the formula of Newton-Raphson.

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{4.4}$$

The method is combined with the bisection method, which requires a interval that may be the root, $I = [a, b]$, where is taken as initial value $x_0 = \frac{a+b}{2}$. Being an iterative method, it stops when the function evaluated at nth root is less than

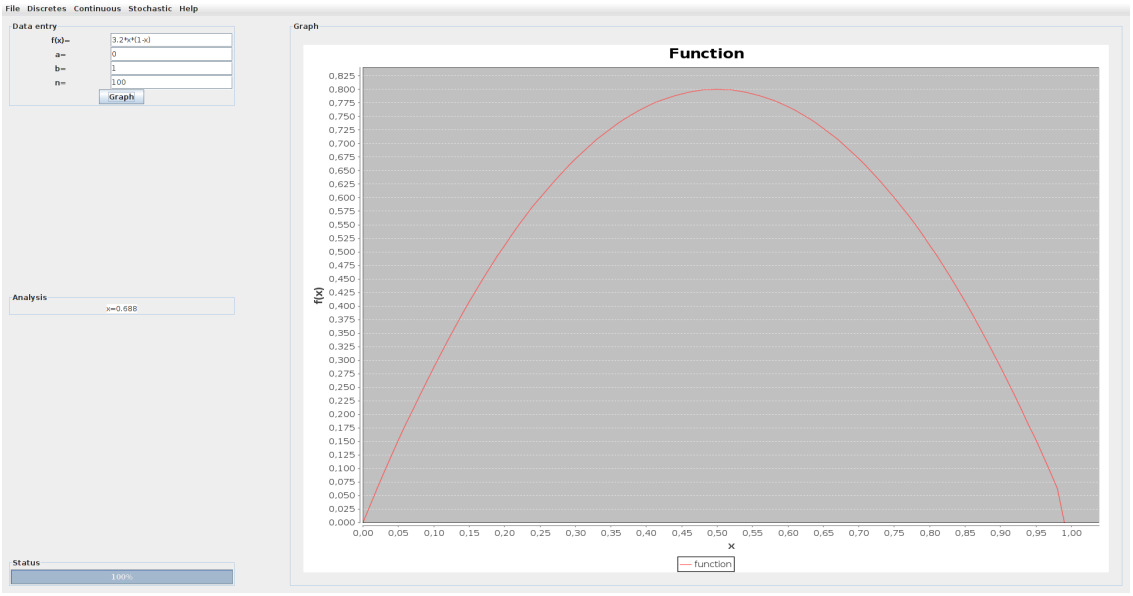


Figure 4.22: Newton-Raphson method

an epsilon, that is $f(x_{i+1}) < \epsilon$, $\epsilon = 1 \times 10^{-1(k+1)}$, where k is the number of significant figures in the desired root [7].

If the values of x converge to a single value as a fixed point, the value that converge is the desired root, if the method diverge is not possible to find a root in that interval.

In figure 4.22 is showed the root of $f(x) = 3.2x(1-x) - x$ in the interval [0, 1].

4.6.1 Gui

Figure 4.22 is explained:

- **Data entry panel:**

  f(x;r): is the function.

  $I = [a, b]$ is the interval in where it found the root of the function f.

  n: is the number of iterations.

- **Graph panel:**

  Is where it looks the graph of the input data.

- **Analysis panel:**

  Shows the fixed of the system.

Chapter 5

CONTINUOUS DYNAMICAL SYSTEMS

Are mathematical models that evolve in time continuously, represented by differential equations. These systems are most common in the physical world, and can be linear and nonlinear.

Dynamical systems are divided into systems of linear differential and nonlinear equations.

$$\begin{aligned} \frac{dx}{dt} &= f(x,y) \\ \frac{dy}{dt} &= g(x,y) \end{aligned} \tag{5.1}$$

If the system 5.1 is linear can be written as follows:

$$\frac{d\mathbf{Y}}{dt} = A\mathbf{Y} \tag{5.2}$$

where A represent the coefficient matrix of the equations system 5.1 and $\mathbf{Y}$ the column vector of dependent variables that is $\mathbf{Y}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$.

Is called equilibrium point (x_0, y_0) to the respective values of x and y obtained from

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned} \tag{5.3}$$

The dynamical system 5.1 is first order with two state variables x and y, a system can be n state variables, but when the number of state variables are equal or exceed 3 are very complex and tend to be chaotic.

Linear systems with a single equilibrium point that is the origin. To analyze this single equilibrium point using the values that are the roots of a polynomial, called the *characteristic polynomial*, which is using the matrix A of the system 5.3 and the identity matrix I multiplied by the eigenvalue λ, as follows:

$$det(A - I\lambda) = 0 \tag{5.4}$$

A generic method for finding the characteristic polynomial is given by

$$\lambda^2 - T\lambda + D = 0 \tag{5.5}$$

where T represent the trace of the matrix A and D the determinant of the matrix A. Once it obtanied the characteristic polynomial 5.5 are its roots, which are the values of the system, which in this case two eigenvalues are obtained λ_1 and λ_2. With these two values is classified the equilibrium point using the following cases:

Case 5.0.1 *Let be* $\lambda_1, \lambda_2 \in \mathbb{R}$
if $\lambda_1 < 0 < \lambda_2$ *is a saddle point*
if $\lambda_1 < \lambda_2 < 0$ *is a sink*
if $\lambda_1 > \lambda_2 > 0$ *is a source*

Case 5.0.2 *Let be* $\lambda_1, \lambda_2 \in \mathbb{C}$ *where* $\lambda_1 = \alpha + i\beta$ *and* $\lambda_2 = \alpha - i\beta$
if $\alpha < 0$ *is a spiral sink*
if $\alpha > 0$ *is a spiral source*
if $alpha = 0$ *is a center*

Case 5.0.3 *Let be* $\lambda_1 = \lambda_2 = \lambda$
if $\lambda < 0$ *is a sink*
if $\lambda > 0$ *is a source*

Case 5.0.4 *Let be* $\lambda_1 = 0$ *and* $\lambda_2 \neq 0$
if $\lambda_2 < 0$ *is a sink*
if $\lambda_2 > 0$ *is a source*

With these cases be found the stability of the system, if the point of equilibrium corresponds to a sink or spiral sink the system will be stable, otherwise unstable, this can be viewed graphically through the vector field which indicates how fast moves the system over time and what direction. If a system tends to infinity in its solutions it will be unstable. The following are the general solutions of the system for each case mentioned.

- For the case 5.0.1:

$$Y(t) = K_1 e^{\lambda_1 t} V_1 + K_2 e^{\lambda_2 t} V_2 \tag{5.6}$$

  where V_i represents the eigenvector matrix A corresponding to the eigenvalue λ_i, the definition of an eigenvector is one in which the vector field pointing in the same direction or opposite to the vector itself [5] and is calculated as follows:

$$AV_i = \lambda_i V_i \tag{5.7}$$

  Here and in all cases $i = 1, 2$.

- For the case 5.0.2:

$$\begin{aligned} Y(t) &= e^{(\alpha+i\beta)t}V_o \\ Y(t) &= k_1 Y_{real} + k_2 Y_{imag} \end{aligned} \tag{5.8}$$

In where $e^{(\alpha+i\beta)t} = e^{\alpha t}cos(\beta t) + ie^{(\alpha t)}sin(\beta t)$ and V_0 is the eigenvector of matrix A corresponding to the eigenvalue λ_1.

- For the case 5.0.3:

$$Y(t) = k_1 e^{\lambda t}V_1 + k_2 e^{\lambda t}(tV_1 + V_2) \tag{5.9}$$

that in this situation by presenting a single eigenvalue λ, V_2 is calculated as follows:

$$(A - I\lambda)V_2 = V_1 \tag{5.10}$$

- For the case 5.0.4:

$$Y(t) = k_1 V_1 + k_2 e^{\lambda_2 t}V_2 \tag{5.11}$$

In a three-dimensional system is applied the following cases to classify the equilibrium point:

Case 5.0.5 *Let be* $\lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}$
if $\lambda_1 < \lambda_2 < \lambda_3 < 0$ *is a sink*
if $\lambda_1 > \lambda_2 > \lambda_3 > 0$ *is a source otherwise is a saddle point*

Case 5.0.6 *Let be* $\lambda_i \in (\mathbb{R}, \mathbb{C}), \lambda_1 \in \mathbb{R} and \lambda_2, \lambda_3 \in \mathbb{C}$
if $\lambda_1 < 0$ *and* $\mathbb{R}e(\lambda_2) > 0$ *is a spiral saddle point*
if $\lambda_1 < 0$ *and* $\mathbb{R}e(\lambda_2) < 0$ *is a spiral sink*
if $\lambda_1 > 0$ *and* $\mathbb{R}e(\lambda_2) > 0$ *is a spiral source*

Thus has been analyzed behavior and stability of a linear system.

5.0.2 Linearization

It is a method to find the stability of a nonlinear system. To approximate a nonlinear system to an equilibrium point by a linear system is what is called linearization and is applied frequently, studying the linear approximation, can be predicted the behavior of the solutions of the nonlinear system, at least near the equilibrium point [6]. Let be $f_1, f_2, ..., f_n$ the number of equations of a nonlinear dynamic system and $x_1, x_2, ..., x_n$ the number of variables of the system, it calculated the Jacobian matrix 5.12 evaluated at the respective

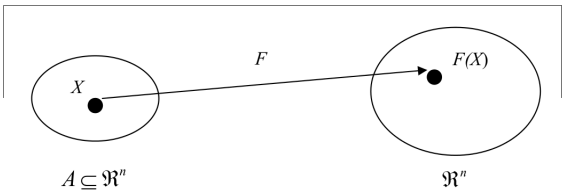


Figure 5.1: Schematic representation of a vector field defined by $F : A \subseteq \mathbb{R}^n \to \mathbb{R}^n$

equilibrium point to be analyzed. This matrix is now the matrix A and thus the system 5.1 is linearized and proceeded to obtain the characteristic polynomial as is done in a linear system to meet stability and performance.

$$J(x_1, x_2, ..., x_n) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{5.12}$$

5.1 Vector Field

The vector field is the field slope to solution curves of the phase portrait.

A vector field in $\mathbb{R}^n$ is a function $F : A \subseteq \mathbb{R}^n \to \mathbb{R}^n$ that assigns to each point $X = (x_1, x_2, ..., x_n)$ of its domain A a vector $F(X) = (F_1(X), F_2(X), ..., F_n(X))$.

If $F : A \subseteq \mathbb{R}^2 \to \mathbb{R}^2$ then referred to as *vector field on the plane*, this function $F(x, y)$ defined for points in $\mathbb{R}^2$ to the two-dimensional set of vectors, and is written

$$F(x, y) = \begin{bmatrix} F_1(x, y) \\ F_2(x, y) \end{bmatrix} = (F_1(x, y), F_2(x, y)) \tag{5.13}$$

in where, $F_1(x, y)$ and $F_2(x, y)$ are scalar functions.

Figure 5.1 shows a schematic form of to represent a vector field, of $\mathbb{R}^n \to \mathbb{R}^n$.

However, for a better way to visualize what the field represents in $\mathbb{R}^n$, is preferable to draw the vector $X \in A \subseteq \mathbb{R}^n$ as a point about space $\mathbb{R}^n$ and to $F(X) \in \mathbb{R}^n$, as a vector for the same space, as showed in Figure 1.

The representation of a two-dimensional vector field in the Cartesian plane, performed representing a set of vectors $F(x, y)$ for several points (x, y) of the domain, representing the vector $F(x, y) = (F_1(x, y), F_2(x, y))$ so that the starting point of the vector is located in (x, y); this procedure can also be applied to the representation of a vector field in space.

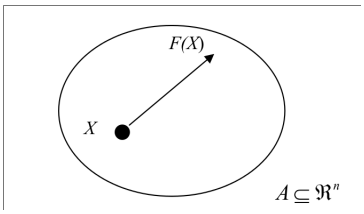


Figure 5.2: Representation of a vector field on R^n, defined by $F : A \subseteq \mathbb{R}^n \to \mathbb{R}^n$

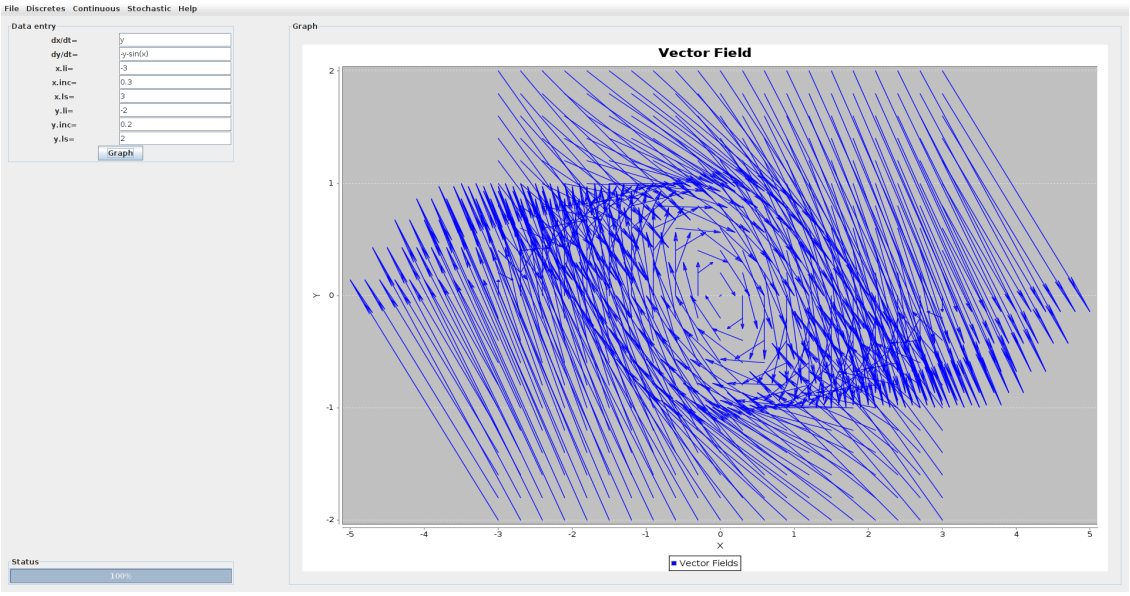


Figure 5.3: Vector field of the model of motion of a nonlinear pendulum

5.1.1 Gui

Figure 5.3 is explained:

- **Data entry panel:**

  dx/dt and *dy/dt:* are the function in terms of x and y representing the functions F_1 and F_2 of the equation 5.13.

  $I_x = [x.li, x.ls]$ is the interval of the system in the x-axis on the which is observed its vector field.

  x.inc: is the difference in the arithmetic progression which is the interval I_x. Takes a value of 0.3 and you adjust.

  $I_y = [x.li, x.ls]$ is the interval of the system in the y-axis on the which is observed its vector field.

  y.inc: is the difference in the arithmetic progression which is the interval I_y. Takes a value of 0.3 and you adjust.

- **Graph panel:**

  Is where it looks the graph of the input data.

5.1.2 File menu: Save

To save must first to graph. It is saved in a flat file, TXT format, comprising n columns, represented as follows:

data x: between the first column and $\frac{n}{4} - 1$ column.

data y: between the $\frac{n}{4}$ column and $\frac{2n}{4} - 1$ column.

data dx: between the $\frac{2n}{4}$ column and $\frac{3n}{4} - 1$ column.

data dy: between the $\frac{3n}{4}$ column and n column.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in Gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where the first replaces the comma by point, and load file gnuplot.

All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done as follows in linux: perl -p -i -e 's/,/./g' graph.txt

Script file:

```
#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot
load 'graph.gnuplot'
EOF
```

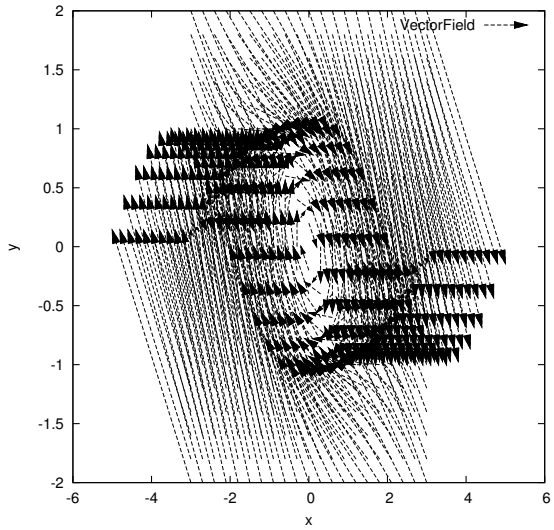


Figure 5.4: Vector Field using GNUPLOT

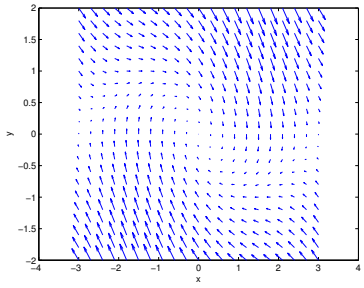
Gnuplot file:
set term postscript eps enhanced
set output "VectorField.eps"
set size square
set xlabel "x"
set ylabel "y"
plot "graph.txt" using 1:2:3:4 ti "VectorField" with vectors head filled lt 2

Once you save the file gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as shown in Figure 5.4. The data are the same as those used in figure 5.3.

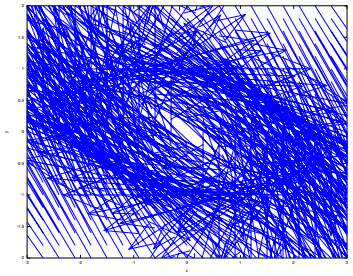
In Matlab and Octave it can plot the data from txt file, saving a file m format the following code:

```
clear all
close all
clc
X=load('graph.txt');
quiver(X(:,1),X(:,2),X(:,3),X(:,4))
xlabel('x')
ylabel('y')
```

The figure 5.5 displays the graphs obtained in Matlab and Octave respectively. To save in Octave is used:
octave:5> print -color filename.eps



(a) Vector field in MATLAB loading data



(b) Vector field in Octave loading data

Figure 5.5: Vector field in MATLAB and Octave loading data

5.1.3 File menu: Export to Matlab

To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.

This is the code that is generated with the data showed in figure 5.3.

```
% DS Simulator
% Vector Field
clear all
close all
rangox=-3.0:0.3:3.0;
rangoy=-2.0:0.2:2.0;
[xx,yy]=meshgrid(rangox,rangoy);
[n,m]=size(xx);
for i=1:n
for j=1:m x=xx(i,j);
y=yy(i,j);
dx(i,j)=y;
dy(i,j)=-y-sin(x);
end
end
```

```
quiver(xx,yy,dx,dy,10);
xlabel('x');
ylabel('y');
```
Figure 5.6a shows the result.

5.1.4 File menu: Export to Octave

To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

This is the code that is generated with the data showed in figure 5.3.

```
% DS Simulator
% Vector field
clear all
close all
rangox=-3.0:0.3:3.0;
rangoy=-2.0:0.2:2.0;
[xx,yy]=meshgrid(rangox,rangoy);
[n,m]=size(xx);
for i=1:n
for j=1:m
x=xx(i,j);
y=yy(i,j);
dx(i,j)=y;
dy(i,j)=-y-sin(x);
endfor
endfor
quiver(xx,yy,dx,dy,10);
xlabel('x');
ylabel('y');
```
Figure 5.6b shows the result.

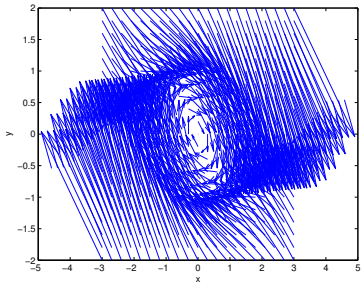
5.2 Phase portrait

A phase portrait is a geometric representation of the trajectories of a dynamical system in the phase plane. Each set of initial conditions is representated by a different curve, or point.

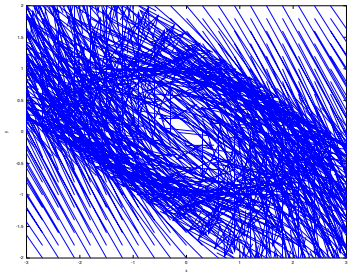
Phase portraits are an invaluable tool in the study of dynamical systems. They consist of a plot of typical trajectories in the state space. This reveals information such as whether an attractor, a repellor or limit cycle is present for the chosen parameter value. The concept of topological equivalence is important in classifying the behaviour of systems by specifying when two different phase portraits represent the same qualitative dynamic behavior [8].

In Tables 5.1 and 5.2 are showed typical phase portraits in dynamical systems.

To generate the phase portrait is used Runge Kutta method of order 4.



(a) Vector field exporting code to MATLAB



(b) Vector field exporting code to Octave

Figure 5.6: Code exported to Matlab and Octave from Vector field

5.2.1 Runge Kutta

The formula for the Euler method is

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{5.14}$$

which advances a solution from x_n to$x_{n+1} \equiv x_n + h$. The formula is unsymmetrical: It advances the solution through an interval h, but uses derivative information only at the beginning of that interval (see Figure 5.9). That means (and you can verify by expansion in power series) that the step's error is only one power of h smaller than the correction, i.e $O(h^2)$ added to 5.14.

There are several reasons that Euler's method is not recommended for practical use, among them, (i) the method is not very accurate when compared to other, fancier, methods run at the equivalent stepsize, and (ii) neither is it very stable.

Consider, however, the use of a step like 5.14 to take a "trial" step to the midpoint of the interval. Then use the value of both x and y at that midpoint to compute the "real" step across the whole interval. Figure 5.10 illustrates the idea. In equations,

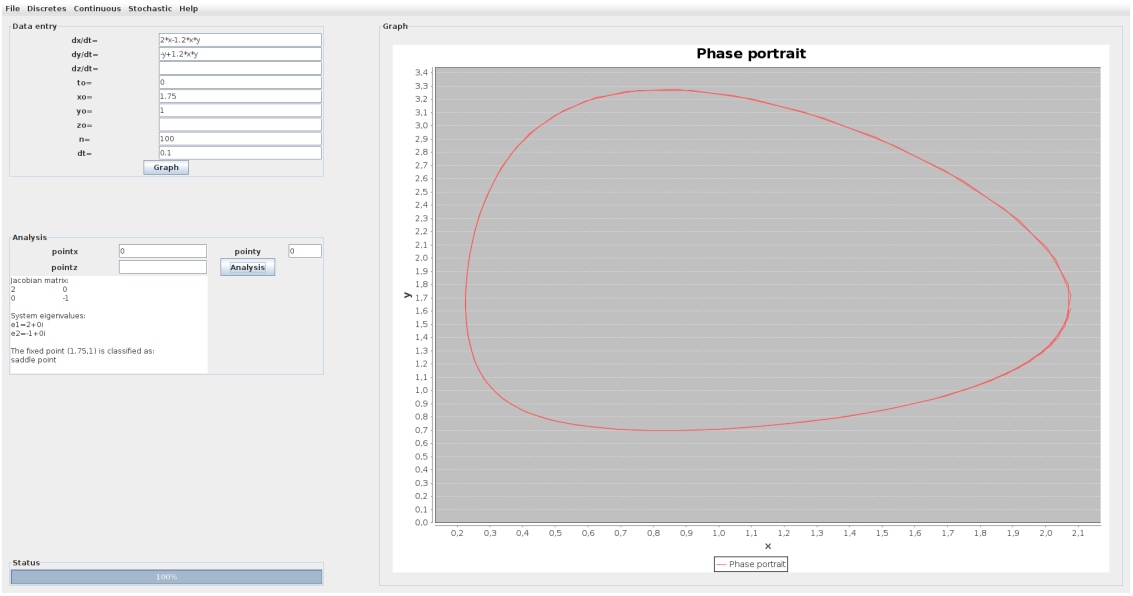


Figure 5.7: Phase portrait of a predator-prey system

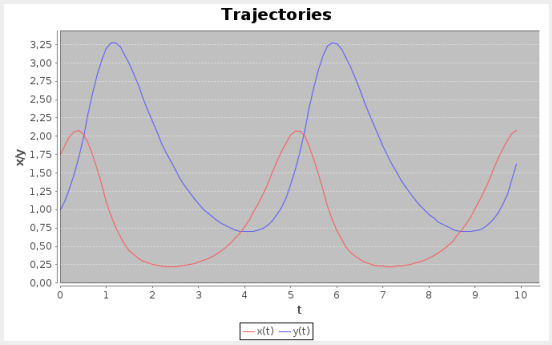
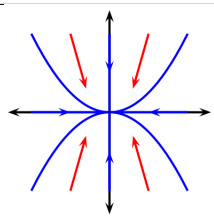
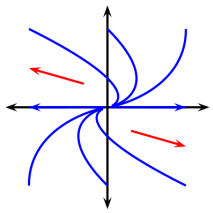
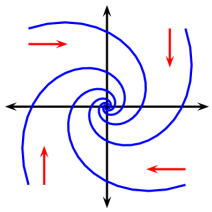
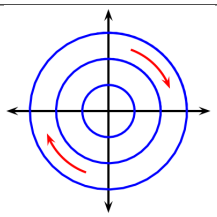


Figure 5.8: Trajectories of the predator-prey system

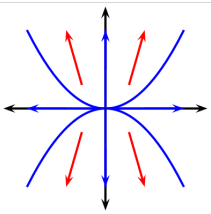
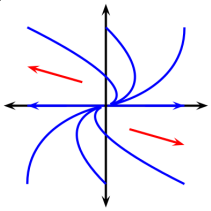
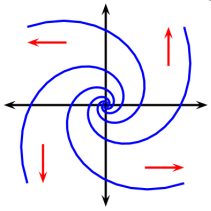
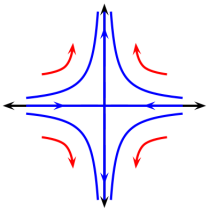
Tabla 5.1: Stable phase portraits typical

Name	**Portrait**	**values λ**
Stable		Negative real
Stable multiplicity 2		Repeated negative real
Siphon		Complex with negative real part
Center		Pure imaginary

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\ y_{n+1} &= y_n + k_2 + O(h^3) \end{aligned} \tag{5.15}$$

As indicated in the error term, this symmetrization cancels out the first-order error term, making the method second order. [A method is conventionally called nth order if its error term is $O(h^{n+1})$.] In fact, 5.15 is called the *second-order Runge-Kutta* or *midpoint method*.

Tabla 5.2: Unstable phase portraits typical

Name	**Portrait**	**values λ**
Unstable		Positive real
Unstable multiplicity 2		Repetid positive real
Source		Complex with positive real part
Saddle point		Real of different sign

It not needed stop there. There are many ways to evaluate the right-hand side$f(x, y)$ that all agree to first order, but that have different coefficients of higher-order error terms. Adding up the right combination of these, be can eliminated the error terms order by order. That is the basic idea of the Runge-Kutta method. Abramowitz and Stegun, and Gear, give various specific formulas that derive from this basic idea. By far the most often used is the classical fourth-order Runge-Kutta formula, which has a certain sleekness of organization about it:

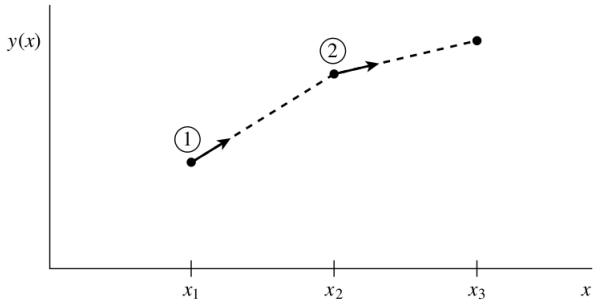


Figure 5.9: Euler's method. In this simplest (and least accurate) method for integrating an ODE, the derivative at the starting point of each interval is extrapolated to find the next function value. The method has first-order accuracy

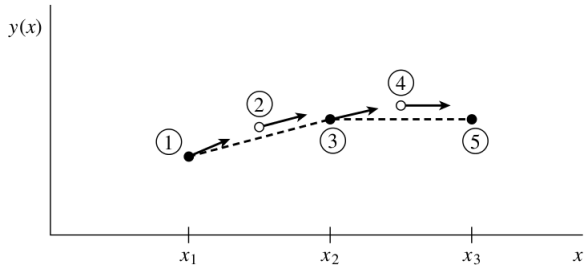


Figure 5.10: Midpoint method. Second-order accuracy is obtained by using the initial derivative at each step to find a point halfway across the interval, then using the midpoint derivative across the full width of the interval. In the figure, filled dots represent final function values, while open dots represent function values that are discarded once their derivatives have been calculated and used.

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\ k_3 &= hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \\ k_4 &= hf(x_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5) \end{aligned} \tag{5.16}$$

The fourth-order Runge-Kutta method requires four evaluations of the right-hand side per step h (see Figure 5.11). This will be superior to the midpoint method 5.15 if at least twice as large a step is possible with 5.16 for the same accuracy. Is that so? The answer is: often, perhaps even usually, but surely not always! This takes us back to a central theme, namely that *high order* does not always mean high accuracy. The statement "fourth-order Runge-Kutta is

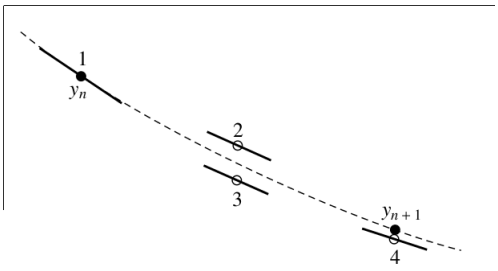


Figure 5.11: Fourth-order Runge-Kutta method. In each step the derivative is evaluated four times: once at the initial point, twice at trial midpoints, and once at a trial endpoint. From these derivatives the final function value (shown as a filled dot) is calculated.

generally superior to second-order" is a true one, but you should recognize it as a statement about the contemporary practice of science rather than as a statement about strict mathematics. That is, it reflects the nature of the problems that contemporary scientists like to solve [9].

5.2.2 Gui

Figure 5.7 is explained:

- **Data entry panel:**

  dx/dt, dy/dt and dz/dt: are the function in terms of x and y representing the functions F_1, F_2 and F_3 of the equations system.

  t0: is the start time.

  x0, y0 and *z0:* are the initial conditions at time *t0.*

  n: is the number of points of the solution obtained by the Runge Kutta method (see equation 5.16).

  dt: is the step size (h) of Runge Kutta method.

- **Graph panel:**

  Is where it looks the graph of the input data, can be selected a point on this graph and it displayed in the panel analysis.

- **Analysis panel:**

  Point x and *Point y* shows the point of the x-axis and y-axis respectively that were selected from the graph panel, or also can be entered by the user. The *point z* can only be entered. By analyzing shows the classification of equilibrium point according to the cases 5.0.1, 5.0.2, 5.0.3 and 5.0.4 or 5.0.5 and 5.0.6 if two or three the number of equations respectively, if is one equation is classified the point according to proposition 4.1.1. When you

run the analyze button displays the Jacobian matrix, the eigenvalues and the equilibrium points of the system with their respective classification.

5.2.3 File menu: Save

To save must first to graph. It is saved in a flat file, TXT format, where the first two columns represent time data, the next column represent data variables x, y and z, depending on the number of equations of the system entered.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in Gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where the first replaces the comma by point, and load file gnuplot.

All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done as follows in linux: perl -p -i -e 's/,/./g' graph.txt

Script file:

#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot
load 'graph.gnuplot'
EOF

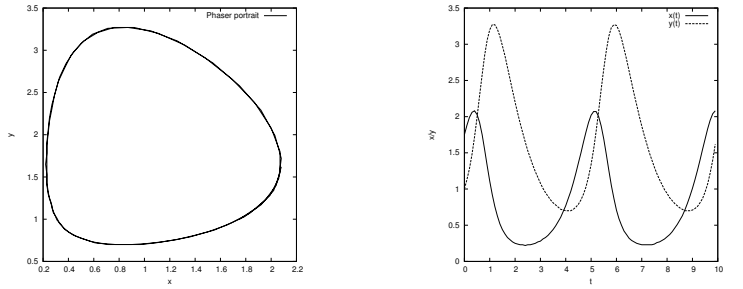
Gnuplot file:

set term postscript eps enhanced
set output "Trajectories.eps"
set size square
set xlabel "t"
set ylabel "x/y"
plot "graph.txt" using 1:2 ti "x(t)" with lines, "graph.txt" using 1:3 ti "y(t)" with lines
set term postscript eps enhanced
set output "Phaser portrait.eps"
set size square
set xlabel "x" set ylabel "y"
plot "graph.txt" using 2:3 ti "Phaser portrait" with lines

Once you save the file gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as showed in figure 5.12. The data are the same as those used in figure 5.7.

In Matlab and Octave can plot the data from txt file, saving a file m format the following code:

```
clear all
close all
clc
```



(a) Phaser portrait using GNUPLOT (b) Trajectories using GNUPLOT

Figura 5.12: Phaser portrait using GNUPLOT

```
X=load('graph.txt');
figure(1)
hold on
plot(X(:,1),X(:,2),'b')
plot(X(:,1),X(:,3),'g')
xlabel('t')
ylabel('x/y')
legend('x(t)','y(t)')
grid on
figure(2)
plot(X(:,2),X(:,3))
title('Phaser portrait')
grid on
```

The figure 5.13 displays the graphs obtained in Matlab and Octave respectively. To save in Octave is used:

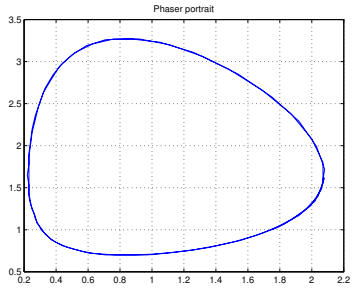
octave:5> print -color filename.eps

5.2.4 File menu: Export to Matlab

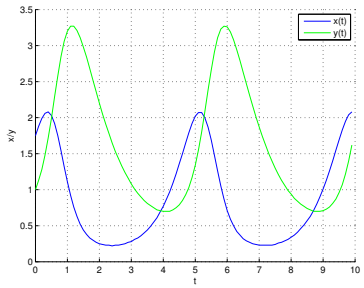
To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.

This is the code that is generated with the data showed in figure 5.7.

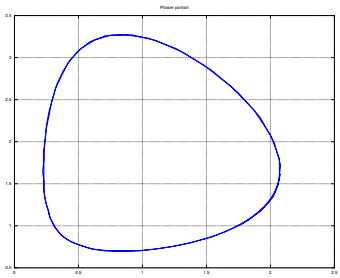
```
% DS Simulator
% Phase portrait
function varargout = phasep(varargin)
run;
function run()
X=RK4([1.75, 1.0],0.0,0.1,100.0);
figure(1)
plot(X(:,2),X(:,3))
title('phase portrait')
```



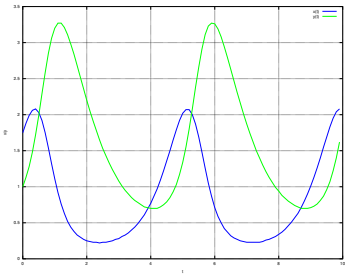
(a) Phaser portrait in MATLAB loading data



(b) Trajectories in Matlab loading data



(c) Phaser portrait in Octave loading data



(d) Trajectories in Octave loading data

Figura 5.13: Phaser portrait loading data in MATLAB and Octave

```
grid on
xlabel('x')
ylabel('y')
figure(2)
hold on
plot(X(:,1),X(:,2),'r')
plot(X(:,1),X(:,3),'b')
title('Trajectories')
grid on
xlabel('t')
ylabel('x/y')
legend('x(t)','y(t)')
function X=RK4(yo, to, dt, n)
n1=length(yo);
X=zeros(n,n1+1);
fi=zeros(n1,4);
xy=zeros(n1,1);
X(1,1)=to;
```

```
for i=2:n1+1
X(1,i)=yo(i-1);
end
for i=1:n-1
X(i+1,1)=X(i,1)+dt;
for l=2:n1+1
xy(l-1)=X(i,l);
end
fi=step(xy,X(i,1),dt);
for k=2:n1+1
X(i+1,k)=X(i,k)+(dt/6.0)*(fi(k-1,1)+2.0*fi(k-1,2)+2.0*fi(k-1,3)+fi(k-1,4));

end
end
return
function fi= step(yk, tk, dt)
x=0;y=0;z=0;t=0;
n1=length(yk);
fi=zeros(n1,4);
x=yk(1);
y=yk(2);
t=tk;
fi(1,1)=2*x-1.2*x*y;
fi(2,1)=-y+1.2*x*y;
x=yk(1)+(dt/2)*fi(1,1);
y=yk(2)+(dt/2)*fi(2,1);
fi(1,2)=2*x-1.2*x*y;
fi(2,2)=-y+1.2*x*y;
x=yk(1)+(dt/2)*fi(1,2);
y=yk(2)+(dt/2)*fi(2,2);
fi(1,3)=2*x-1.2*x*y;
fi(2,3)=-y+1.2*x*y;
x=yk(1)+dt*fi(1,3);
y=yk(2)+dt*fi(2,3);
fi(1,4)=2*x-1.2*x*y;
fi(2,4)=-y+1.2*x*y;
```

Figure 5.14 shows the result.

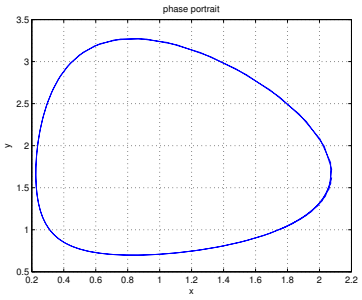
5.2.5 File menu: Export to Octave

To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

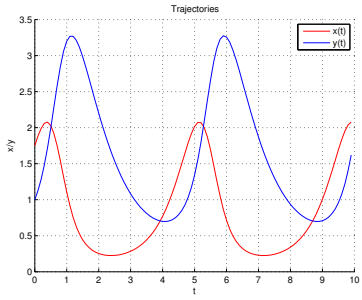
This is the code that is generated with the data showed in figure 5.7.

```
% DS Simulator
% Phase portrait
function varargout = phasep(varargin)
```

```
run;
function run()
X=RK4([1.75, 1.0],0.0,0.1,100.0);
figure(1)
plot(X(:,2),X(:,3))
title('phase portrait')
grid on
xlabel('x')
ylabel('y')
figure(2)
hold on
plot(X(:,1),X(:,2),'r')
plot(X(:,1),X(:,3),'b')
title('Trajectories')
grid on
xlabel('t')
ylabel('x/y')
legend('x(t)','y(t)')
function X=RK4(yo, to, dt, n)
n1=length(yo);
X=zeros(n,n1+1);
fi=zeros(n1,4);
xy=zeros(n1,1);
X(1,1)=to;
for i=2:n1+1
X(1,i)=yo(i-1);
endfor
for i=1:n-1
X(i+1,1)=X(i,1)+dt;
for l=2:n1+1
xy(l-1)=X(i,l);
endfor
fi=step(xy,X(i,1),dt);
for k=2:n1+1
X(i+1,k)=X(i,k)+(dt/6.0)*(fi(k-1,1)+2.0*fi(k-1,2)+2.0*fi(k-1,3)+fi(k-1,4));
endfor
endfor
return
function fi= step(yk, tk, dt)
x=0;y=0;z=0;t=0;
n1=length(yk);
fi=zeros(n1,4);
x=yk(1);
y=yk(2);
t=tk;
fi(1,1)=2*x-1.2*x*y;
```

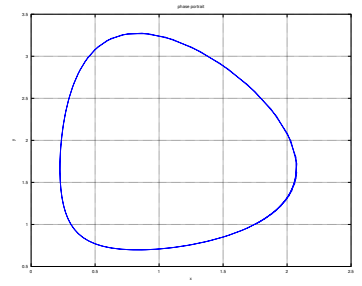


(a) Phaser portrait exporting code to MATLAB

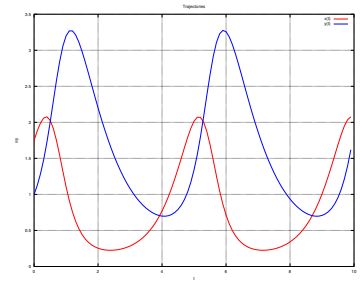


(b) Trajectories exporting code to Matlab

Figura 5.14: Code exported to Matlab from Phaser portrait



(a) Phaser portrait exporting code to Octave



(b) Trajectories exporting code to Octave

Figura 5.15: Code exported to Octave from Phaser portrait

```
fi(2,1)=-y+1.2*x*y;
x=yk(1)+(dt/2)*fi(1,1);
y=yk(2)+(dt/2)*fi(2,1);
fi(1,2)=2*x-1.2*x*y;
fi(2,2)=-y+1.2*x*y;
x=yk(1)+(dt/2)*fi(1,2);
y=yk(2)+(dt/2)*fi(2,2);
fi(1,3)=2*x-1.2*x*y;
fi(2,3)=-y+1.2*x*y;
x=yk(1)+dt*fi(1,3);
y=yk(2)+dt*fi(2,3);
fi(1,4)=2*x-1.2*x*y;
fi(2,4)=-y+1.2*x*y;
```

Figure 5.15 shows the result.

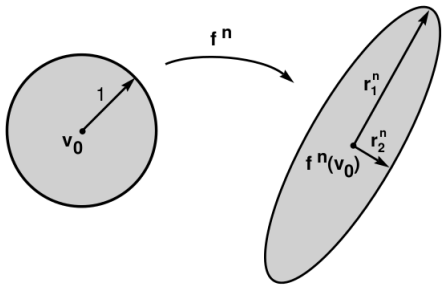


Figure 5.16: Evolution of an initial infinitesimal disk. After n iterates of a two-dimensional map, the disk is mapped into an ellipse

5.3 Lyapunov exponents

For a map on $\mathbb{R}^m$, each orbit has m Lyapunov numbers, which measure the rates of separation from the current orbit point along m orthogonal directions. These directions are determined by the dynamics of the map. The first will be the direction along which the separation between nearby points is the greatest (or which is least contracting, if the map is contracting in all directions). The second will be the direction of greatest separation, chosen from all directions perpendicular to the first. The third will have the most stretching of all directions perpendicular to the first two directions, and so on. The stretching factors in each of these chosen directions are the Lyapunov numbers of the orbit.

Figures 5.16 and 5.17 illustrate this concept pictorially. Consider a sphere of small radius centered on the first point v_0 of the orbit. If we examine the image $f(S)$ of the small sphere under one iteration of the map, we see an approximately ellipsoidal shape, with long axes along expanding directions for f and short axes along contracting directions.

After n iterates of the map f, the small sphere will have evolved into a longer and thinner ellipsoid-like object. The per-iterate changes of the axes of this image "ellipsoid" are the Lyapunov numbers. They quantify the amount of stretching and shrinking due to the dynamics near the orbit beginning at v_0 . The natural logarithm of each Lyapunov number is a Lyapunov exponent. For the formal definition, replace the small sphere about v_0 and the map f by the unit sphere N and the first derivative matrix $Df(v_0)$, since we are interested in the infinitesimal behavior near v_0 . Let $J_n = Df^n(v_0)$ denote the first derivative matrix of the nth iterate of f. Then J_nN will be an ellipsoid with m orthogonal axes. This is because for every matrix A, the image AN is necessarily an ellipsoid.

The axes will be longer than 1 in expanding directions of $f^n(v_0)$, and shorter than 1 in contracting directions, as shown in Figure 5.16. The m average multiplicative expansion rates of the m orthogonal axes are the Lyapunov numbers.

Definition 5.3.1 *Let f be a smooth map on $\mathbb{R}^m$, let $J_n = Df^n(v_0)$, and*

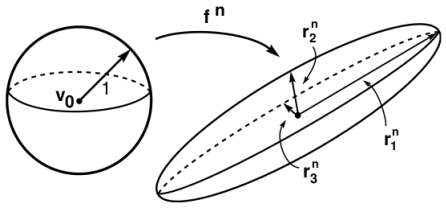


Figure 5.17: A three-dimensional version of Figure 5.16. A small ball and its image after n iterates, for a three-dimensional map

$fork = 1, ..., m$, let r_k^n be the length of the kth longest orthogonal axis of the n ellipsoid J_n N for an orbit with initial point v_0 . Then r_k^n measures the contraction or expansion near the orbit of v_0 during the first n iterations. The kth Lyapunov number of v_0 is defined by

$$L_k = \lim_{n\to\infty} (r_k^n)^(1/n),$$

if this limit exists. The kth Lyapunov exponent of v_0 is $h_k = lnL_k$. Notice that we have built into the definition the property that $L_1 \geq L_2 \geq ... \geq L_m$ and $h_1 \geq h_2 \geq ... \geq h_m$.

If N is the unit sphere in $\mathbb{R}^n$ and A is an mxm matrix, then the orthogonal axes of the ellipsoid AN can be computed in a straightforward way. The lengths of the axes are the square roots of the m eigenvalues of the matrix AA^T , and the axis directions are given by the m corresponding orthonormal eigenvectors. Using the concept of Lyapunov exponent, be can extended the definition of chaotic orbit to orbits of higher dimensional maps.

Definition 5.3.2 *Let f be a map of $\mathbb{R}^m$, $m \geq 1$, and let $\{v_0, v_1, v_2, ...\}$ be a bounded orbit of f. The orbit is chaotic if*
1. it is not asymptotically periodic,
2. no Lyapunov number is exactly one, and
3. $L_1(v_0) \geq 1$.

In terms of Lyapunov exponents, part 3 of Definition 5.3.2 is equivalent to $h_1(v_0) \geq 0$ [10].

5.3.1 Numerical calculation of lyapunov exponents

For most interesting maps, there is no direct way to determine Lyapunov exponents from knowledge of the map and its Jacobian matrices. Normally, the matrix $J_n = Df^n(v_0)$ is difficult to determine exactly for large n, and be must resort to the approximation of the image ellipsoid J_nN of the unit sphere by computational algorithms.

The ellipsoid J_nN has semi-major axes of length s_i in the directions u_i . The direct approach to calculating the Lyapunov exponents would be to

explicitly form $J_n J_n^T$ and find its eigenvalues s_i^2 . In the case that the ellipsoid has stretching and n shrinking directions, it will be very long and very thin for large n. The eigenvalues of $J_n J_n^T$ will include both very large and very small numbers. Because of the limited number of digits allowed for each stored number, computer calculations become difficult when numbers of vastly different sizes are involved in the same calculation. The problem of computing the s_i gets worse as n increases. For this reason, the direct calculation of the ellipsoid $J_n N$ is usually avoided.

The indirect approach that works better in numerical calculations involves following the ellipsoid as it grows. Since

$$J_n U = Df(v_{n-1}) \cdots Df(v_o) N \tag{5.17}$$

be can compute one iterate at a time. Start with an orthonormal basis $\{w_1^0, ..., w_m^0\}$ for $\mathbb{R}^m$, and compute the vectors $z_1, ..., z_m$:

$$z_1 = Df(v_0) w_1^0, ..., z_m = Df(v_0) w_m^0 \tag{5.18}$$

These vectors lie on the new ellipse $Df(v_0)N$, but they are not necessarily orthogonal. it will remedy this situation by creating a new set of orthogonal vectors $\{w_1^1, ..., w_m^1\}$ which generate an ellipsoid with the same volume as $Df(v_0)N$. Use the Gram-Schmidt orthogonalization procedure, which defines

$$\begin{aligned}
y_1 &= z_1 \\
y_2 &= z_2 - \frac{z_2 \cdot y_1}{|y_1|^2} y_1 \\
y_3 &= z_3 - \frac{z_3 \cdot y_1}{|y_1|^2} y_1 - \frac{z_3 \cdot y_2}{|y_2|^2} y_2 \\
&\vdots \\
y_m &= z_m - \frac{z_m \cdot y_1}{|y_1|^2} y_1 - ... - \frac{z_m \cdot y_{m-1}}{|y_{m-1}|^2} y_{m-1}
\end{aligned} \tag{5.19}$$

where $\cdot$ denotes the dot or scalar product and $|\cdot|$ denotes Euclidean length [10].

It used Lyapunov exponent calcullation for ODE-system. The algorithm employed in software for determining Lyapunov exponents was proposed by A. Wolf [10] and the function implemented in the Matlab software MATDS [12].

5.3.2 Gui

Figure 5.18 is explained:

- **Data entry panel:**

  dx/dt, dy/dt and dz/dt: are the function in terms of x and y representing the functions F_1, F_2 and F_3 of the equations system.

  x0, y0 and *z0:* are the initial conditions at time *tstart.*

  tstart: start values of independent value (time t).

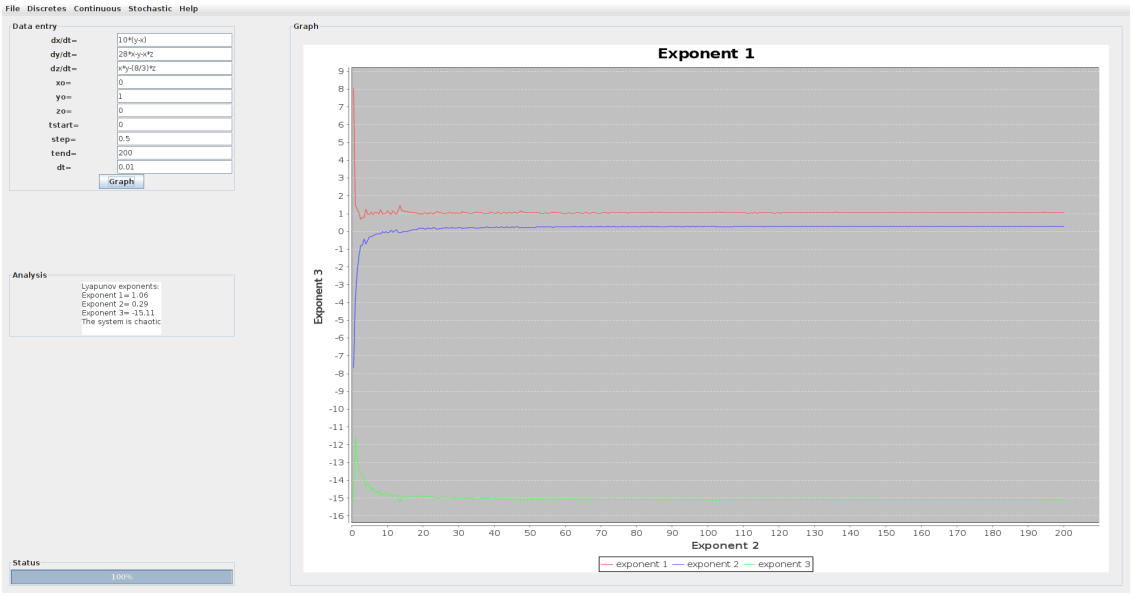


Figure 5.18: Lyapunov exponents of the system of Lorenz

step: step on t-variable for Gram-Schmidt renormalization procedure. Take as an initial value 0.5 and go setting.

tendn: finish value of time.

dt: is the step size (h) of Runge Kutta method. Take as an initial value 0.01 and go setting.

- **Graph panel:**

  Is where it looks the graph of the input data.

- **Analysis panel:**

  Display the Lyapunov exponents and description if the system is chaotic or stable according to definition *5.3.2.*

5.3.3 File menu: Save

To save must first to graph. It is saved in a flat file, TXT format, where the first column represent the time t, and the other columns represent the Lyapunov exponents of according to the number of system equations.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in Gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where first replaces the comma by point, and load file gnuplot.

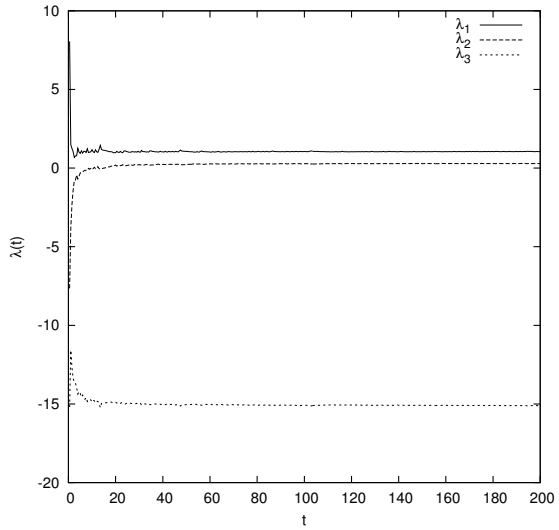


Figure 5.19: Lyapunov exponents using GNUPLOT

All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done as follows in linux: perl -p -i -e 's/,/./g' graph.txt

Script file:
#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot
load 'graph.gnuplot'
EOF

Gnuplot file:
set term postscript eps enhanced
set output "LyapunovExponents.eps"
set size square
set xlabel "t"
set ylabel "{/Symbol l}(t)"
plot "graph.txt" using 1:2 ti "{/Symbol l}_{1}" with lines,"graph.txt" using 1:3 ti "{/Symbol l}_{2}" with lines, "graph.txt" using 1:4 ti "{/Symbol l}_{3}" with lines

Once you save the file gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as showed in figure 5.19. The data are the same as those used in figure 5.18.

In Matlab and Octave it can plot the data from txt file, saving a file m format the following code:

```
clear all
close all
clc
X=load('graph.txt');
figure(1)
hold on
plot(X(:,1),X(:,2),'b')
plot(X(:,1),X(:,3),'g')
plot(X(:,1),X(:,4),'r')
title('Lyapunov exponents')
xlabel('t')
ylabel('\lambda')
legend('\lambda_1','\lambda_2','\lambda_3')
grid on
```

The figure 5.20 displays the graphs obtained in Matlab and Octave respectively. To save in Octave is used:

octave:5> print -color filename.eps

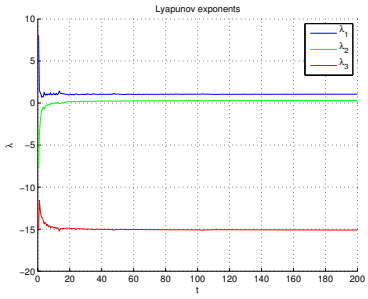
5.3.4 File menu: Export to Matlab

To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.

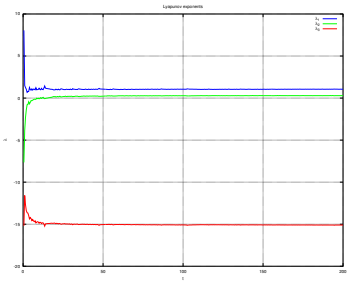
This is the code that is generated with the data showed in figure 5.18.

```
% DS Simulator
% Lyapunov exponents
function varargout = lyac(varargin)
run;
function run()
[T,Res]=lyapunov(3,0.0,0.5,200.0,[0.0 1.0 0.0], 0.01);
plot(T,Res) ;
title('Dynamics of Lyapunov exponents');
xlabel('Time'); ylabel('Lyapunov exponents');
function [Texp,Lexp]=lyapunov(n,tstart,tstept,tend,ystart,dt)
% Lyapunov exponent calcullation for ODE-system.
% The alogrithm employed in this m-file for determining Lyapunov

% exponents was proposed in % A. Wolf, J. B. Swift, H. L. Swinney,
and J. A. Vastano,
% "Determining Lyapunov Exponents from a Time Series," Physica D,
```



(a) Lyapunov exponents in MATLAB loading data



(b) Lyapunov exponents in Octave loading data

Figure 5.20: Lyapunov exponents in MATLAB and Octave loading data

```
   % Vol.  16, pp.  285-317, 1985.  % For integrating ODE system can
be used any MATLAB ODE-suite methods.
   % This function is a part of MATDS program - toolbox for dynamical
system investigation
   % See:  http://www.math.rsu.ru/mexmat/kvm/matds/
   % --------------------------------------------------------------------

   % Copyright (C) 2004, Govorukhin V.N.
   % This file is intended for use with MATLAB and was produced for
MATDS-program
   % http://www.math.rsu.ru/mexmat/kvm/matds/
   % lyapunov.m is free software.  lyapunov.m is distributed in the
hope that it
   % will be useful, but WITHOUT ANY WARRANTY.
   ioutp=10; n1=n; n2=n1*(n1+1);
   % Number of steps
   nit = round((tend-tstart)/tstept);
   % Memory allocation
```

```
y=zeros(n2,1); cum=zeros(n1,1); y0=y;
gsc=cum; znorm=cum;
% Initial values
y(1:n)=ystart(:);
for i=1:n1 y((n1+1)*i)=1.0;end;
t=tstart;
% Main loop
for ITERLYAP=1:nit
% Solutuion of extended ODE system
[T,Y]=RK4_12([tstart, tstept, tend],y,dt);
t=t+tstept;
y=Y(size(Y,1),:);
for i=1:n1
for j=1:n1
y0(n1*i+j)=y(n1*j+i);
end;
end;
% construct new orthonormal basis by gram-schmidt
znorm(1)=0.0;
for j=1:n1
znorm(1)=znorm(1)+y0(n1*j+1)^2;
end;
znorm(1)=sqrt(znorm(1));
for j=1:n1 y0(n1*j+1)=y0(n1*j+1)/znorm(1); end;
for j=2:n1
for k=1:(j-1)
gsc(k)=0.0;
for l=1:n1
gsc(k)=gsc(k)+y0(n1*l+j)*y0(n1*l+k); end;
end;
for k=1:n1
for l=1:(j-1) y0(n1*k+j)=y0(n1*k+j)-gsc(l)*y0(n1*k+l); end;
end;
znorm(j)=0.0;
for k=1:n1 znorm(j)=znorm(j)+y0(n1*k+j)^2; end;
znorm(j)=sqrt(znorm(j));
for k=1:n1 y0(n1*k+j)=y0(n1*k+j)/znorm(j); end;
end;
% update running vector magnitudes
for k=1:n1
cum(k)=cum(k)+log(znorm(k));
end;
% normalize exponent
for k=1:n1 lp(k)=cum(k)/(t-tstart); end;
% Output modification
if ITERLYAP==1
```

```
Lexp=lp;
Texp=t;
else
Lexp=[Lexp; lp];
Texp=[Texp; t];
end;
if (mod(ITERLYAP,ioutp)==0)
fprintf('t=%6.4f',t);
for k=1:n1 fprintf(' %10.6f',lp(k));
end;
fprintf('\n');
end;
for i=1:n1
for j=1:n1
y(n1*j+i)=y0(n1*i+j);
end;
end;
end;
function [T,X]=RK4_12(t,y,dt)
xk(1)=y(1);
yk(1)=y(2);
zk(1)=y(3);
o1(1)=y(4);
o2(1)=y(5);
o3(1)=y(6);
o4(1)=y(7);
o5(1)=y(8);
o6(1)=y(9);
o7(1)=y(10);
o8(1)=y(11);
o9(1)=y(12);
ni=(t(2)-t(1))/dt;
T(1)=t(1);
for i=1:ni
T(i+1)=T(i)+dt;
o=[o1(i) o4(i) o7(i); o2(i) o5(i) o8(i); o3(i) o6(i) o9(i)];
[f1,f2,f3,f4]=step_12(xk(i),yk(i),zk(i),T(i),o,dt);
xk(i+1)=xk(i)+(dt/6)*(f1(1)+2*(f2(1))+2*(f3(1))+f4(1));
yk(i+1)=yk(i)+(dt/6)*(f1(2)+2*(f2(2))+2*(f3(2))+f4(2));
zk(i+1)=zk(i)+(dt/6)*(f1(3)+2*(f2(3))+2*(f3(3))+f4(3));
o1(i+1)=o1(i)+(dt/6)*(f1(4)+2*(f2(4))+2*(f3(4))+f4(4));
o2(i+1)=o2(i)+(dt/6)*(f1(5)+2*(f2(5))+2*(f3(5))+f4(5));
o3(i+1)=o3(i)+(dt/6)*(f1(6)+2*(f2(6))+2*(f3(6))+f4(6));
o4(i+1)=o4(i)+(dt/6)*(f1(7)+2*(f2(7))+2*(f3(7))+f4(7));
o5(i+1)=o5(i)+(dt/6)*(f1(8)+2*(f2(8))+2*(f3(8))+f4(8));
o6(i+1)=o6(i)+(dt/6)*(f1(9)+2*(f2(9))+2*(f3(9))+f4(9));
```

```
o7(i+1)=o7(i)+(dt/6)*(f1(10)+2*(f2(10))+2*(f3(10))+f4(10));
o8(i+1)=o8(i)+(dt/6)*(f1(11)+2*(f2(11))+2*(f3(11))+f4(11));
o9(i+1)=o9(i)+(dt/6)*(f1(12)+2*(f2(12))+2*(f3(12))+f4(12));
end
X=[xk' yk' zk' o1' o2' o3' o4' o5' o6' o7' o8' o9'];
return
function [f1,f2,f3,f4]=step_12(xk,yk,zk,tk,o,h)
x=xk;
y=yk;
z=zk;
t=tk;
Jac=jacobian([xk,yk,zk],tk)*o;
f1(1)=10*(y-x);
f1(2)=28*x-y-x*z;
f1(3)=x*y-(8/3)*z;
f1(4)=Jac(1,1);
f1(5)=Jac(2,1);
f1(6)=Jac(3,1);
f1(7)=Jac(1,2);
f1(8)=Jac(2,2);
f1(9)=Jac(3,2);
f1(10)=Jac(1,3);
f1(11)=Jac(2,3);
f1(12)=Jac(3,3);
x=xk+(h/2);
y=yk+(h/2);
z=zk+h*(f1(3))/2;
Jac=jacobian([x,y,z],t)*o;
f2(1)=10*(y-x);
f2(2)=28*x-y-x*z;
f2(3)=x*y-(8/3)*z;
f2(4)=Jac(1,1);
f2(5)=Jac(2,1);
f2(6)=Jac(3,1);
f2(7)=Jac(1,2);
f2(8)=Jac(2,2);
f2(9)=Jac(3,2);
f2(10)=Jac(1,3);
f2(11)=Jac(2,3);
f2(12)=Jac(3,3);
x=xk+(h/2);
y=yk+(h/2);
z=zk+h*(f2(3))/2;
Jac=jacobian([x,y,z],t)*o;
f3(1)=10*(y-x);
f3(2)=28*x-y-x*z;
```

```
f3(3)=x*y-(8/3)*z;
f3(4)=Jac(1,1);
f3(5)=Jac(2,1);
f3(6)=Jac(3,1);
f3(7)=Jac(1,2);
f3(8)=Jac(2,2);
f3(9)=Jac(3,2);
f3(10)=Jac(1,3);
f3(11)=Jac(2,3);
f3(12)=Jac(3,3);
x=xk+h;
y=yk+h;
z=zk+h*(f3(3));
Jac=jacobian([x,y,z],t)*o;
f4(1)=10*(y-x);
f4(2)=28*x-y-x*z;
f4(3)=x*y-(8/3)*z;
f4(4)=Jac(1,1);
f4(5)=Jac(2,1);
f4(6)=Jac(3,1);
f4(7)=Jac(1,2);
f4(8)=Jac(2,2);
f4(9)=Jac(3,2);
f4(10)=Jac(1,3);
f4(11)=Jac(2,3);
f4(12)=Jac(3,3);
return
function J=jacobian(yo,to)
h=0.00001;
sum=0;
x=yo(1)+2*h;
y=yo(2);
z=yo(3);
t=to;
sum=sum+(-1*(10*(y-x)));
x=yo(1)+h;
sum=sum+8*(10*(y-x));
x=yo(1)-h;
sum=sum-8*(10*(y-x));
x=yo(1)-2*h;
sum=sum+10*(y-x);
pdfx=sum/(12*h);
sum=0;
y=yo(2)+2*h;
x=yo(1);
z=yo(3);
```

```
t=to;
sum=sum+(-1*(10*(y-x)));
y=yo(2)+h;
sum=sum+8*(10*(y-x));
y=yo(2)-h;
sum=sum-8*(10*(y-x));
y=yo(2)-2*h;
sum=sum+10*(y-x);
pdfy=sum/(12*h);
sum=0;
z=yo(3)+2*h;
y=yo(2);
x=yo(1);
t=to;
sum=sum+(-1*(10*(y-x)));
z=yo(3)+h;
sum=sum+8*(10*(y-x));
z=yo(3)-h;
sum=sum-8*(10*(y-x));
z=yo(3)-2*h;
sum=sum+10*(y-x);
pdfz=sum/(12*h);
sum=0;
x=yo(1)+2*h;
y=yo(2);
z=yo(3);
t=to;
sum=sum+(-1*(28*x-y-x*z));
x=yo(1)+h;
sum=sum+8*(28*x-y-x*z);
x=yo(1)-h;
sum=sum-8*(28*x-y-x*z);
x=yo(1)-2*h;
sum=sum+28*x-y-x*z;
pdgx=sum/(12*h);
sum=0;
y=yo(2)+2*h;
x=yo(1);
z=yo(3);
t=to;
sum=sum+(-1*(28*x-y-x*z));
y=yo(2)+h;
sum=sum+8*(28*x-y-x*z);
y=yo(2)-h;
sum=sum-8*(28*x-y-x*z);
y=yo(2)-2*h;
```

```
sum=sum+28*x-y-x*z;
pdgy=sum/(12*h);
sum=0;
z=yo(3)+2*h;
y=yo(2);
x=yo(1);
t=to;
sum=sum+(-1*(28*x-y-x*z));
z=yo(3)+h;
sum=sum+8*(28*x-y-x*z);
z=yo(3)-h;
sum=sum-8*(28*x-y-x*z);
z=yo(3)-2*h;
sum=sum+28*x-y-x*z;
pdgz=sum/(12*h);
sum=0;
x=yo(1)+2*h;
y=yo(2);
z=yo(3);
t=to;
sum=sum+(-1*(x*y-(8/3)*z));
x=yo(1)+h;
sum=sum+8*(x*y-(8/3)*z);
x=yo(1)-h;
sum=sum-8*(x*y-(8/3)*z);
x=yo(1)-2*h;
sum=sum+x*y-(8/3)*z;
pdhx=sum/(12*h);
sum=0;
y=yo(2)+2*h;
x=yo(1);
z=yo(3);
t=to;
sum=sum+(-1*(x*y-(8/3)*z));
y=yo(2)+h;
sum=sum+8*(x*y-(8/3)*z);
y=yo(2)-h;
sum=sum-8*(x*y-(8/3)*z);
y=yo(2)-2*h;
sum=sum+x*y-(8/3)*z;
pdhy=sum/(12*h);
sum=0;
z=yo(3)+2*h;
y=yo(2);
x=yo(1);
t=to;
```

```
sum=sum+(-1*(x*y-(8/3)*z));
z=yo(3)+h;
sum=sum+8*(x*y-(8/3)*z);
z=yo(3)-h;
sum=sum-8*(x*y-(8/3)*z);
z=yo(3)-2*h;
sum=sum+x*y-(8/3)*z;
pdhz=sum/(12*h);
J=[pdfx,pdfy,pdfz;pdgx,pdgy,pdgz;pdhx,pdhy,pdhz];
return
```

Figure 5.21a shows the result.

5.3.5 File menu: Export to Octave

To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

This is the code that is generated with the data showed in figure 5.18.

```
% DS Simulator
% Lyapunov exponents
function varargout = lyac(varargin)
run;
function run()
[T,Res]=lyapunov(3,0.0,0.5,200.0,[0.0 1.0 0.0], 0.01);
plot(T,Res) ;
title('Dynamics of Lyapunov exponents');
xlabel('Time'); ylabel('Lyapunov exponents');
function [Texp,Lexp]=lyapunov(n,tstart,tstept,tend,ystart,dt)
% Lyapunov exponent calcullation for ODE-system.
% The alogrithm employed in this m-file for determining Lyapunov

% exponents was proposed in
% A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano,
% "Determining Lyapunov Exponents from a Time Series," Physica D,

% Vol.  16, pp.  285-317, 1985.
% For integrating ODE system can be used any MATLAB ODE-suite methods.

% This function is a part of MATDS program - toolbox for dynamical
system investigation
% See:  http://www.math.rsu.ru/mexmat/kvm/matds/
% --------------------------------------------------------------------

% Copyright (C) 2004, Govorukhin V.N.
% This file is intended for use with MATLAB and was produced for
MATDS-program
% http://www.math.rsu.ru/mexmat/kvm/matds/
```

```
   % lyapunov.m is free software.  lyapunov.m is distributed in the
hope that it
   % will be useful, but WITHOUT ANY WARRANTY.
   ioutp=10;
   n1=n;
   n2=n1*(n1+1);
   % Number of steps
   nit = round((tend-tstart)/tstept);
   % Memory allocation
   y=zeros(n2,1);
   cum=zeros(n1,1);
   y0=y; gsc=cum; znorm=cum;
   % Initial values
   y(1:n)=ystart(:);
   for i=1:n1 y((n1+1)*i)=1.0; end;
   t=tstart;
   % Main loop
   for ITERLYAP=1:nit
   % Solutuion of extended ODE system
   [T,Y]=RK4_12([tstart, tstept, tend],y,dt);
   t=t+tstept;
   y=Y(size(Y,1),:);
   for i=1:n1
   for j=1:n1
   y0(n1*i+j)=y(n1*j+i);
   endfor;
   endfor;
   % construct new orthonormal basis by gram-schmidt
   znorm(1)=0.0;
   for j=1:n1 znorm(1)=znorm(1)+y0(n1*j+1)^2; endfor;
   znorm(1)=sqrt(znorm(1));
   for j=1:n1 y0(n1*j+1)=y0(n1*j+1)/znorm(1); endfor;
   for j=2:n1
   for k=1:(j-1)
   gsc(k)=0.0;
   for l=1:n1 gsc(k)=gsc(k)+y0(n1*l+j)*y0(n1*l+k); endfor;
   endfor;
   for k=1:n1 for l=1:(j-1) y0(n1*k+j)=y0(n1*k+j)-gsc(l)*y0(n1*k+l);
endfor;
   endfor;
   znorm(j)=0.0;
   for k=1:n1 znorm(j)=znorm(j)+y0(n1*k+j)^2; endfor;
   znorm(j)=sqrt(znorm(j));
   for k=1:n1 y0(n1*k+j)=y0(n1*k+j)/znorm(j); endfor;
   endfor;
   % update running vector magnitudes
```

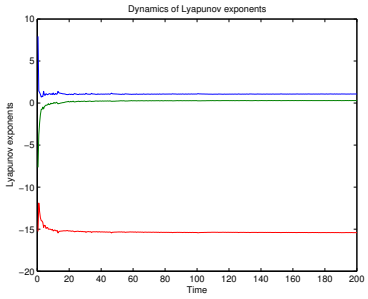
```
for k=1:n1 cum(k)=cum(k)+log(znorm(k)); endfor;
% normalize exponent
for k=1:n1 lp(k)=cum(k)/(t-tstart); endfor;
% Output modification
if ITERLYAP==1
Lexp=lp;
Texp=t;
else
Lexp=[Lexp; lp];
Texp=[Texp; t];
end;
if (mod(ITERLYAP,ioutp)==0)
fprintf('t=%6.4f',t);
for k=1:n1
fprintf(' %10.6f',lp(k));
endfor;
fprintf('\n');
end;
for i=1:n1
for j=1:n1
y(n1*j+i)=y0(n1*i+j);
endfor;
endfor;
endfor;
function [T,X]=RK4_12(t,y,dt)
xk(1)=y(1);
yk(1)=y(2);
zk(1)=y(3);
o1(1)=y(4);
o2(1)=y(5);
o3(1)=y(6);
o4(1)=y(7);
o5(1)=y(8);
o6(1)=y(9);
o7(1)=y(10);
o8(1)=y(11);
o9(1)=y(12);
ni=(t(2)-t(1))/dt;
T(1)=t(1);
for i=1:ni
T(i+1)=T(i)+dt;
o=[o1(i) o4(i) o7(i); o2(i) o5(i) o8(i); o3(i) o6(i) o9(i)];
[f1,f2,f3,f4]=step_12(xk(i),yk(i),zk(i),T(i),o,dt);
xk(i+1)=xk(i)+(dt/6)*(f1(1)+2*(f2(1))+2*(f3(1))+f4(1));
yk(i+1)=yk(i)+(dt/6)*(f1(2)+2*(f2(2))+2*(f3(2))+f4(2));
zk(i+1)=zk(i)+(dt/6)*(f1(3)+2*(f2(3))+2*(f3(3))+f4(3));
```

```
o1(i+1)=o1(i)+(dt/6)*(f1(4)+2*(f2(4))+2*(f3(4))+f4(4));
o2(i+1)=o2(i)+(dt/6)*(f1(5)+2*(f2(5))+2*(f3(5))+f4(5));
o3(i+1)=o3(i)+(dt/6)*(f1(6)+2*(f2(6))+2*(f3(6))+f4(6));
o4(i+1)=o4(i)+(dt/6)*(f1(7)+2*(f2(7))+2*(f3(7))+f4(7));
o5(i+1)=o5(i)+(dt/6)*(f1(8)+2*(f2(8))+2*(f3(8))+f4(8));
o6(i+1)=o6(i)+(dt/6)*(f1(9)+2*(f2(9))+2*(f3(9))+f4(9));
o7(i+1)=o7(i)+(dt/6)*(f1(10)+2*(f2(10))+2*(f3(10))+f4(10));
o8(i+1)=o8(i)+(dt/6)*(f1(11)+2*(f2(11))+2*(f3(11))+f4(11));
o9(i+1)=o9(i)+(dt/6)*(f1(12)+2*(f2(12))+2*(f3(12))+f4(12));
endfor
X=[xk' yk' zk' o1' o2' o3' o4' o5' o6' o7' o8' o9'];
return
function [f1,f2,f3,f4]=step_12(xk,yk,zk,tk,o,h)
x=xk;
y=yk;
z=zk;
t=tk;
Jac=jacobian([xk,yk,zk],tk)*o;
f1(1)=10*(y-x);
f1(2)=28*x-y-x*z;
f1(3)=x*y-(8/3)*z;
f1(4)=Jac(1,1);
f1(5)=Jac(2,1);
f1(6)=Jac(3,1);
f1(7)=Jac(1,2);
f1(8)=Jac(2,2);
f1(9)=Jac(3,2);
f1(10)=Jac(1,3);
f1(11)=Jac(2,3);
f1(12)=Jac(3,3);
x=xk+(h/2);
y=yk+(h/2);
z=zk+h*(f1(3))/2;
Jac=jacobian([x,y,z],t)*o;
f2(1)=10*(y-x);
f2(2)=28*x-y-x*z;
f2(3)=x*y-(8/3)*z;
f2(4)=Jac(1,1);
f2(5)=Jac(2,1);
f2(6)=Jac(3,1);
f2(7)=Jac(1,2);
f2(8)=Jac(2,2);
f2(9)=Jac(3,2);
f2(10)=Jac(1,3);
f2(11)=Jac(2,3);
f2(12)=Jac(3,3);
```

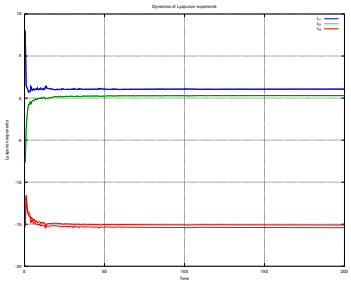
```
x=xk+(h/2);
y=yk+(h/2);
z=zk+h*(f2(3))/2;
Jac=jacobian([x,y,z],t)*o;
f3(1)=10*(y-x);
f3(2)=28*x-y-x*z;
f3(3)=x*y-(8/3)*z;
f3(4)=Jac(1,1);
f3(5)=Jac(2,1);
f3(6)=Jac(3,1);
f3(7)=Jac(1,2);
f3(8)=Jac(2,2);
f3(9)=Jac(3,2);
f3(10)=Jac(1,3);
f3(11)=Jac(2,3);
f3(12)=Jac(3,3);
x=xk+h;
y=yk+h;
z=zk+h*(f3(3));
Jac=jacobian([x,y,z],t)*o;
f4(1)=10*(y-x);
f4(2)=28*x-y-x*z;
f4(3)=x*y-(8/3)*z;
f4(4)=Jac(1,1);
f4(5)=Jac(2,1);
f4(6)=Jac(3,1);
f4(7)=Jac(1,2);
f4(8)=Jac(2,2);
f4(9)=Jac(3,2);
f4(10)=Jac(1,3);
f4(11)=Jac(2,3);
f4(12)=Jac(3,3);
return
function J=jacobian(yo,to)
h=0.00001;
sum=0;
x=yo(1)+2*h;
y=yo(2);
z=yo(3);
t=to;
sum=sum+(-1*(10*(y-x)));
x=yo(1)+h;
sum=sum+8*(10*(y-x));
x=yo(1)-h;
sum=sum-8*(10*(y-x));
x=yo(1)-2*h;
```

```
sum=sum+10*(y-x);
pdfx=sum/(12*h);
sum=0;
y=yo(2)+2*h;
x=yo(1);
z=yo(3);
t=to;
sum=sum+(-1*(10*(y-x)));
y=yo(2)+h;
sum=sum+8*(10*(y-x));
y=yo(2)-h;
sum=sum-8*(10*(y-x));
y=yo(2)-2*h;
sum=sum+10*(y-x);
pdfy=sum/(12*h);
sum=0;
z=yo(3)+2*h;
y=yo(2);
x=yo(1);
t=to;
sum=sum+(-1*(10*(y-x)));
z=yo(3)+h;
sum=sum+8*(10*(y-x));
z=yo(3)-h;
sum=sum-8*(10*(y-x));
z=yo(3)-2*h;
sum=sum+10*(y-x);
pdfz=sum/(12*h);
sum=0;
x=yo(1)+2*h;
y=yo(2);
z=yo(3);
t=to;
sum=sum+(-1*(28*x-y-x*z));
x=yo(1)+h;
sum=sum+8*(28*x-y-x*z);
x=yo(1)-h;
sum=sum-8*(28*x-y-x*z);
x=yo(1)-2*h;
sum=sum+28*x-y-x*z;
pdgx=sum/(12*h);
sum=0;
y=yo(2)+2*h;
x=yo(1);
z=yo(3);
t=to;
```

```
sum=sum+(-1*(28*x-y-x*z));
y=yo(2)+h;
sum=sum+8*(28*x-y-x*z);
y=yo(2)-h;
sum=sum-8*(28*x-y-x*z);
y=yo(2)-2*h;
sum=sum+28*x-y-x*z;
pdgy=sum/(12*h);
sum=0;
z=yo(3)+2*h;
y=yo(2);
x=yo(1);
t=to;
sum=sum+(-1*(28*x-y-x*z));
z=yo(3)+h;
sum=sum+8*(28*x-y-x*z);
z=yo(3)-h;
sum=sum-8*(28*x-y-x*z);
z=yo(3)-2*h;
sum=sum+28*x-y-x*z;
pdgz=sum/(12*h);
sum=0;
x=yo(1)+2*h;
y=yo(2);
z=yo(3);
t=to;
sum=sum+(-1*(x*y-(8/3)*z));
x=yo(1)+h;
sum=sum+8*(x*y-(8/3)*z);
x=yo(1)-h;
sum=sum-8*(x*y-(8/3)*z);
x=yo(1)-2*h;
sum=sum+x*y-(8/3)*z;
pdhx=sum/(12*h);
sum=0;
y=yo(2)+2*h;
x=yo(1);
z=yo(3);
t=to;
sum=sum+(-1*(x*y-(8/3)*z));
y=yo(2)+h;
sum=sum+8*(x*y-(8/3)*z);
y=yo(2)-h;
sum=sum-8*(x*y-(8/3)*z);
y=yo(2)-2*h;
sum=sum+x*y-(8/3)*z;
```



(a) Lyapunov exponents exporting code to MATLAB



(b) Lyapunov exponents exporting code to Octave

Figure 5.21: Code exported to Matlab and Octave from Lyapunov exponents

```
pdhy=sum/(12*h);
sum=0;
z=yo(3)+2*h;
y=yo(2);
x=yo(1);
t=to;
sum=sum+(-1*(x*y-(8/3)*z));
z=yo(3)+h;
sum=sum+8*(x*y-(8/3)*z);
z=yo(3)-h;
sum=sum-8*(x*y-(8/3)*z);
z=yo(3)-2*h;
sum=sum+x*y-(8/3)*z;
pdhz=sum/(12*h);
J=[pdfx,pdfy,pdfz;pdgx,pdgy,pdgz;pdhx,pdhy,pdhz];
return
```

Figure 5.21b shows the result.

5.4 Hamiltonian systems

Trajectories of conservative systems are constrained to one energy level. In other words, energy remains constant along a solution. The set of equations that govern the motion of the pendulum is one example of a general class of differential equations called Hamiltonian systems. Specifically, for a smooth function $H : \mathbb{R}^2 \to \mathbb{R}$ the *Hamiltonian function* or *energy function*, Hamilton's equations are [10]:

$$\begin{aligned} x' &= \frac{\partial H}{\partial y}(x, y) \\ y' &= -\frac{\partial H}{\partial x}(x, y) \end{aligned} \tag{5.20}$$

where $H : \mathbb{R}^2 \to \mathbb{R}$ is a C^∞.

What makes Hamiltonian systems so important is the fact that the Hamiltonian function is a first integral or constant of the motion. That is, H is constant along every solution of the systems, or $H \equiv 0$.

Proposition 5.4.1 *For a Hamiltonian system on* $\mathbb{R}^2$ *, H is constant along every solution curve.*

The importance of knowing that a given system is Hamiltonian is the fact that we can essentially draw the phase portrait without solving the system. Assuming that H is not constant on any open set, we simply plot the level curves $H(x, y) = constant$. The solutions of the system lie on these level sets; all we need to do is figure out the directions of the solution curves on these level sets. But this is easy since we have the vector field. Note also that the equilibrium points for a Hamiltonian system occur at the critical points of H , that is, at points where both partial derivatives of Hvanish [3].

Let $f = \dfrac{dx}{dt}$ and $g = \dfrac{dy}{dt}$ functions of a system, you can check if the system is conservative by the following equation

$$\frac{\partial f}{\partial x} = \frac{\partial^2 H}{\partial x \partial y} = \frac{\partial^2 H}{\partial y \partial x} = \frac{-\partial g}{\partial y} \tag{5.21}$$

5.4.1 Gui

Figure 5.22 is explained:

- **Data entry panel:**

  f(x,y) and g(x,y): are the function in terms of x and y representing the functions $\dfrac{dx}{dt}$ and $\dfrac{dy}{dt}$ of the equations system.

  $I_x = [x.li, x.ls]$ is the interval of the system in the x-axis on the which is the Hamiltonian function.

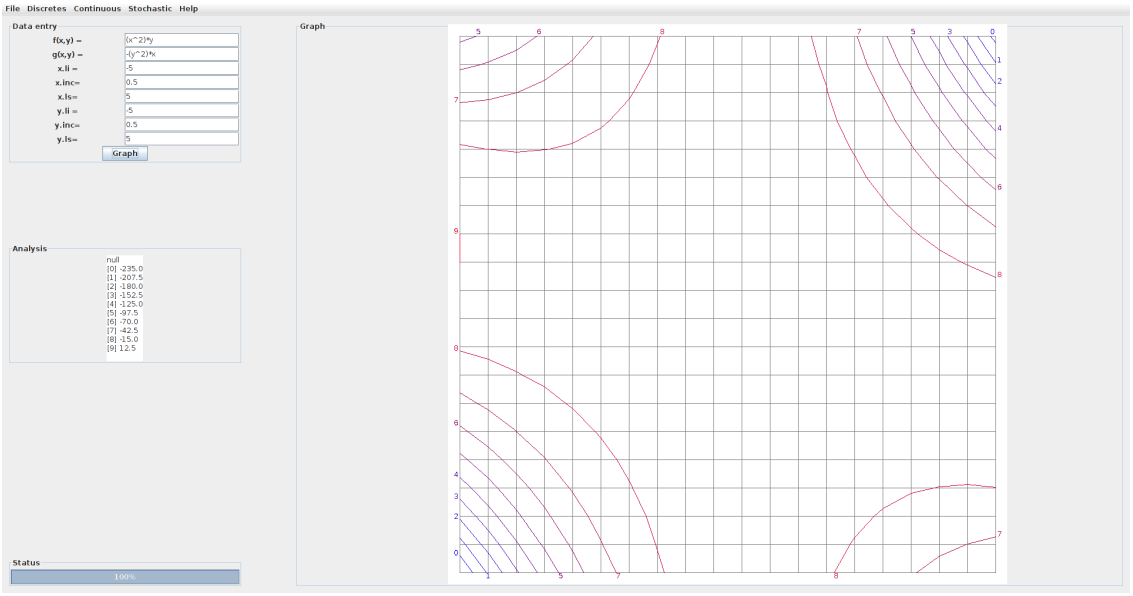


Figure 5.22: Hamiltonian system

x.inc: is the difference in the arithmetic progression which is the interval I_x. Takes a value of 0.3 and you adjust.

$I_y = [x.li, x.ls]$ is the interval of the system in the y-axis on the which is observed the Hamiltonian function.

y.inc: is the difference in the arithmetic progression which is the interval I_y. Takes a value of 0.3 and you adjust.

- **Analysis panel:**

  Display the levels of contour lines of the Hamiltonian system and if is consertive.

5.4.2 File menu: Save

To save must first to graph. 2 files are saved in a flat file, TXT format, where the first contains the Hamiltonian function and the second file a matrix of values of the Hamiltonian function.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in Gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where first replaces the comma by point, and load file gnuplot.

All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done

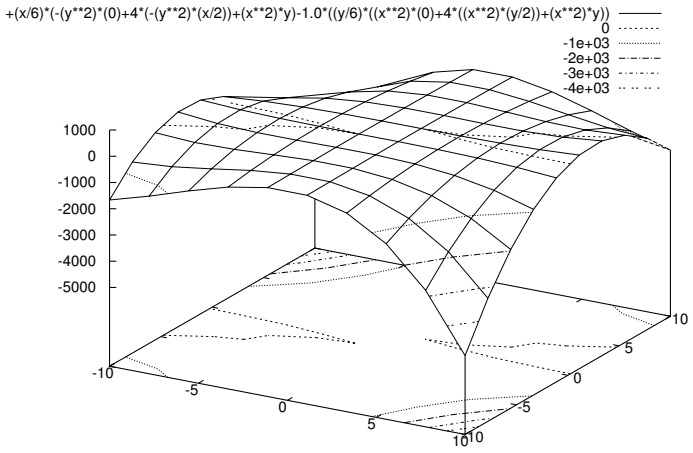


Figure 5.23: Lyapunov exponents using GNUPLOT

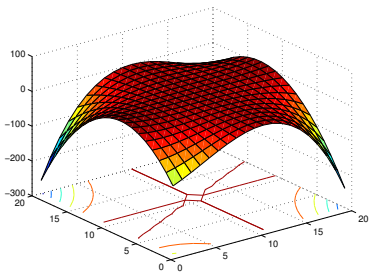
as follows in linux: perl -p -i -e 's/,/./g' graph.txt. You should also replace the character ^ by ** to plot in Gnuplot.

Script file:
#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot
load 'graph.gnuplot'
EOF

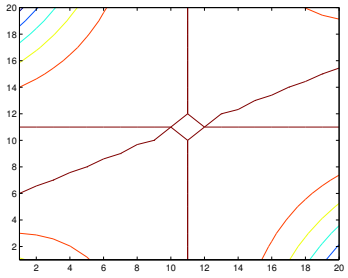
Gnuplot file:
set term postscript eps enhanced
set output "HamiltonSystem.eps"
set hidden3d
set contour both
splot (y/6)*((x**2)*(0)+4*((x**2)*(y/2))+(x**2)*y)+(x/6)*(-(y**2)*(0)+4*
(-(y**2)*(x/2))+(x**2)*y)-1.0*((y/6)*((x**2)*(0)+4*((x**2)*(y/2))+(x**2)*y))

Once you save the file gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as showed in figure 5.23. The data are the same as those used in figure 5.22.

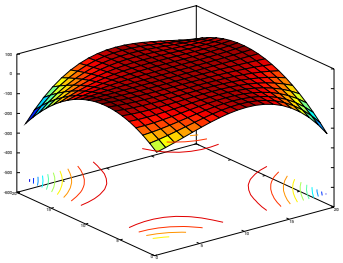
In Matlab and Octave it can plot the data from txt file, saving a file m format the following code:



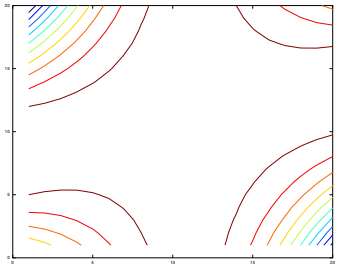
(a) Surface of Hamiltonian System in MATLAB loading data



(b) Contour of Hamiltonian system in Matlab loading data



(c) Surface of Hamiltonian System in Octave loading data



(d) Contour of Hamiltonian System in Octave loading data

Figura 5.24: Hamiltonian System in MATLAB and Octave loading data

```
clear all
close all
clc
H=load('graph.txt');
figure(1)
surfc(H)
figure(2)
contour(H)
```

The figure 5.24 displays the graphs obtained in Matlab and Octave respectively. To save in Octave is used:

octave:5> print -color filename.eps

5.4.3 File menu: Export to Matlab

To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.

This is the code that is generated with the data showed in figure 5.22.

```
% DS Simulator
% Hamiltonian system
clear all
close all
clc
syms x y a b c d e h t
f=(x^2)*y;
g=-(y^2)*x;
%% Hamiltonian function H(x,y)
if diff(f,x)==-diff(g,y)
% System is conservativ
ify=int(f,y);
fi=-g-diff(ify,x);
if fi==0
cx=0;
else
cx=int(fi,x);
end
H=ify+cx;
figure(1)
ezcontour(H);
figure(2)
ezsurf(H)
figure(3)
ezsurfc(H)
else
disp('System no conservativ')
end
```

Figure 5.25 shows the result.

5.4.4 File menu: Export to Octave

To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

This is the code that is generated with the data showed in figure 5.22.

```
% DS Simulator
% Hamiltonian system
function varargout = Hamiltonian(varargin)
run;
function run()
if hamilts()==true
h=fhamiltonian();
figure(1)
contour(h);
xlabel('x');
```

```
ylabel('y');
figure(2)
surfc(h)
else
disp('no conservativ');
end
function H=hamiltonian(x, y)
%integral use simpson
H=(y/6)*((x^2)*(0)+4*((x^2)*(y/2))+(x^2)*y)+(x/6)*(-(y^2)*(0)+4*
(-(y^2)*(x/2))+(x^2)*y)-1.0*((y/6)*((x^2)*(0)+4*((x^2)*(y/2))+(x^2)*y));

return
function res=hamilts()
err=1e-6;
c=0;
h=0.00001;
x(1)=-5.0;
y(1)=-5.0;
xinc=0.5;
yinc=0.5;
n=20;
for i=1:n-1
x(i+1)=x(i)+xinc;
y(i+1)=y(i)+yinc;
if (pdiffx(x(i), y(i))+pdiffy(x(i), y(i)))<err
c=0;
else
c=1;
break;
end
endfor
if (c==1)
res= false;
else
res= true;
end
return
function df=pdiffx(xx,y)
df=0;
t=0;
h=0.00001;
x=xx+2*h;
df=-1*((x^2)*y);
x=xx+h;
df=df+8*((x^2)*y);
x=xx-h;
```

```
df=df-8*((x^2)*y);
x=xx-2*h;
df=df+(x^2)*y;
df=df/(12);
return
function df=pdiffy(x,yy)
df=0;
t=0;
h=0.00001;
y=yy+2*h;
df=-1*(-(y^2)*x);
y=yy+h;
df=df+8*(-(y^2)*x);
y=yy-h;
df=df-8*(-(y^2)*x);
y=yy-2*h;
df=df+-(y^2)*x;
df=df/(12);
return
function H=fhamiltonian()
[x,y]=meshgrid(-5.0:0.5:5.0,-5.0:0.5:5.0);
[m,n]=size(x);
for i=1:m
for j=1:n
H(i,j)=hamiltonian(x(i,j),y(i,j));
endfor
endfor
return
```

Figure 5.26 shows the result.

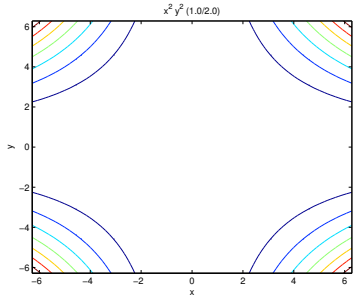
5.5 Equilibrium points

To find equilibrium points, is used Newton-Raphson method for nonlinear Systems of Equations.

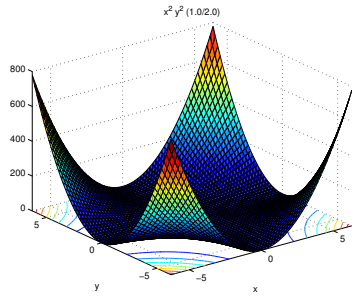
Consider the case of two dimensions, where it wanted to solve simultaneously

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned} \tag{5.22}$$

The functions f and g are two arbitrary functions, each of which has zero contour lines that divide the (x,y) plane into regions where their respective function is positive or negative. These zero contour boundaries are of interest to us. The solutions that we seek are those points (if any) that are common to the zero contours of f and g (see figure 5.27). Unfortunately, the functions f and g have, in general, no relation to each other at all. There is nothing special about a common point from either f's point of view, or from g's. In order to



(a) Contour of Hamiltonian system exporting code to MATLAB



(b) Surf of Hamiltonian system exporting code to Matlab

Figure 5.25: Code exported to Matlab from Hamiltonian system

find all common points, which are the solutions of our nonlinear equations, it will (in general) have to do neither more nor less than map out the full zero contours of both functions. Note further that the zero contours will (in general) consist of an unknown number of disjoint closed curves.

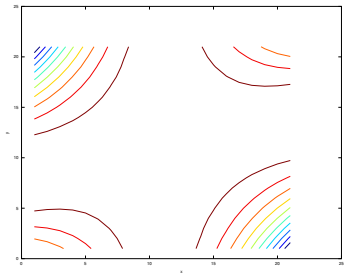
For problems in more than two dimensions, it need to find points mutually common to N unrelated zero-contour hypersurfaces, each of dimension $N-1$

This method gives you a very efficient means of converging to a root, if you have a sufficiently good initial guess. It can also spectacularly fail to converge, indicating (though not proving) that your putative root does not exist nearby.

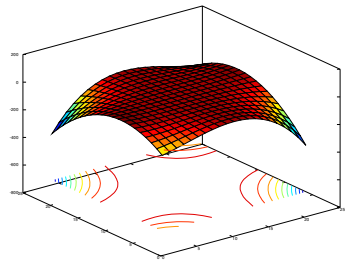
A typical problem gives N functional relations to be zeroed, involving variables $x_i, i = 1, 2, ..., N$:

$$F_i(x_1, x_2, ..., x_N) = 0 \quad i = 1, 2, ..., N \tag{5.23}$$

It let x denote the entire vector of values x_i and F denote the entire vector of functions F_i . In the neighborhood of x, each of the functions F_i can be expanded in Taylor series



(a) Contour of Hamiltonian system exporting code to Octave



(b) Surf of Hamiltonian system exporting code to Octave

Figure 5.26: Code exported to Octave from Hamiltonian system

$$F_i(x+\delta x) = F_i(x) + \sum_{j=1}^{N} \frac{\partial F_i}{\partial x_j}\delta x_j + O(\delta x^2) \tag{5.24}$$

The matrix of partial derivatives appearing in equation 5.24 is the Jacobian matrix J:

$$J_{ij} \equiv \frac{\partial F_i}{\partial x_j} \tag{5.25}$$

In matrix notation equation 5.24 is

$$F(x+\delta x) = F(x) + J \cdot \delta x + O(dx^2) \tag{5.26}$$

By neglecting terms of order δx^2 and higher and by setting $F(x+\delta x) = 0$, we obtain a set of linear equations for the corrections δx that move each function closer to zero simultaneously, namely

$$J \cdot \delta x = -F \tag{5.27}$$

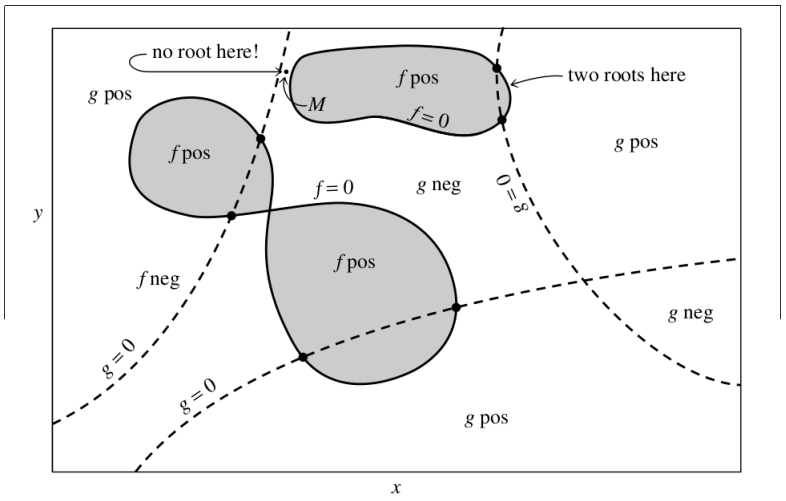


Figure 5.27: Solution of two nonlinear equations in two unknowns. Solid curves refer to $f(x, y)$, dashed curves to $g(x, y)$. Each equation divides the (x, y) plane into positive and negative regions, bounded by zero curves. The desired solutions are the intersections of these unrelated zero curves. The number of solutions is a priori unknown.

Matrix equation 5.27 can be solved by LU decomposition. The corrections are then added to the solution vector,

$$x_{new} = x_{old} + \delta x \tag{5.28}$$

and the process is iterated to convergence. In general it is a good idea to check the degree to which both functions and variables have converged. Once either reaches machine accuracy, the other won't change [9].

5.5.1 Gui

Figure 5.28 is explained:

- **Data entry panel:**

  f(x,y) and *g(x,y):* are the function in terms of x and y representing the functions of the equations system to find equilibrium points.

  x0 and *y0*: are the inital aproximation to the solution.

  delta: is the tolerance for x0 and y0.

  epsilon: is the tolerance for f(x0,y0) and g(x0,y0).

  max: is the maximum number of iterations requerid.

- **Analysis panel:**

  Displays the equilibrium point found near the initial point (x0,y0).

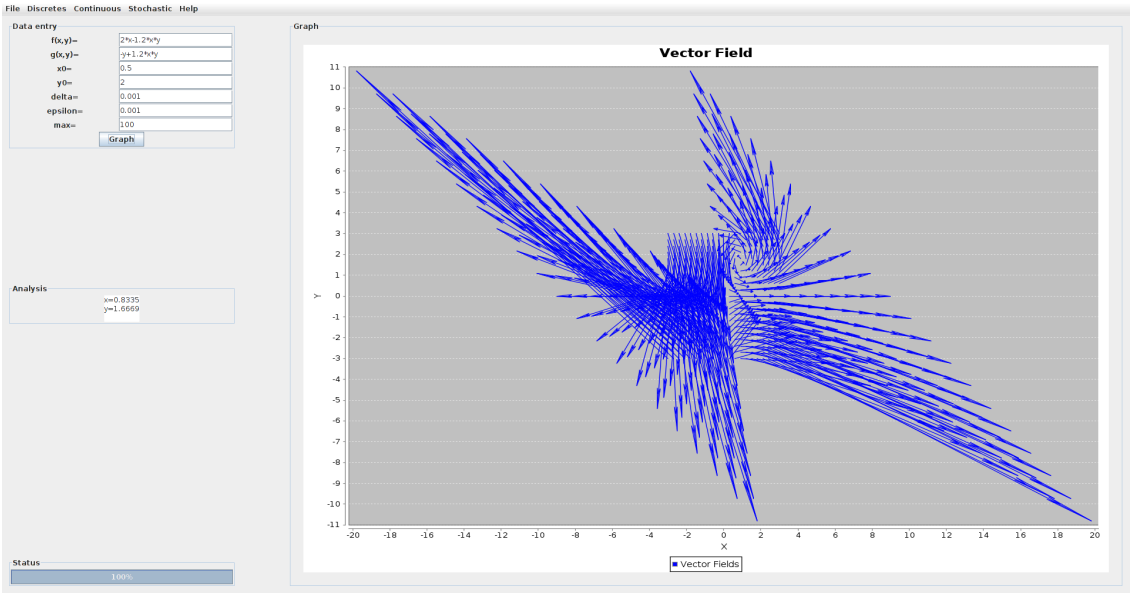


Figure 5.28: Equilibrium points of a predator-prey system

Chapter 6

STOCHASTIC DYNAMICAL SYSTEMS

A stochastic dynamical system is a dynamical system subjected to the effects of noise. Such effects of fluctuations have been of interest for over a century since the seminal work of Einstein (1905). Fluctuations are classically referred to as "noisy" or "stochastic" when their suspected origin implicates the action of a very large number of variables or "degrees of freedom". For example, the action of many water molecules on the motion of a large protein can be seen as noise. In principle the equations of motion for such high-dimensional dynamics can be written and studied analytically and numerically. However, it is possible to study a system subjected to the action of the large number of variables by coupling its deterministic equations of motion to a "noise" that simple mimics the perpetual action of many variables.

One of the difficulties with modeling noise is that we may not have access to the noise variable itself, but rather, to a state variable perturbed by one or more sources of noise. Thus, one may have to make assumptions about the nature of the noise and its coupling to the dynamical state variables. The accuracy of these assumptions can later be assessed by looking at the agreement between the predictions of the resulting model and the experimental data.

Stochastic differential equations(SDE) have applications in many areas of scientific knowledge such as biology, chemistry, mechanics, microelectronics, economics and finance

- **Additive noise**

A classification exists according to how the noise and dynamical variable interact. If the coefficient of η in the evolution equation is independent of the state x of the system:

$$\frac{dx}{dt} = a(x; \langle \mu \rangle) + \eta(t) \tag{6.1}$$

the noise is said to be "additive". In other words, the noise is simply added to the deterministic part of the dynamics.

- **Multiplicative noise**

Alternatively, one can have multiplicative noise, for which the coefficient of the noise depends on the value of one or many state variables. In such a case, the evolution equation would take the form:

$$\frac{dx}{dt} = a(x; \langle \mu \rangle) + b(x)\eta(t) \tag{6.2}$$

Now the strength of the noise is dependent on $x(t)$, so if $b(x)$ is large at time t, the effect of noise will be large at that time. Note that the functions a and b depend on time by virtue of their dependence on the state variable $x(t)$ [13].

6.0.2 Euler-Maruyama Numerical Solution of Some Stochastic Functional Differential Equations

Given functions f and g , the stochastic process X(t) is a soluton of the stochastic system equations

$$dX(t) = f(X(t))dt + g(X(t))dW(t) \tag{6.3}$$

with $X(0) = X_0$ and $0 \leq t \leq T$. This equation can also be written as

$$X(t) = X(0) = \int_0^t f(X(s))ds + \int_0^t g(X(s))dW_s \tag{6.4}$$

If to discretize equation 6.4 in j subintervals, would have

$$X(\tau_j) = X(\tau_{j-1}) + \int_{\tau_{j-1}}^{\tau_j} f(X(s))ds + \int_{\tau_{j-1}}^{\tau_j} g(X(s))dW_s \tag{6.5}$$

The Euler-Maruyama method uses the following approximation for the integrals of the form $\int_a^b f(x)dx$

Let be f a continuous function on an interval, by definition it has:

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_{i-1}) \triangle_i x \tag{6.6}$$

If it is considered the partition $a = x_0 < x_1 < x_2 < ... < x_n = b$, with $\triangle_i x = x_i - x_{i-1}$ for $i = 1, 2, ..., n$ the approximation for the integral would

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f(x_{i-1}) \triangle_i x \tag{6.7}$$

Now, it takes a $\triangle_i x$ small enough, $f(x_{i-1}) \approx f(x_0)$. Then the right term in equation 6.7 is rewritten as

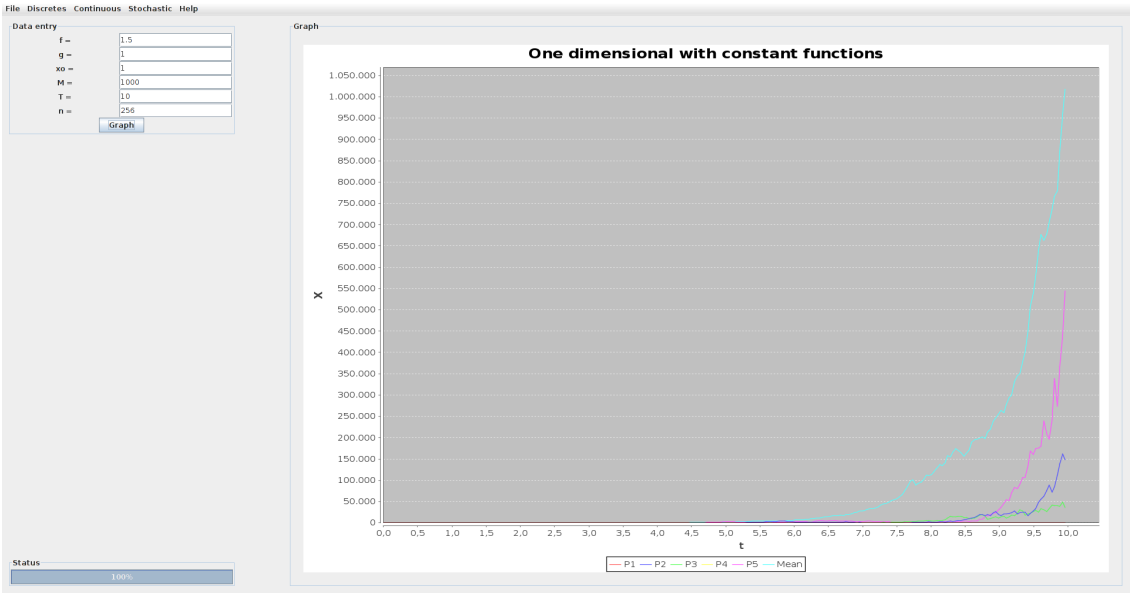


Figure 6.1: Stochastic dynamical system with constant functions

$$\begin{aligned}\sum_{i=1}^{n} f(x_{i-1})\triangle_i x &\approx f(x_0)\sum_{i=1}^{n}\triangle_i x \\ &= f(x_0)[(x_1-x_0)+(x_2-x_1)+...+(x_n-x_{n-1})] \\ &= f(x_0)(x_n-x_0) \\ &= f(a)(b-a)\end{aligned}$$

This analysis can be extended to stochastic integrals, in this case the integrals of equation 6.5, assuming a uniform partition where $b-a=\tau_j-\tau_{j-1}=\triangle t$ and it used the previous approximation, the equation 6.5 is rewritten as:

$$X(j)=X(j-1)+f(X_{j-1})\triangle t+g(X_{j-1})(W(\tau_j)-W(\tau_{j-1})) \tag{6.8}$$

6.1 One Dimensional Autonomuos

Plots the solution of the system:

$$dX(t)=f(X(t))dt+g(X(t))dW \tag{6.9}$$

Where X is a stochastic variable, not deterministic. In this equation, f and g are constant functions.

6.1.1 Gui

Figure 6.1 is explained:

- **Data entry panel:**

  f and g: constant function.

  xo: initial value of the variable X.

  M: number of trajectories

  T: number of periods

  n: number of iterations.

- **Graph panel:**

  Is where it looks the graph of the input data using Euler-Maruyama method 6.8.

6.1.2 File menu: Save

To save must first to graph. It is saved in a flat file, TXT format, where the first column represent the time (t) variable, the second column represent the the median of all stochastic processes and the rest of columns represent all stochastic processes.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in Gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where the first replaces the comma by point, and load file gnuplot.

All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done as follows in linux: perl -p -i -e 's/,/./g' graph.txt

Script file:

```
#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot
load 'graph.gnuplot'
EOF
```

Gnuplot file:

```
set term postscript eps enhanced
set output "SDE with constant functions.eps"
set xlabel "t"
set ylabel "X"
plot "graph.txt" using 1:2 ti "mean" with lines,"<awk '{for(i=2; i<=5;i++){print
$1, $i} }' graph.txt" ti "X
```

Once you save the file gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as showed in figure 6.2. The data are the same as those used in figure 6.1.

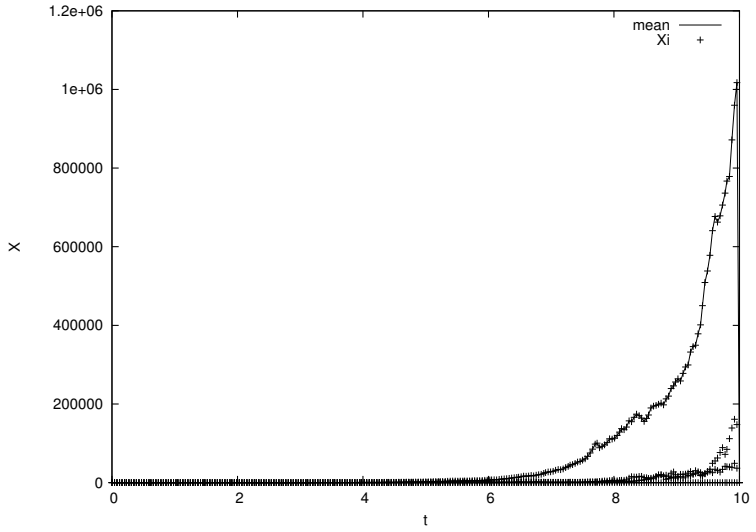


Figure 6.2: Top 4 processes of the stochastic system using GNUPLOT

In Matlab and Octave can plot the data from txt file, saving a file m format the following code:

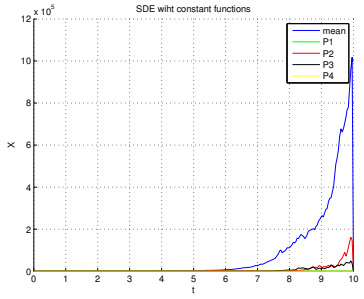
```
clear all
close all
clc
X=load('graph.txt');
hold on
plot(X(:,1),X(:,2),'b')
plot(X(:,1),X(:,3),'g')
plot(X(:,1),X(:,4),'r')
plot(X(:,1),X(:,5),'k')
plot(X(:,1),X(:,6),'y')
title('SDE wiht constant functions')
xlabel('t')
ylabel('X')
legend('mean','P1','P2','P3','P4')
grid on
```

The figure 6.3 displays the graphs obtained in Matlab and Octave respectively. To save in Octave is used:

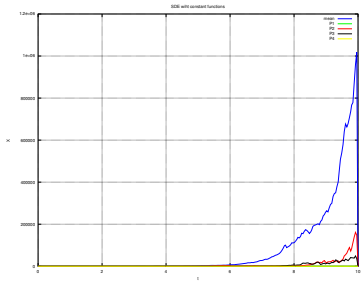
octave:5> print -color filename.eps

6.1.3 File menu: Export to Matlab

To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.



(a) Top 4 processes of the stochastic system in MATLAB loading data



(b) Top 4 processes of the stochastic system in Octave loading data

Figure 6.3: Top 4 processes of the stochastic system in MATLAB and Octave loading data

This is the code that is generated with the data showed in figure 6.1.

```
% DS Simulator
% Stochastic system
clear all
close all
clc n=256;
T=10;
M=1000;
Xo=1.0;
XX=zeros(M,n+1);
dt=T/n;
rand('state',100);
tt=0:dt:T;
dw=sqrt(dt)*randn(M,n);
Xa=Xo*ones(M,1);
XX(:,1)=Xa;
gbarra = waitbar(0,'Procesando...');
```

```
for i=1:n
waitbar(i/n)
XX(:,i+1)=Xa+(1.5)*Xa.*dt+(1.0)*Xa.*dw(:,i);
Xa=XX(:,i+1);
end
XXm=mean(XX);
close(gbarra);
figure(1)
hold on
plot(tt,XXm,'g');
for i=1:5
plot(tt,XX(i,:));
end
xlabel('t')
ylabel('X')
legend('Mean','P1','P2','P3','P4','P5')
```

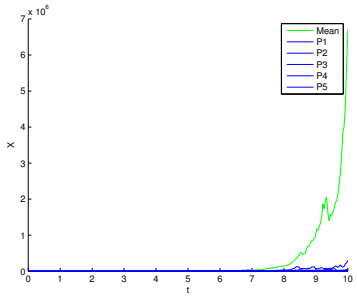
Figure 6.4a shows the result.

6.1.4 File menu: Export to Octave

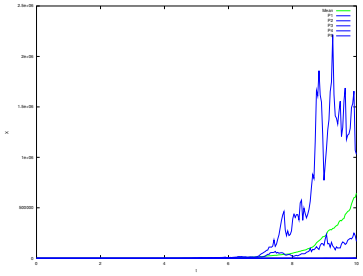
To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

This is the code that is generated with the data showed in figure 6.1.

```
% DS Simulator
% Stochastic system
clear all
close all
clc n=256;
T=10;
M=1000;
Xo=1.0;
XX=zeros(M,n+1);
dt=T/n;
rand('state',100);
tt=0:dt:T;
dw=sqrt(dt)*randn(M,n);
Xa=Xo*ones(M,1);
XX(:,1)=Xa;
for i=1:n
XX(:,i+1)=Xa+(1.5)*Xa.*dt+(1.0)*Xa.*dw(:,i);
Xa=XX(:,i+1);
endfor
XXm=mean(XX);
figure(1)
hold on
plot(tt,XXm,'g');
```



(a) SDE with constant functions exporting code to MATLAB



(b) SDE with constant functions exporting code to Octave

Figure 6.4: Code exported to Matlab and Octave from SDE with constant functions

```
for i=1:5
plot(tt,XX(i,:));
endfor xlabel('t')
ylabel('X')
legend('Mean','P1','P2','P3','P4','P5')
```

Figure 6.4b shows the result.

6.2 One Dimensional

The type of system is in the form of equation 6.3, where f and g are functions in terms of x and t.

6.2.1 Gui

Figure 6.5 is explained:

- **Data entry panel:**

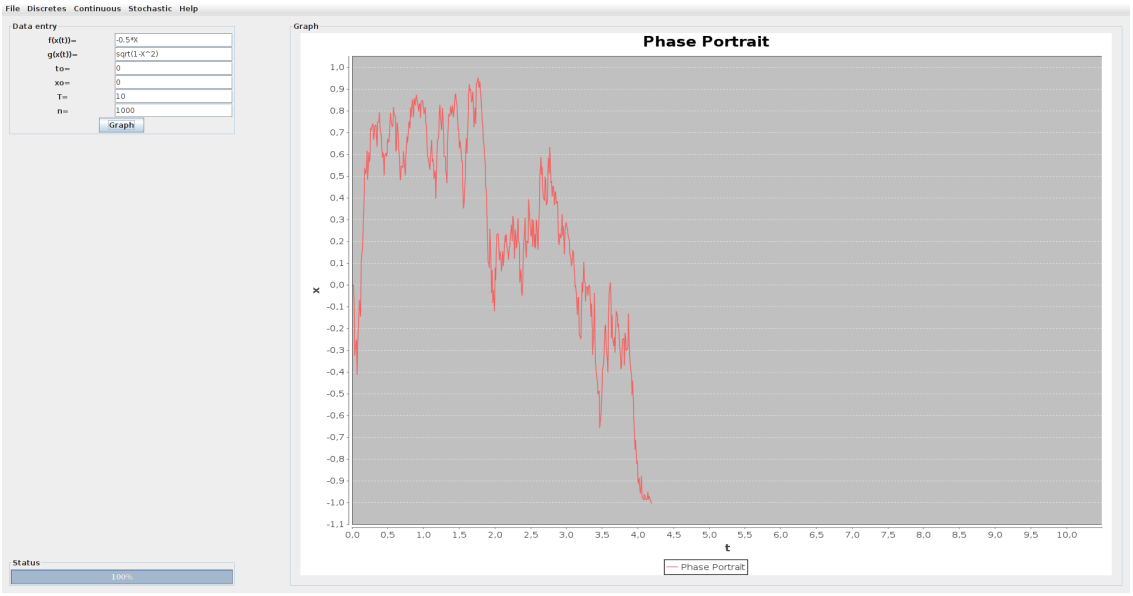


Figure 6.5: Stochastic dynamical system with non-constant functions

f and g: functions in terms of x and t.

to: initial time

xo: initial value of the variable X.

T: number of periods

n: number of iterations.

- **Graph panel:**

  Is where it looks the graph of the input data.

6.2.2 File menu: Save

To save must first to graph. It is saved in a flat file, TXT format, where the first column represent the time (t) variable, the second column represent the X variable.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in Gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where the first replaces the comma by point, and load file gnuplot.

All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done as follows in linux: perl -p -i -e 's/,/./g' graph.txt

Script file:

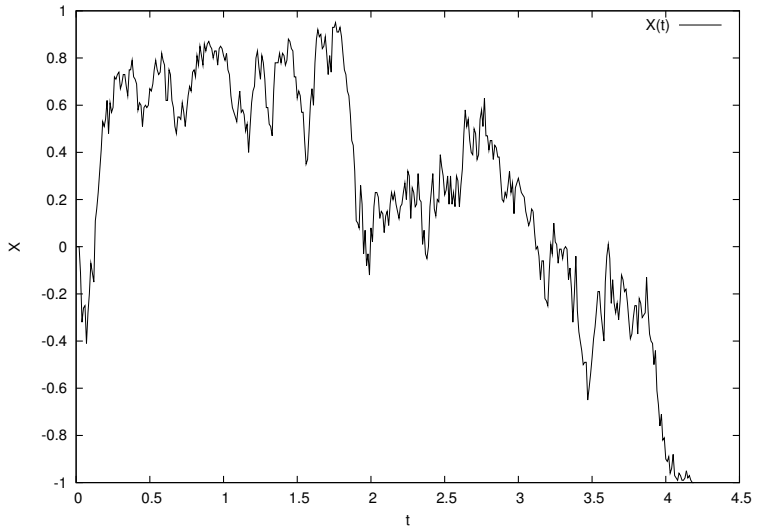


Figure 6.6: SDE with non-constant functions using GNUPLOT

```
#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot
load 'graph.gnuplot'
EOF
```

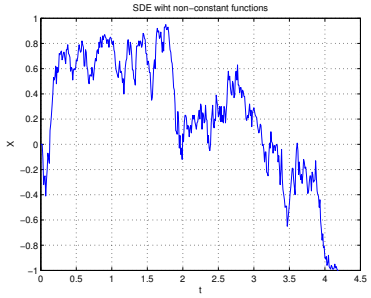
Gnuplot file:

```
set term postscript eps enhanced
set output "SDE with non-constant functions.eps"
set xlabel "t"
set ylabel "X"
plot "graph.txt" using 1:2 ti "X(t)" with lines
```

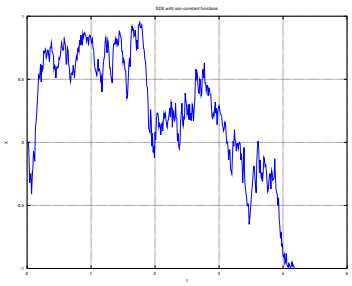
Once you save the file gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as showed in figure 6.6. The data are the same as those used in figure 6.5.

In Matlab and Octave can plot the data from txt file, saving a file m format the following code:

```
clear all
close all
clc
X=load('graph.txt');
plot(X(:,1),X(:,2),'b')
title('SDE wiht non-constant functions')
xlabel('t')
```



(a) SDE with non-constant functions in MATLAB loading data



(b) SDE with non-constant functions in Octave loading data

Figure 6.7: SDE with non-constant functions in MATLAB and Octave loading data

```
ylabel('X')
grid on
```

The figure 6.7 displays the graphs obtained in Matlab and Octave respectively. To save in Octave is used:

octave:5> print -color filename.eps

6.2.3 File menu: Export to Matlab

To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.

This is the code that is generated with the data showed in figure 6.5.

```
% DS Simulator
% Stochastic system
clear all
close all
clc
```

```
XX(1)=0.0;
n=1000;
T=10;
dt=T/n;
X=XX(1);
B(1)=0;
tt=0:dt:T;
gbarra = waitbar(0,'Procesando...');
for i=1:n
B(i+1)=B(i)+sqrt(dt)*randn;
dW=B(i+1)-B(i);
X=XX(i);
t=tt(i);
XX(i+1)=XX(i)+(-0.5*X)*dt+(sqrt(1-X^2))*dW;
waitbar(i/n);
end
close(gbarra);
plot(tt,XX)
xlabel('t')
ylabel('X')
```

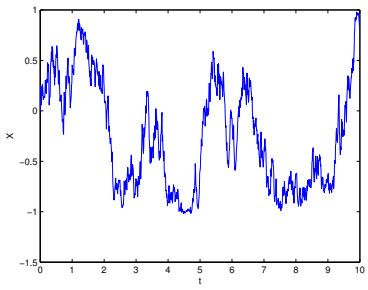
Figure 6.8a shows the result.

6.2.4 File menu: Export to Octave

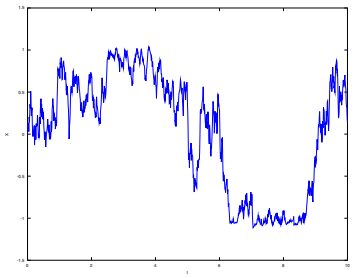
To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

This is the code that is generated with the data showed in figure 6.5.

```
% DS Simulator
% Stochastic system
clear all
close all
clc
XX(1)=0.0;
n=1000;
T=10;
dt=T/n;
X=XX(1);
B(1)=0;
tt=0:dt:T;
for i=1:n
B(i+1)=B(i)+sqrt(dt)*randn;
dW=B(i+1)-B(i);
X=XX(i);
t=tt(i);
XX(i+1)=XX(i)+(-0.5*X)*dt+(sqrt(1-X^2))*dW;
endfor
```



(a) SDE with non-constant functions exporting code to MATLAB



(b) SDE with non-constant functions exporting code to Octave

Figure 6.8: Code exported to Matlab and Octave from SDE with non-constant functions

```
plot(tt,XX)
xlabel('t')
ylabel('X')
```

Figure 6.8b shows the result.

6.3 Two Dimensional

Let be the system

$$\begin{aligned} dX(t) &= f_1(X(t), Y(t))dt + g_1(X(t), Y(t))dW \\ dY(t) &= f_2(X(t), Y(t))dt + g_2(X(t), Y(t))dW \end{aligned} \tag{6.10}$$

where f_1, f_2, g_1 and g_2 are functions in terms of x, y and t.

6.3.1 Gui

Figure 6.9 is explained:

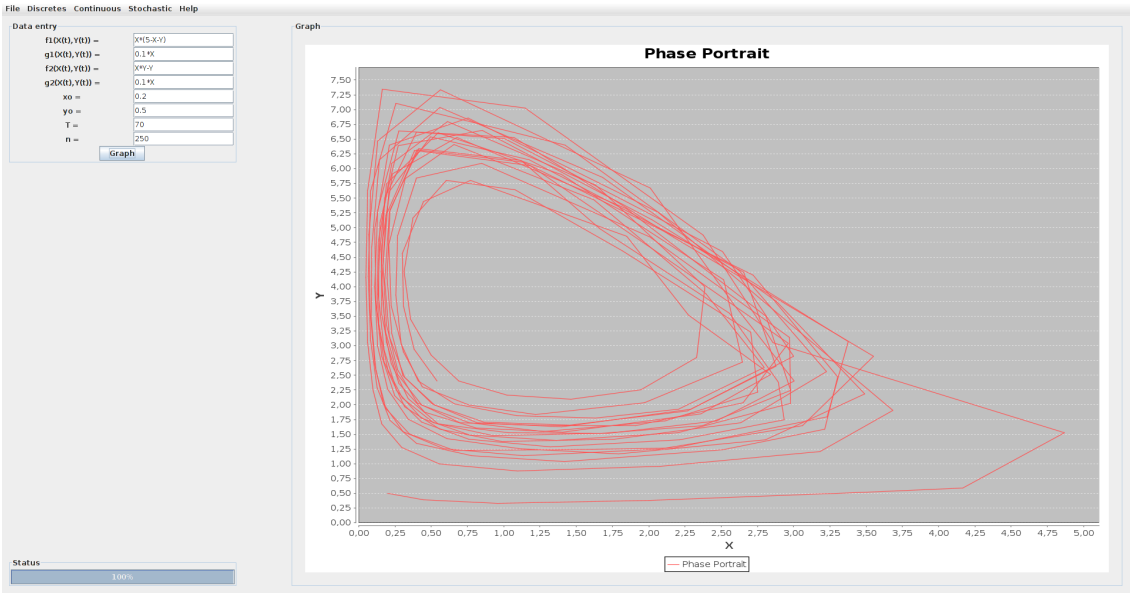


Figure 6.9: Portrait phaser of stochastic dynamical system predator-prey

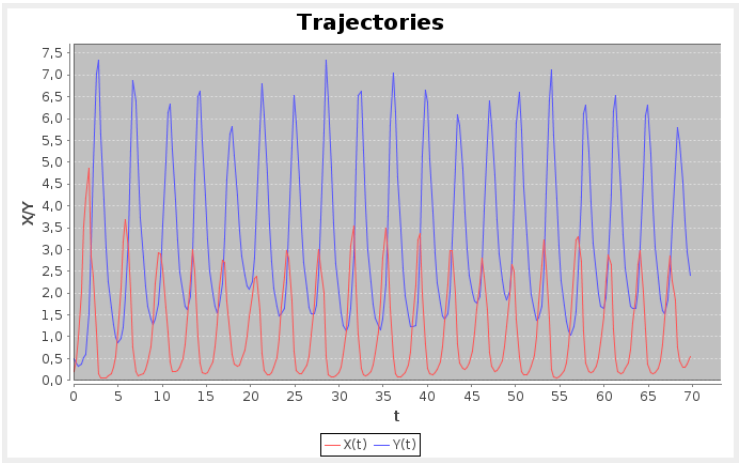


Figure 6.10: Trajectories of stochastic dynamical system predator-prey

- **Data entry panel:**

  f1(X(t),Y(t)), f2(X(t),Y(t)) g1(X(t),Y(t)) and g2(X(t),Y(t)): functions in terms of x, y and t

  xo: initial value of the variable x

  yo: initial value of the variable y

  T: number of periods

  n: number of iterations

- **Graph panel:**

  Is where it looks the graph of the input data.

6.3.2 File menu: Save

To save must first to graph. It is saved in a flat file, TXT format, where the first two columns represent time data, the next column represent data variables X, and Y.

The saved data can be plotted using different tools. For example save the file named graph.txt to graph in Gnuplot uses two files, one file gnuplot which describe the type of graphical and data , and a script to be executed, where the first replaces the comma by point, and load file gnuplot.

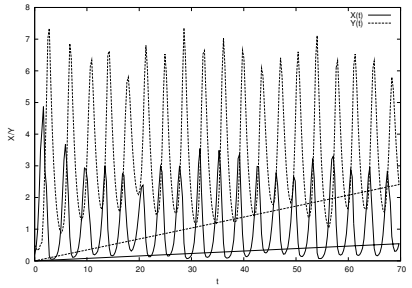
All saved files should be treated first, replace the comma point, as this creates problems at the time to use the file in any program. The replacement is done as follows in linux: perl -p -i -e 's/,/./g' graph.txt

Script file:

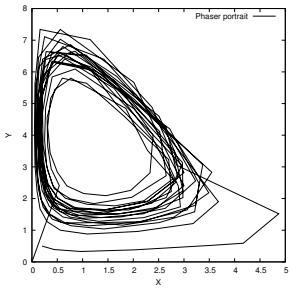
```
#!/bin/bash
perl -p -i -e 's/,/./g' graph.txt
cat << EOF | gnuplot
load 'graph.gnuplot'
EOF
```

Gnuplot file:

```
set term postscript eps enhanced
set output "Trajectories.eps"
set xlabel "t"
set ylabel "X/Y"
plot "graph.txt" using 1:2 ti "X(t)" with lines, "graph.txt" using 1:3 ti "Y(t)" with lines
set term postscript eps enhanced
set output "Phaser portrait.eps"
set size square
set xlabel "X"
set ylabel "Y"
plot "graph.txt" using 2:3 ti "Phaser portrait" with lines
```



(a) Trajectories of stochastic dynamical system using GNUPLOT



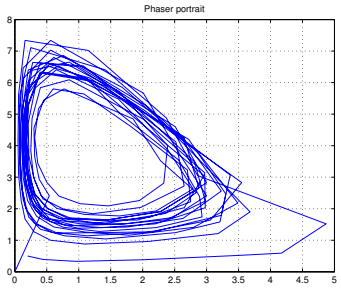
(b) Portrait phaser of stochastic dynamical system using GNUPLOT

Figure 6.11: SDE with non-constant functions in MATLAB and Octave loading data

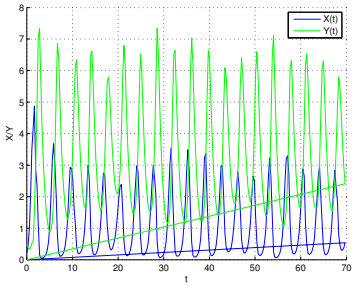
Once you save the file gnuplot as graph.gnuplot it runs the script, remember to change the file permission to run it, and it get the graph as showed in figure 6.11. The data are the same as those used in figure 6.9.

In Matlab and Octave can plot the data from txt file, saving a file m format the following code:

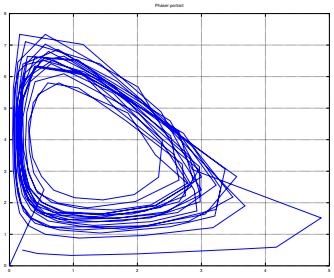
```
clear all
close all
clc
X=load('graph.txt');
figure(1)
hold on
plot(X(:,1),X(:,2),'b')
plot(X(:,1),X(:,3),'g')
xlabel('t') ylabel('X/Y')
legend('X(t)','Y(t)')
grid on
figure(2)
plot(X(:,2),X(:,3))
```



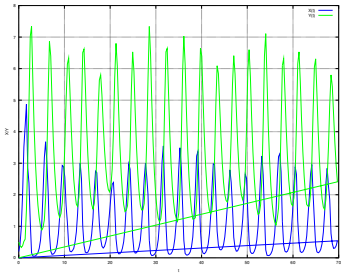
(a) Phaser portrait in MATLAB loading data



(b) Trajectories in Matlab loading data



(c) Phaser portrait in Octave loading data



(d) Trajectories in Octave loading data

Figura 6.12: Phaser portrait loading data in MATLAB and Octave

```
title('Phaser portrait')
grid on
```

The figure 6.12 displays the graphs obtained in Matlab and Octave respectively. To save in Octave is used:

octave:5> print -color filename.eps

6.3.3 File menu: Export to Matlab

To export in Matlab should enter all data only and it saved the code in m format ready to be executed from Matlab.

This is the code that is generated with the data showed in figure 6.9.

```
% DS Simulator
% Stochastic system
clear all
close all
clc
XX(1)=0.2;
YY(1)=0.5;
```

```
n=250;
T=70;
dt=T/n;
B(1)=0;
tt=0:dt:T;
gbarra = waitbar(0,'Procesando...');
for i=1:n
B(i+1)=B(i)+sqrt(dt)*randn;
dW=B(i+1)-B(i);
X=XX(i);
Y=YY(i);
t=tt(i);
XX(i+1)=X+(X*(5-X-Y))*dt+(0.1*X)*dW;
YY(i+1)=Y+(X*Y-Y)*dt+(0.1*X)*dW;
waitbar(i/n);
end
close(gbarra);
figure(1)
hold on
plot(tt,XX,'r')
plot(tt,YY,'b')
xlabel('t')
ylabel('X/Y')
title('trajectories')
figure(2) plot(XX,YY)
xlabel('X')
ylabel('Y')
title('Phase portrait ')
```

Figure 6.13 shows the result.

6.3.4 File menu: Export to Octave

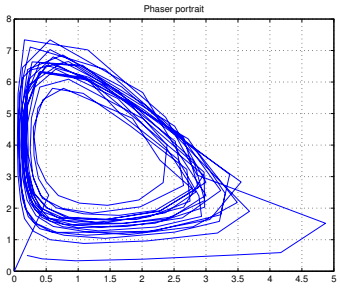
To export in Octave should enter all data only and it saved the code in m format ready to be executed from Octave.

This is the code that is generated with the data showed in figure 6.9.

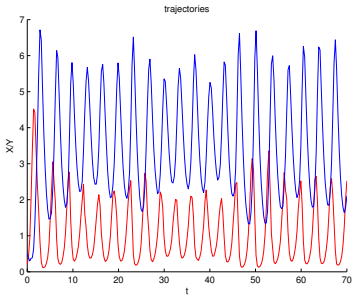
```
% DS Simulator
% Stochastic system
clear all
close all
clc
XX(1)=0.2;
YY(1)=0.5;
n=250;
T=70;
dt=T/n;
B(1)=0;
```

```
tt=0:dt:T;
for i=1:n
B(i+1)=B(i)+sqrt(dt)*randn;
dW=B(i+1)-B(i);
X=XX(i);
Y=YY(i);
t=tt(i);
XX(i+1)=X+(X*(5-X-Y))*dt+(0.1*X)*dW;
YY(i+1)=Y+(X*Y-Y)*dt+(0.1*X)*dW;
endfor
figure(1)
hold on
plot(tt,XX,'r')
plot(tt,YY,'b')
xlabel('t')
ylabel('X/Y')
title('trajectories')
figure(2)
plot(XX,YY)
xlabel('X')
ylabel('Y')
title('Phase portrait ')
```

Figure 6.14 shows the result.

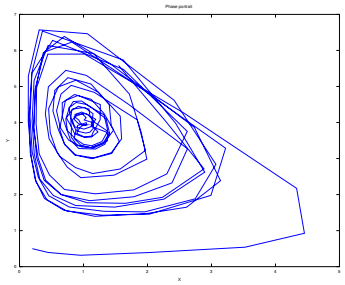


(a) Phaser portrait exporting code to MATLAB

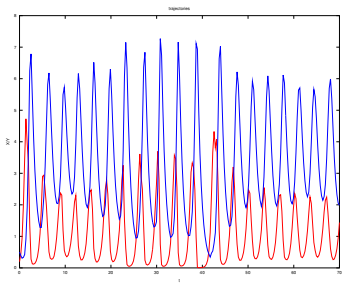


(b) Trajectories exporting code to Matlab

Figure 6.13: Code exported to Matlab from Phaser portrait



(a) Phaser portrait exporting code to Octave



(b) Trajectories exporting code to Octave

Figure 6.14: Code exported to Octave from Phaser portrait

Bibliography

[1] Singularsys [on line]. [Date of access: July 18, 2011. Available in: http://www.singularsys.com/jep/doc/html/functions.html

[2] JFreeChart [on line]. [Date of access: July 18, 2011]. Available in: http://www.jfree.org/jfreechart/

[3] Hirsch M., Smale S., Devaney R. Differentials equations, Dynamical systems and an introduction to chaos. 2 ed. Elsevier, 2004

[4] Holmgren, Richard A. A First Course in Discrete Dynamical Systems. USA. Springer-Verlag, 1994

[5] Blanchard, P., Devaney L., Robert. Ecuaciones diferenciales. México. International Thomson Editores,1999

[6] Campos R., D, Isaza D., J. Prolegómenos a los sistemas dinámicos. Facultad de Ciencias – Departamento de Física, Universidad Nacional, Bogotá, 2002

[7] Mathews, J. Numerical Methods Using MATLAB. 3 ed. USA. Prentice Hall, 1999

[8] Jordan, D. W.; Smith, P. (2007). Nonlinear Ordinary Differential Equations (fourth ed.). Oxford Univeresity Press. ISBN 978-0-19-9208241. Chapter 1.

[9] Numerical Recipes in Fortran: The Art of Scientic Computing (ISBN 0-521-43064-X). Cambridge University Press, 1992

[10] Alligood K., Sauer T., Yorke J. Chaos: An Introduction to Dynamical Systems. Springer-Verlag, New York, 1998

[11] A. Wolf, J. B. Swift, H. L. Swinney, and J. A. Vastano, "Determining Lyapunov Exponents from a Time Series," Physica D, Vol. 16, pp. 285-317, 1985

[12] MATDS - Govorukhin V.N [on line]. [Date of access: July 18, 2011]. Available in: http://www.math.rsu.ru/mexmat/kvm/matds/

[13] Kloeden P.,Platen E. Numerical Solution of Stochastic Differential Equations. Springer Verlag. Berlin, 1992.

List of Figures

List of Tables

Index