Matlab Adventures in Bifurcations & Chaos

James P. McEvoy (99375940)

26 March 2003

Abstract

"Matlab Adventures in Bifurcations & Chaos" (ABC++) is a GUI application, which runs in a Matlab environment and has been designed for the exploration of bifurcations and chaos within the Chua Circuit paradigm. ABC++ has been created to succeed a program written in 1993 by Michael P. Kennedy; called "Adventures in Bifurcation & Chaos" (ABC)[6]. As such it retains all the features of the original program as well as adding some additional features. The program comes with an extensive database of sets of system parameters and initial conditions, which produces a wide variety of the dynamical behaviors reported for Chua's Circuit.

Contents

List of Figures

Part I

Introduction & Objectives

1. Introduction

1.1 Motivation

> "The science of chaos cuts across traditional scientific disciplines, tying together unrelated kinds of wildness and irregularity, from the turbulence of weather to the complicated rhythms of the human heart, from the design of snowflakes to the whorls of windswept desert sands" - *James Gleick*[1]

By now every self-respecting engineer has heard reference to the engineering rule of thumb - 'linearize; then analyze', its no surprise therefore that the study of non-linear dynamics is still a great unexplored discipline. However, what is the study of non-linear dynamics and how does it affect this project?

The method of linearization is little more than a mathematical abstraction of the real world around us, and while having considerable use in a theoretical context it lacks the resolution to describe the majority of real-world dynamics accurately. When I say it lacks the resolution I mean the fact that linearization implies taking a dataset and moulding that set into something which can be described mathematically, during this moulding operation one inevitably loses resolution in rounding operations. Mathematically; humans like dealing with finite precision numbers.

> "Everything is Number" – *Pythagoras*[2]
>
> Nature on the other hand has no such resolution problem - here is some food for thought to prove this.

- Take a square of any dimensions and a ruler of finite precision (meaning a ruler with a finite number of distinct markings). When measuring the diagonal of said square, its length will always lie between two adjacent markings even though the distance between the markings may be infinitesimally small.[3]

Clearly as humans, we can't cope with such huge resolution so we quantize! We take a non-linear system and linearize it in the area we are interested in to ease mathematical analysis. What is wrong with such an intuitive mathematical method, well what if this non-linear dynamic is driving (an input to) a system that is sensitive to its initial conditions? Due to this quantization initial conditions can never be specified exactly and are only known to within a certain tolerance $\zeta > 0$, $\therefore$ if two initial conditions x_0 and $\hat{x}_0$ lie within ζ of each other they cannot be distinguished, yet the output of the system they drive will diverge an become unrelated over time[1] - *clearly a problem.*

> "Clouds are not spheres, mountains are not cones, coastlines are not circles, and bark is not smooth, nor does lightning travel in a straight line." - *Benoit Mandelbrot*

In order to introduce a report on Chaos it is important to introduce its most identifiable symbol, the *Lorenz Attractor* [5], see Figure 1.1

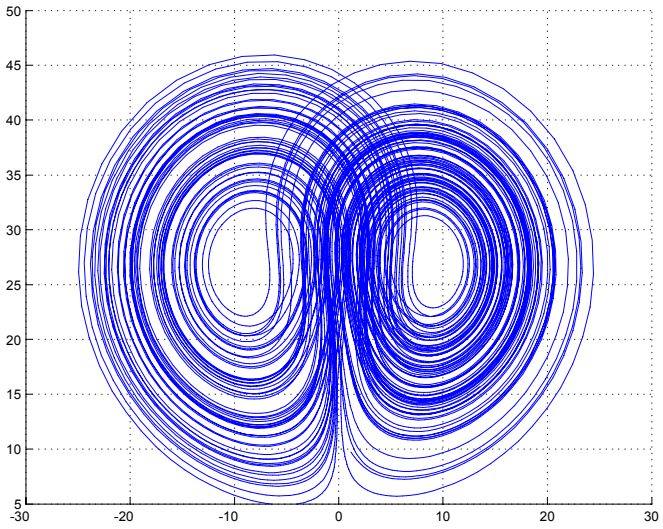

Figure 1.1: The Lorenz Attractor, named after curious meteorologist Edward Lorenz, constructed in Matlab using Runga-Kutta 4

[1] This sensitivity to initial conditions has been immortalized in the now ubiquitously known 'Butterfly Effect'

The Lorenz Attractor can be described by a system of three differential equations shown :

$$\frac{dx}{dt} = \alpha(y - x) \qquad \frac{dy}{dt} = x(\delta - z) - y \qquad \frac{dz}{dt} = xy - \beta z \tag{1.1}$$

where α, δ and β are constants for a given system

When these equations are solved numerically using an integration routine, see Section 4.3.1, and plotted together on a three-dimensional plane, the Lorenz Attractor is produced.

1.2 History

> "Often an irregular noise is heard in the telephone receivers before the frequency jumps to the next lower value. However, this is a subsidiary phenomenon, the main effect being the regular frequency demultiplication" - *Van Der Pol*[8]

In a world of mathematics once confined to the linear world, Var der Pol like many before him disregarded surprisingly complex behavior lurking in simple models as noise. The only systems that could be understood in the past were those that were believed to be linear, that is to say, systems that follow deterministic predictable patterns and arrangements. Linear equations, linear functions and linear algebra are all areas that have been understood and mastered by academia. However, the problem arises that we humans do not live in an even remotely linear world. Common sense testifies to the stochastic nature of practically everything around us, after all what good is the lotto in a deterministic world? So how may one go about understanding a nonlinear system in a world that is confined to the logical, regimented linearity of everything? This is the question that scientists and mathematicians became burdened with in the 19th Century; hence, a new science and mathematics was derived: chaos theory.

Some chaos theory advocates[2] go as far as to say that twentieth-century science will be remembered for just three things: relativity, quantum mechanics,

[2] James Gleick, see Prologue of [1].

and chaos. As one physicist put it: "Relativity eliminated the Newtonian[3] illusion of absolute space and time; quantum theory eliminated the Newtonian dream of a controllable measurement process[4]; and chaos theory eliminates the Laplacian[5] fantasy of deterministic predictability".

Chaos theory has matured since to pioneering work of Lorenz, Feigenbaum, Mandelbrot and others, it is a door into this vibrant exciting area that "Matlab Adventures in Bifurcations & Chaos" (ABC++) attempts to open.

[3]Systems governed by Sir Isaac Newton's three fundamental equations of motion.

[4]See Heisenberg Uncertainty Principle in relevant text.

[5]Mathematician Pierre-Simon de Laplace (1749-1827) envisaged a Vast Intellect, which could predict future behaviour of every particle in the universe knowing only the original position and forces action on those particles.

2. Objectives

2.1 Project Objectives

This section of the report lays out the project objectives as specified by our project supervisor, Professor Michael P. Kennedy. At this stage it is useful to put our project into context and describe ABC[6], the predecessor to ABC++.

ABC is a program for IBM-compatible PCs and was written by Michael P. Kennedy in QUICK-BASIC[1] back in 1993. ABC simulates the Chua Circuit paradigm and is capable of exhibiting the various dynamical behaviors reported for Chua's Circuit, see Chapter 3. Unfortunately, since 1993 QUICK-BASIC has more or less died out due to the creation of VISUAL-BASIC. Thus, ABC like many good programs of its day has been resigned to the computer graveyard of incompatibility.

While the ABC binary still runs, it was designed for VGA[2] systems, which bear little or no resemblance to the SVGA[3] systems of today! Also since 1993, significant progress has been made in program 'user friendliness' and command line programs like ABC certainly don't fit this bracket. Fortunately, the extensive database of sets of initial conditions and parameters is still intact. This database was the result of months of trial and error to get those elusive trajectories the Chua paradigm is capable of producing (including homoclinic and heteroclinic orbits and a plethora of other chaotic attractors) and is therefore an extremely valuable asset.

The main objective of our project is to bring this valuable scientific database back to life and to provide a framework to ensure that ABC's original goals of portability and 'user friendliness' are adhered to. In addition to emulating the original program, we also hope to lay down a scalable modular framework which will enable us (and hopefully others) to add features to the program!

[1] Microsoft's implementation of the BASIC language for DOS.
[2] Video Graphics Array.
[3] Super Video Graphics Array.

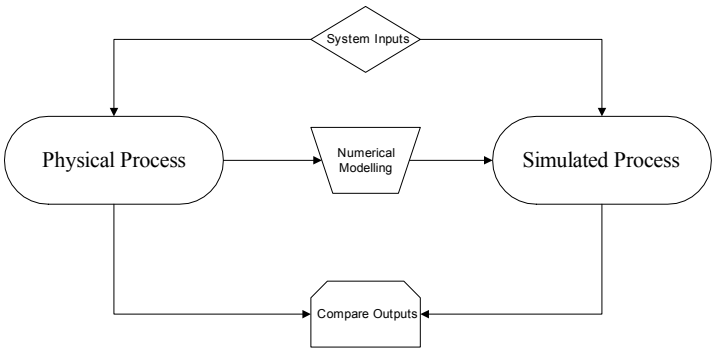

Figure 2.1: Typical Engineering design process

2.2 Personal Objectives

Aside from the assigned Project Objectives, taking on "Matlab Adventures in Bifurcations & Choas" afforded the exploration of some neat mathematics and, what is probably more important for a practicing engineer, how to integrate these mathematical concepts into a programming environment.

Working with Chua's Circuit, see Chapter 3, as a building block for studying non-linear dynamics instead of some other paradigm was also important, as Chua's Circuit is the simplest *real circuit* to be proven mathematically[4] to exhibit chaotic behavior.

Successful completion of the project required good project management skills and an understanding of fundamental engineering design procedures, see Figure 2.1, skills which will stand to any engineer in his/her career.

[4] in the sense of Shil'nikov[9].

Part II

Background Theory

3. Chua's Circuit Paradigm

3.1 Circuit Schematic

See Figure 3.1 for a schematic of Chua's Circuit/Oscillator[10]. This circuit is comprised from a linear inductor L in series with a linear resistor R_0, two linear capacitors C_2 and C_1, a dissipative linear resistor R, and a voltage controlled[1] non-linear resistor N_R called a *Chua Diode*.

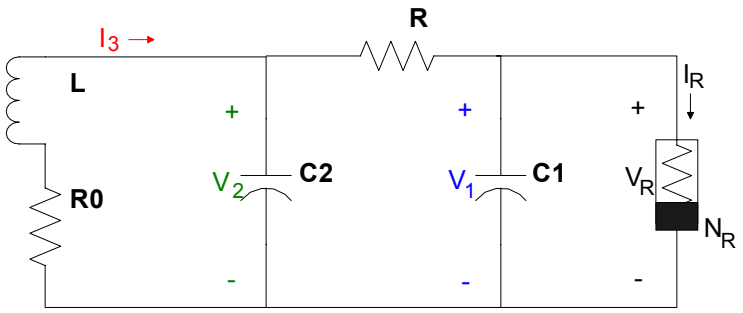

Figure 3.1: Chua's Oscillator

3.2 Circuit Theory

As can be seen from the circuit schematic, Chua's oscillator is readily constructed from low cost standard electronic components.

In addition, as will be shown over the course of this report, exhibits a rich variety of bifurcations and chaos.

In order to exhibit chaos, an autonomous[2] circuit consisting of resistors, capacitors and inductors must contain:

[1] A two-terminal nonlinear resistor is *voltage controlled* if the current through the resistor may be written as a function of the voltage across its terminals.

[2] A system defined by equations, where the independent variable t does not appear explicitly.

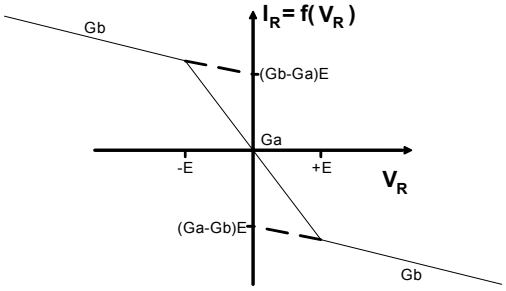

Figure 3.2: *I-V* Transfer Characteristic for *Chua's Diode*

1. at least one nonlinear element
2. at least one locally active resistor
3. at least three energy-storage elements

The driving-point (*I-V* transfer) characteristic of the *Chua Diode* has the form:

$$I_R = f(V_R) = G_b V_R + \frac{1}{2}(G_a - G_b)\{|V_R + E\,| - |\,V_R - E|\} \tag{3.1}$$

As can be seen in Figure 3.2, the *I-V* transfer characteristic is comprised from two outer affine[3] regions and one inner linear region. This piecewise linear characteristic is globally *non-linear*.

Chua's Oscillator can be described by a set of three ordinary differential equations called *Chua's Equations*[11].

$$\frac{dI_3}{dt} = -\frac{R_0}{L} I_3 - \frac{1}{L} V_2 \tag{3.2}$$

$$\frac{dV_2}{dt} = \frac{1}{C_2} I_3 - \frac{G}{C_2}(V_2 - V_1) \tag{3.3}$$

$$\frac{dV_3}{dt} = \frac{G}{C_1}(V_2 - V_1) - \frac{1}{C_1} f(V_1) \tag{3.4}$$

[3] Affine line $\Longrightarrow$ A straight line which does not pass through the origin.

$$\text{where } f(V_1) = G_b V_1 + \frac{1}{2}(G_a - G_b)\{|V_1 + E\,| - |\,V_1 - E|\}$$

3.2.1 Qualitative Circuit Description

An important point to take into consideration when studying *Chua's Oscillator* is the lack of a power source in the circuit, the astute reader would have noticed from Figure 3.2 that $v_R(t)i_R(t) < 0$[4], this imples that N_R is always locally *active* and keeps supplying power to the external circuit.

Also note that the parallel connection[5] of C_2 and L acts as one basic oscillatory mechanism in the (v_{C2}, i_L)-plane, while the resistor R provides the interaction between the (v_{C2}, i_L)-oscillatory component and the *active* resistor N_R together with C_1. Since the *active* resistor N_R keeps supplying power to the circuit the R resistor needs to dissipative to stimulate the attracting nature of chaotic trajectories.

3.2.2 Quantitative Circuit Description

Piecewise Linear Description

As we have seen in Eqns (3.2,3.3 & 3.4), the circuit may be described by three ordinary differential equations (state equations). Choosing I_3,V_2 and V_1 as state variables and observing that the piecewise linear component N_R is in parallel with V_1, we can tell that the vector field associated with these state-variables will also be piecewise in nature[6]. That is to say that the it will be partitioned into three distinct affine regions: $V_1 < E$, $|V_1| \leq E$, $V_1 > E$. See Figure 3.3, where these regions D_{-1}, D_0 and D_1 are partitioned by the U_1 and U_{-1} planes . Using piecewise-linear analysis, we can examine each region separately, then synthesize each distinct piece[7].

[4] Except at the origin.

[5] Tank circuit.

[6] See Figure 3.2, notice piecewise nature of V_R.See Figure 3.1, notice $V_1 = V_R$.

[7] This property of *Chua's Oscillator* shows its strength as a learning tool $\Longrightarrow$ we can analyse *nonlinear* systems using the same mathematical tools used to analyse *linear* systems.

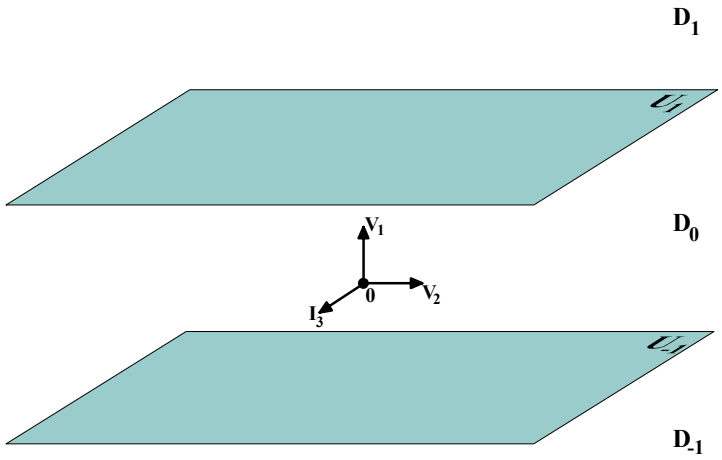

Figure 3.3: Chua's Oscillator *State-Space* partitioned by planes U_1 and U_{-1} into regions D_{-1}, D_0 and D_1.

Linear Algebra Description

Eqns (3.2,3.3 & 3.4) describe a third order autonomous dynamical[8] system of the form

$$\frac{d\mathbf{X}(t)}{dt} = \mathbf{F}(\mathbf{X}(t)) \tag{3.5}$$

where $\mathbf{X}(t) = [I_3(t), V_2(t), V_3(t)]^T$ is called the state of the system at time t, and the solution $\mathbf{X}(t)$ starting from any initial state $\mathbf{X}_0 = [I_3(0), V_2(0), V_3(0)]^T$ is called the *trajectory*[9] of Chua's Oscillator. Thus any trajectory reported for Chua's Oscillator can be fully described by:

$$\dot{\mathbf{X}} = \mathbf{F}(\mathbf{X}), \mathbf{X}(0) = \mathbf{X}_0 \tag{3.6}$$

The experienced reader will recognize the fundamental importance of Eqn

[8] A *dynamical system* is any system that allows one to determine (at least theoretically) future states of the system given its present or past state

[9] How the states of the system progress *w.r.t* time, plotted in three dimensional State-Space is called the *trajectory* of the system.

3.6 in the context of chaotic systems. $\dot{\mathbf{X}} = \mathbf{F}(\mathbf{X})$[10] highlights the fact that chaos is a science of *process*[11], $\mathbf{X}(0) = \mathbf{X}_0$ highlights the inherent reliance of chaotic systems on initial conditions.

An *Equilibrium Point*[12] of (3.6) is a state $\mathbf{X_Q}$ at which $\dot{\mathbf{X}} = 0$. Thus if $\mathbf{X}(0) = \mathbf{X_Q}$, then $\dot{\mathbf{X}} = 0$ (since $\mathbf{F}(\mathbf{X_Q}) = 0$) $\therefore \mathbf{X}(\infty) = \mathbf{X_Q}$ i.e. a trajectory starting from an equilibrium point remains forever at that point.

Equilibrium points for Chua's Oscillator are defined by:

$$\begin{aligned} 0 &= -\frac{R_0}{L} I_3 - \frac{1}{L} V_2 \\ 0 &= \frac{1}{C_2} I_3 - \frac{G}{C_2}(V_2 - V_1) \\ 0 &= \frac{G}{C_1}(V_2 - V_1) - \frac{1}{C_1} I_R \end{aligned} \tag{3.7}$$

where I_R is defined by Eqn 3.1. Again due to the piecewise linear nature of I_R, Chua's Oscillator has three distinct equilibrium points $[P^+, \mathbf{0}, P^-]$, one for each region D_{-1}, D_0 and D_1 respectively. These are found using the following:

$$P^- = \begin{bmatrix} \frac{G(G_b - G_a)E}{G_b' + GG_bR_0} \\ \frac{GR_0(G_b - G_a)E}{G_b' + GG_bR_0} \\ \frac{(1+GR_0)(G_b - G_a)E}{\tilde{G}_b + GG_bR_0} \end{bmatrix}, \mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, P^+ = \begin{bmatrix} \frac{G(G_a - G_b)E}{G_b' + GG_bR_0} \\ \frac{GR_0(G_a - G_b)E}{G_b' + GG_bR_0} \\ \frac{(1+GR_0)(G_a - G_b)E}{\tilde{G}_b + GG_bR_0} \end{bmatrix} \tag{3.8}$$

where $G = \frac{1}{R}$ and $\tilde{G_b} = G + G_b$. The dynamics of each region D_{-1}, D_0 and D_1 is specified by a set of three eigenvalues[14] associated with **each** equilibrium point. We will now proceed to study each *State-Space* region D_{-1}, D_0 and D_1 in a little more detail, taking into consideration the eigenvalues mentioned above.

[10] $\mathbf{F}(\mathbf{X}(t))$ is known as a *vector field* because it defines the direction & speed of a trajectory at every location in the state-space.

[11] Of becoming rather than being.

[12] An equilibrium point of an electronic circuit is simply a *dc solution*.

3.3 Dynamics of Distinct State-Space Regions

3.3.1 Inner Region

In the D_0 region $|V_1| \leq E \therefore I_R = G_a V_R$ (see Eqn 3.1) $\therefore$ our *State-Equations*, (Eqns 3.2,3.3 & 3.4), reduce to

$$\begin{aligned} \frac{dI_3}{dt} &= -\frac{R_0}{L} I_3 - \frac{1}{L} V_2 \\ \frac{dV_2}{dt} &= \frac{1}{C_2} I_3 - \frac{G}{C_2}(V_2 - V_1) \\ \frac{dV_3}{dt} &= \frac{G}{C_1} V_2 - \frac{\tilde{G}_a}{C_1} V_1 \end{aligned} \tag{3.9}$$

where $\tilde{G}_a = G + G_a$. This can be simplified to

$$\begin{aligned} \dot{\mathbf{X}} &= \mathbf{A}\mathbf{X} \\ \text{where } \mathbf{A} &= \begin{bmatrix} -\frac{R_0}{L} & \frac{1}{L} & 0 \\ \frac{1}{C_2} & -\frac{G}{C_2} & \frac{G}{C_2} \\ 0 & \frac{G}{C_1} & -\frac{\tilde{G}_a}{C_1} \end{bmatrix} \end{aligned} \tag{3.10}$$

We also know that the D_0 region has an equilibrium point $\mathbf{0}$ at the origin, this equilibrium point's stability[13] is completely governed by the eigenvalues of the coefficient matrix $\mathbf{A}$. Or in other terms the zeros of the characteristic equation, $\det(\lambda\mathbf{I}^{14} - \mathbf{A}) = 0$,

$$\lambda^3 + \left(\frac{G}{C_2} + \frac{\tilde{G}_a}{C_1} + \frac{R_0}{L}\right)\lambda^2 + \left(\frac{1 + GR_0}{LC_2} + \frac{\tilde{G}_a R_0}{LC_1} + \frac{GG_a}{C_1 C_2}\right)\lambda + \frac{GG_a R_0 + \tilde{G}_a}{LC_1 C_2} = 0 \tag{3.11}$$

[13] An equilibrium point is *stable* if trajectories starting close to it remain close for all time.

[14] Identity matrix

Solving this polynomial returns the three eigenvalues λ_1, λ_2 and λ_3 for the inner region D_0. Observe that the roots of Eqn 3.11 are either all real or made up from a real root and a complex conjugate pair. Let us assume the later and presume $\lambda_1 = \gamma_0, \lambda_2 = \sigma_0 + j\omega_0$ and $\lambda_3 = \sigma_0 + j\omega_0$.

In this special case, we now know that the solution to Eqn 3.10 has the form[14]:

$$\mathbf{x}(t) = c_r \exp(\gamma_0 t)\vec{\xi}_{\gamma_0} + 2c_c \exp(\sigma_0 t)\left[\cos\left(\omega_0 t + \phi_c\right)\vec{\eta}_r - \sin\left(\omega_0 t + \phi_c\right)\vec{\eta}_i\right] \tag{3.12}$$

where $\vec{\xi}_{\gamma_0}$ is the eigenvector defined by $\mathbf{A}\vec{\xi}_{\gamma_0} = \gamma_0\vec{\xi}_{\gamma_0}$[15], where $\vec{\eta}_r$ and $\vec{\eta}_i$ are the real and imaginary parts if the eigenvector associated with the complex conjugate pair of eigenvalues, and where c_r, c_c and ϕ_c are real constants determined by the systems initial conditions.

Renaming the real eigenvector from $\vec{\xi}_{\gamma_0}$ to $E^r(\mathbf{0})$ $\therefore$

$$\mathbf{A}E^r(\mathbf{0}) = \gamma_0 E^r(\mathbf{0}) \tag{3.13}$$

And defining $E^c(\mathbf{0})$ as the plane spanned by $\vec{\eta}_r$ and $\vec{\eta}_i$ a we can break the solution $\mathbf{x}(t)$ of (3.10) into two distinct parts

$$\begin{aligned} \mathbf{x}_r(t) &= c_r \exp(\gamma_0 t)E^r(0),\ \mathbf{x}_r(t) \in E^r(\mathbf{0}) \\ \mathbf{x}_c(t) &= 2c_c \exp(\sigma_0 t)\left[\cos\left(\omega_0 t + \phi_c\right)\vec{\eta}_r - \sin\left(\omega_0 t + \phi_c\right)\vec{\eta}_i\right],\ \mathbf{x}_c(t) \in E^c(\mathbf{0}) \end{aligned} \tag{3.14}$$

These equations are of *fundamental inportance* to the stability of the inner equilibrium point $\mathbf{0}$. We can see from Eqn 3.14 that real parts of the eigenvalues λ_1, λ_2 and λ_3, namely γ_0 and σ_0, determine *trajectory stability* near $\mathbf{0}$. If $\gamma_0 > 0$, $\mathbf{x}_r(t)$ grows exponentially in the direction of $E^r(\mathbf{0})$, if $\gamma_0 < 0$, $\mathbf{x}_r(t)$ tends to $\mathbf{0}$ asymptotically. When $\sigma_0 > 0$, $\mathbf{x}_c(t)$ spirals away from $\mathbf{0}$ along $E^c(\mathbf{0})$ and vice versa.

So how do we solve for $E^r(\mathbf{0})$ and $E^c(\mathbf{0})$? Some basic manipulation of Eqn 3.13, results in $(\gamma_0\mathbf{I} - \mathbf{A})E^r(\mathbf{0}) = \mathbf{0}$. Writing $E^r(\mathbf{0})$ as $[x, y, z]^T$, results in

[15] Following standard linear-algebra design methodology[15]

$$\begin{bmatrix} \gamma_0 + \frac{R_0}{L} & -\frac{1}{L} & 0 \\ -\frac{1}{C_2} & \gamma_0 + \frac{G}{C_2} & -\frac{G}{C_2} \\ 0 & -\frac{G}{C_1} & \gamma_0 + \frac{\tilde{G}_a}{C_1} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.15}$$

Solving the above and then letting $z = 1$[16], the corresponding eigenvector is

$$E^r(\mathbf{0}) = \begin{bmatrix} \left(\gamma_0 + \frac{G}{C_2}\right)\left(\gamma_0 + \frac{\tilde{G}_a}{C_1}\right)\frac{C_1 C_2}{G} - G \\ \frac{C_1}{G}\left(\gamma_0 + \frac{\tilde{G}_a}{C_1}\right) \\ 1 \end{bmatrix} = \begin{bmatrix} x_{\mathbf{0}} \\ y_{\mathbf{0}} \\ z_{\mathbf{0}} \end{bmatrix} \tag{3.16}$$

The real and imaginary parts of the complex eigenvectors determined by $\sigma_0 \pm j\omega_0$ span the complex eigenplane $E^c(\mathbf{0})$. Any state $\mathbf{X} \in \mathbb{R}^3$ lies on the plane $E^c(\mathbf{0})$ if $(\mathbf{X} - \mathbf{0}) \bullet \vec{\Psi}_{E^c(\mathbf{0})} = 0$[17] where $\vec{\Psi}_{E^c(\mathbf{0})}$ is the vector normal to $E^c(\mathbf{0})$, thus

$$\vec{\Psi}_{E^c(\mathbf{0})} = \begin{bmatrix} -\frac{LC_2}{G}\left(\gamma_0 + \frac{G}{C_2}\right)\left(\gamma_0 + \frac{\tilde{G}_a}{C_1}\right) + \frac{LG}{C_1} \\ \frac{C_2}{G}\left(\gamma_0 + \frac{\tilde{G}_a}{C_1}\right) \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{L}{C_1}x_{\mathbf{0}} \\ \frac{C_2}{C_1}y_{\mathbf{0}} \\ z_{\mathbf{0}} \end{bmatrix} \tag{3.17}$$

Now that we know $\vec{\Psi}_{E^c(\mathbf{0})}$, the vector normal to the complex eigenplane $E^c(\mathbf{0})$, and we know, $\mathbf{0}$, a point on the plane $\therefore$ we can specify the $E^c(\mathbf{0})$ plane itself with ease[18].

$$E^c(\mathbf{0}) \Longrightarrow \left(-\frac{L}{C_1}x_{\mathbf{0}}\right)(x - \mathbf{0}\,[1]) + \left(\frac{C_2}{C_1}y_{\mathbf{0}}\right)(y - \mathbf{0}\,[2]) + (z_{\mathbf{0}})(z - \mathbf{0}\,[3]) = 0$$

$$\therefore E^c(\mathbf{0}) \Longrightarrow \left(-\frac{L}{C_1}x_{\mathbf{0}}\right)x + \left(\frac{C_2}{C_1}y_{\mathbf{0}}\right)y + (z_{\mathbf{0}})\,z = 0, \text{ since } \mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.18}$$

[16] z is a free variable $\therefore$ once specified the other variables can be calculated.
[17] Where $\mathbf{0}$ is the inner equilibrium point & '$\bullet$' represents the dot product.
[18] See any good Linear Algebra book's [14] section on plane equations.

We now have everything we need to describe, both qualitatively and quantitatively, the dynamics of a Chua's Oscillator trajectory in the D_0 region around the middle equilibrium point **0**.

3.3.2 Outer Regions

The approach we took understanding the dynamics of the inner D_0 region also serves us well in understanding the dynamics of the two outer regions D_1 and D_{-1}(where $|V_1| > E$). At this point it is useful to point out that Eqns (3.2, 3.3 and 3.4) are symmetric $w.r.t$ the origin, that is to say the vector field is invariant under the transformation

$$(x, y, z) \longrightarrow (-x, -y, -z)$$

Therefore, we can focus simply on the D_1 region and be confident that the dynamics in the D_{-1} region are symmetric about the origin. Another point to notice is the fact the only system variable to change within regions D_{-1}, D_0 and D_1, is the slope of the I-V transfer characteristic, see Figure 3.2, where it changes from G_a to G_b. Ok lets begin, in the D_1 outer region our *State-Equations*, (Eqns 3.2,3.3 & 3.4), reduce to:

$$\begin{aligned} \frac{dI_3}{dt} &= -\frac{R_0}{L} I_3 - \frac{1}{L} V_2 \\ \frac{dV_2}{dt} &= \frac{1}{C_2} I_3 - \frac{G}{C_2}(V_2 - V_1) \\ \frac{dV_3}{dt} &= \frac{G}{C_1} V_2 - \frac{\tilde{G}_b}{C_1} V_1 - \left(\frac{G_a - G_b}{C_1} \right) E \end{aligned} \tag{3.19}$$

where $\tilde{G}_b = G + G_b$. As before the stability of the outer equilibrium point P^+, and the dynamics of the D_1 region as a whole, can be understood by examining the matrix

$$\mathbf{J} = \begin{bmatrix} -\frac{R_0}{L} & \frac{1}{L} & 0 \\ \frac{1}{C_2} & -\frac{G}{C_2} & \frac{G}{C_2} \\ 0 & \frac{G}{C_1} & -\frac{\tilde{G}_b}{C_1} \end{bmatrix}$$

The stability of the D_1 region equilibrium point, P^+, is completely governed by the eigenvalues of the matrix $\mathbf{J}$. Or in other terms the zeros of the the characteristic equation, $\det(\lambda\mathbf{I} - \mathbf{J}) = 0$,

$$\lambda^3 + \left(\frac{G}{C_2} + \frac{\tilde{G}_b}{C_1} + \frac{R_0}{L}\right)\lambda^2 + \left(\frac{1 + GR_0}{LC_2} + \frac{\tilde{G}_b R_0}{LC_1} + \frac{GG_b}{C_1 C_2}\right)\lambda + \frac{GG_b R_0 + \tilde{G}_b}{LC_1 C_2} = 0 \tag{3.20}$$

Which solve to give us $\lambda_1 = \gamma_{P^+}$, $\lambda_2 = \sigma_{P^+} + j\omega_{P^+}$ and $\lambda_3 = \sigma_{P^+} + j\omega_{P^+}$. This leaves us in much the same position that we were with the inner region around Eqn 3.11, we can apply the same logic and process to yield the following

$$E^r(P^+) = \begin{bmatrix} \left(\gamma_{P^+} + \frac{G}{C_2}\right)\left(\gamma_{P^+} + \frac{\tilde{G}_b}{C_1}\right)\frac{C_1 C_2}{G} - G \\ \frac{C_1}{G}\left(\gamma_{P^+} + \frac{\tilde{G}_b}{C_1}\right) \\ 1 \end{bmatrix} = \begin{bmatrix} x_{P^+} \\ y_{P^+} \\ z_{P^+} \end{bmatrix} \tag{3.21}$$

and

$$\vec{\Psi}_{E^c(P^+)} = \begin{bmatrix} -\frac{LC_2}{G}\left(\gamma_{P^+} + \frac{G}{C_2}\right)\left(\gamma_{P^+} + \frac{\tilde{G}_b}{C_1}\right) + \frac{LG}{C_1} \\ \frac{C_2}{G}\left(\gamma_{P^+} + \frac{\tilde{G}_b}{C_1}\right) \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{L}{C_1}x_{P^+} \\ \frac{C_2}{C_1}y_{P^+} \\ z_{P^+} \end{bmatrix} \tag{3.22}$$

$\therefore$ as before we now know $\vec{\Psi}_{E^c(P^+)}$, the vector normal to the complex eigenplane $E^c(P^+)$, and we know a point, P^+, on the plane, we $\therefore$ can specify the $E^c(P^+)$ plane itself with ease

$$E^c(P^+) \Longrightarrow$$
$$\left(-\frac{L}{C_1}x_{P^+}\right)\left(x - P^+[1]\right) + \left(\frac{C_2}{C_1}y_{P^+}\right)\left(y - P^+[2]\right) + (z_{P^+})\left(z - P^+[3]\right) = 0 \tag{3.23}$$

$$\text{where } P^{+} = \begin{bmatrix} \frac{G(G_a - G_b)E}{G_b' + GG_bR_0} \\ \frac{GR_0(G_a - G_b)E}{G_b' + GG_bR_0} \\ \frac{(1+GR_0)(G_a - G_b)E}{\tilde{G}_b + GG_bR_0} \end{bmatrix}$$

We now have everything we need to describe, both qualitatively and quantitatively, the dynamics of a Chua's Oscillator trajectory in the D_1 region around the middle equilibrium point P^{+}. Consequently, due to symmetry about the origin we can specific dynamics of a Chua's Oscillator trajectory in the D_{-1} region around the middle equilibrium point P^{-}.

Now that we have the mathematical formulae that derive eigenvalues, eigenvectors and eigenplanes for each distinct region, as well as the state-equations which describe the system as a whole, we can begin to look at implementing and synthesizing these mathematical abstractions in a programming environment.

Part III

Software Implementation

4. Software Implementation

4.1 Design Restrictions

As mentioned in Section 2.1, the legacy of the original ABC is its valuable database of initial conditions and Chua Circuit parameters. One of the project's software design goals is to provide a portable framework that will breathe life back into this asset and ensure that it will not become unusable overtime, as it did when under ABC's custody. The other main software design goal was to provide a Graphical User Interface (GUI) framework to ease usability.

In the early stages of the project, it was contemplated designing ABC++ purely in C++ (or its new proprietary cousin C#). It was felt that this would give the optimal integration routine (recursive algorithm). The GUI was to be designed in C++/C# and the graphic stage, where the trajectories would be plotted, was to utilize OpenGL[1] (of SGI fame). Having investigated a set of OpenGL wrappers for C#[2], the Lorenz Paradigm was simulated to start with, see Figure 4.1.

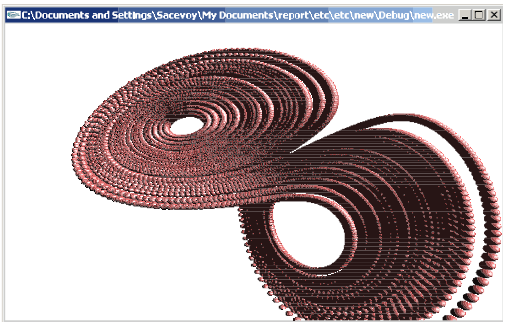

Figure 4.1: The Lorenz Attractor visualised with OpenGL

[1] Opensource Graphics Library ⇒ http://www.opengl.org
[2] These wrappers can be found at http://csgl.sf.net

This system would have worked but it failed one of ABC++'s core design restrictions, portability, as C# needs a *.Net* platform to run. This highlights the crux of the project's software design problem; finding a solution with the following parameters

- A GUI widget set[3].
- Low iteration latency[4].
- Fully portable (for us *NIX[5] users out there).

It is fair to say that, in general, programs can be divided into two sets $\Longrightarrow$ compiled programs and interpreted programs.

Compiled programs, written in languages such C, C++ and Fortran, offer very low iteration latency[6] and usually offer a widget set (think MFC[7] for C++). They are, however, very platform dependent and even architecture dependent, this hinders portability. Even if one were to distribute source code that the user could then compile on his/her platform there is still an issue, since widget sets are, by there very nature, tied to a specific Operating System[8].

Interpreted programs, written in languages such as Ruby, Python, Perl and Java, solve this platform dependency issue[9]. Again most of them offer widget sets, so why not use these? Well, like all things 'interpreted', these languages need some engine ('runtime environment') to convert your code into machine-code comprehendible to your computer processor. These 'runtime environments' are unfortunately unavailable in UCC.

Another point to note about interpreted languages, is the fact they tradeoff low iteration latency for portability. This is because interpreted languages

[3]The ability to create a graphical user interface. Examples of widgets are buttons, input boxes etc.

[4]Integration routines are recursive and $\therefore$ to be efficient require low iteration latency.

[5]This report itself was typeset with LaTeX on a Linux machine (2.4.7-10 kernel).

[6]Since they are compiled into machine code specific to you machine architecture.

[7]Microsoft Foundation Classes.

[8]There are some expensive platform independent widget sets, such as QT - http://www.trolltech.com

[9]Any interpreted language worth its salt is platform independent, that is to say, they provide a "runtime environment" for most platforms.

first need to be read by the 'run-time' engine then this 'run-time' engine needs to tell your processor what to do, not as efficient as compiled programs.

As is the case with most problems, the solution is often right under your nose and so it was with this problem. Matlab is the de-facto standard in computerized mathematical packages amongst engineers, and provides its own high level programming language (resembling C++) for mathematical calculation and visualization. Matlab is available for Windows & *NIX platforms (as well as multiple system architectures x86/Sparc/Risc). One can also be confident that most engineering/scientific college courses offer their students access to the Matlab package. As such, the marriage of ABC++ and Matlab was a match made in heaven.

4.2 Matlab

Matlab allowed ABC++ to be constructed using a modular design; it allowed the segmentation if the various key components, such as the integration routine, file routines, and graphic routines. Simulink, another component of Matlab, allowed Chua Circuit paradigm to be modeled using block diagrams, so the feasibility of using Matlab with the initial condition database could be tested.

From this initial feasibility testing, it was concluded that Matlab could in fact replicate all the functionality of the original ABC. Of course, Simulink would not be used in the final application for portability reasons (Application would require Simulink to run!) ... but was useful in understanding the Chua Circuit paradigm! See Appendix B.

Emulating the original ABC, with Matlab, means providing functionality to read the initial condition and system parameter database, and the functionality to calculate and draw the *exact same* equilibrium points, eigenvalues, eigenspaces, and trajectories as ABC would. The next section of this report talks a little more in detail about the software realization of these core components.

4.3 Core Software Components

For a visual appreciation of these core components and how they fit together within an GUI framework, see Figure C.1 in Appendix C.

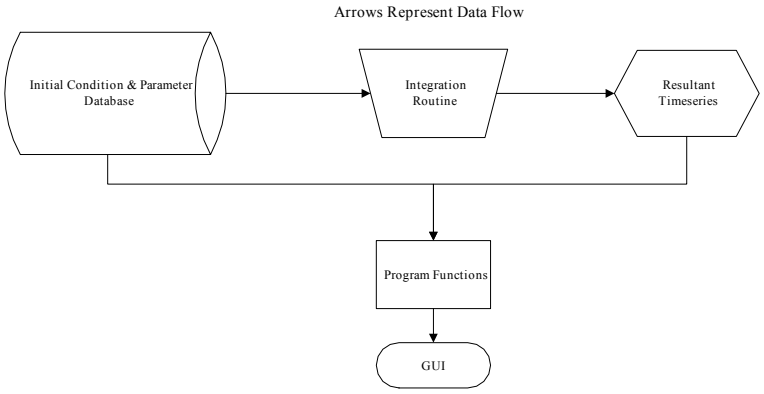

Figure 4.2: Dataflow chart depicting to flow of data in ABC++

4.3.1 Integration Routine

> "Physicists like to think that all you have to do is say, these are the conditions, now what happens next?" - *Richard P. Feynman*[4].

The importance of the integration routine in the context of our project cannot be underestimated, see Figure 4.2.

As can be seen, data which is used to generate the various graphics for the GUI can be piped from one of two sources the 'Initial condition & Parameter Database' or the 'Resultant Timeseries'. Since the 'Initial condition & Parameter Database' is the same for ABC & ABC++ the only datapool which must remain the same is the 'Resultant Timeseries' dataset. This dataset is generated by the integration routine, thus for this dataset to be the same in ABC & ABC++ the same integration routine must be used.

But, what is this Integration Routine and why is it needed? Glancing back at our set of three ordinary differential equations (Eqns 3.2,3.3 and 3.4) which define the Chua Circuit paradigm. And remembering we need to solve these over a finite amount of time to produce our desired *State-Space trajectory*, we note that this system of ordinary differential equations cannot be solved either implicitly or explicitly. The integration routine we must use is an iterative numerical algorithm, which can *approximate* the solution to the system of differential equations[16].

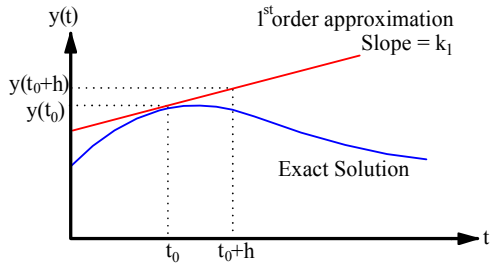

Figure 4.3: Visualization of Euler's Integration Method.

A point to note here is the fact that this integration routine is numerical rather than analytical, the importance of this is that if ABC & ABC++ are to have identical integration routines, ABC++ must use the same numerical resolution as ABC did. Thankfully the IEEE standard for double precision numbers has remained the same since 1993 when ABC was written, so numerical resolution is consistent[10].

ABC uses Runga-Kutta 4 and therefore so does ABC++, but in order to understand numerical integration methods[13] it is useful to investigate an easier routine, such as Euler's Method.

Euler's Method

In Figure 4.3, one can see how the time step, h, and the slope of the approximation line, $k_1 = \frac{dy(t_0)}{dt}$, are used to generate $y(t_0 + h)$ from $y(t_0)$.

$$y(t_0 + h) = y(t_0) + h \times k_1 \tag{4.1}$$

This is only a 1^{st} order approximation[11] and even for small time steps the error can be quite large.

Runga-Kutta

Runga-Kutta is a fourth order approximation[12] to the exact solution, at each step the derivative is evaluated four times: once at the initial point, twice

[10] Role on 64bit computing.
[11] See Chapter 19 of [13].
[12] See page 947 of [13].

at trial midpoints, and once at a trial endpoint. From these derivatives the final function value is calculated.

$$
\begin{aligned}
k_1 &= h \times f(x_n, y_n) \\
k_2 &= h \times f\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \\
k_3 &= h \times f\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \\
k_4 &= h \times f(x_n + h, y_n + k_3) \\
y(t_0 + h) &= y(t_0) + \frac{1}{6}\left(k_1 + 2k_3 + 2k_3 + k_4\right)
\end{aligned}
\tag{4.2}
$$

4.3.2 Equilibrium Points

The function designed to calculate the equilibrium points for a given system is an analytical one. We have equations, see Eqn 3.8, which specify exactly what the equilibrium points are. All that is required is to read in, from a file or GUI input box, specific circuit parameters for a given Chua's Circuit configuration and pass these values to our 'equilibrium points' function, which will calculate the equilibrium points. This 'equilibrium points' function will then return the systems equilibrium points to the GUI, so they can be plotted on ABC++'s graphics stage.

4.3.3 EigenValues, EigenVectors & Eigenplanes

If the integration rountine was critical in reproducing state-space trajectories from the original ABC's 'Initial condition & Parameter Database', then the 'EigenValues, EigenVectors & Eigenplanes' component is critical for reproducing the correct eigenplanes and eigenvectors which are so important to understanding the dynamics of said state-space trajectories.

Again this function is an analytical one which, involves solving for the roots of the characteristic equations (Eqns 3.11 & 3.20) for the inner and outer regions using Cardano's Method[12]. Once again this function takes the values stored in the correct input boxes as arguments and calculates the corresponding roots. As we know from Section 3.3, these roots are the eigenvalues for the inner and outer regions respectively.

Using these eigenvalues, we can easily solve for the associated eigenvectors using Eqns 3.16 & 3.21. Now that we have solved for these eigenvectors another simple computation returns the associated eigenplanes, see Eqns 3.18 & 3.23. Solving for these eigenplanes is a relatively simple procedure using the maths we derived in Section 3.3. However, plotting these planes in a coherent fashion over the state-space trajectory on the graphics stage proved to be quite tedious.

4.3.4 File Routines

The file routines created during development provides a gateway between ABC++'s core components and the 'Initial condition & Parameter Database'. But, what is this 'Initial condition & Parameter Database'? Well it is essentially about a hundred or so files, each one consisting of a set of circuit parameters and initial conditions, and each one meticulously fabricated[13] to produce a unique state-space trajectory.

Unfortunately these files, when created for the original ABC, were not structured in a fashion Matlab could work easily with. They contained a mixture of ASCII strings and numbers each one CRLF[14] delimited. However a simple Perl Script was capable of removing the ASCII strings, which simply labeled the proceeding number and were $\therefore$ redundant as long as we maintained a consistent structure across each file, and was able to replace the CRLF delimiters with comma delimiters. Thankfully, Matlab is more than happy to work with comma delimited files.

With this new file format writing and reading from files was simply a matter of using the Matlab DLMREAD function.

Along with the above ABC++ allows the user to save a graphic of his/her state-space trajectory, from the programs graphic stage, in jpeg format and, for LaTeX users, encapsulated postscript format.

4.3.5 Other Components

ABC++ contains other functionality, which was coded by Tomas, in addition to the components I coded above. These include the ability to work back

[13] In some cases significant to 20 decimal places.

[14] Carriage Return - Line Feed, See Matlab help for info on delimiters.

from user-defined eigenvalues to sample system parameters; this enables the user to predefine the trajectory dynamics he/she is looking for and work back to get sample system parameters which would result in those desired dynamics, very useful.

So as to get a feel for why this works consider the following $\Longrightarrow$ the shape of any state-space trajectory is governed by the systems eigenvectors & eigenplanes $\Longrightarrow$ these eigenvectors & eigenplanes are in turn governed by the systems eigenvalues $\Longrightarrow$ which are in turn governed by the specific circuit parameters for a given scenario.

Another additional component developed for ABC++ was the Poincaré Section, (see Chapter 5 of [15]). The Poincaré Section should be covered in detail in Tomas' report and is $\therefore$ only briefly mentioned here. The Poincaré Section takes a two-dimensional slice of the three-dimensional state-space trajectory and calculates the points where the trajectory intersects with this two dimension plane. It is used as a stroboscopic means of analyzing the dynamics of nonlinear systems.

A Fast Fourier Transform[17] component was also coded, primarily to exhibit a fundamental characteristic of chaotic systems, its broadband spectrum.

4.4 GUI Developement

> "Shéer plód makes plough down sillion shine" - *Gerard Manley Hopkins - The Windhover*

This aspect of the project was my main responsibility and as such, it will be discussed in suitable depth. As mentioned in Section 4.1, Matlab comes with a comprehesive widget set for creating graphical user interfaces, there are, however, a number of ways one can approach this.

Matlab comes kitted with a tool called GUIDE[15] for creating GUI's, however after toying around with GUIDE and being already experienced with other GUI development enviroments such as Visual C++, it was felt that GUIDE abstracted too much of the complexity required in building a comprehesive GUI (GUIDE is really only suitable for building simplistic GUI's). Fortunately one can code a GUI in Matlab from scratch using *ui-component*[16]

[15] **G**raphical **U**ser **I**nterface **D**evelopment **E**nvironment

[16] UserInterface components are the same as widgets i.e. an input box or sliderbar etc.

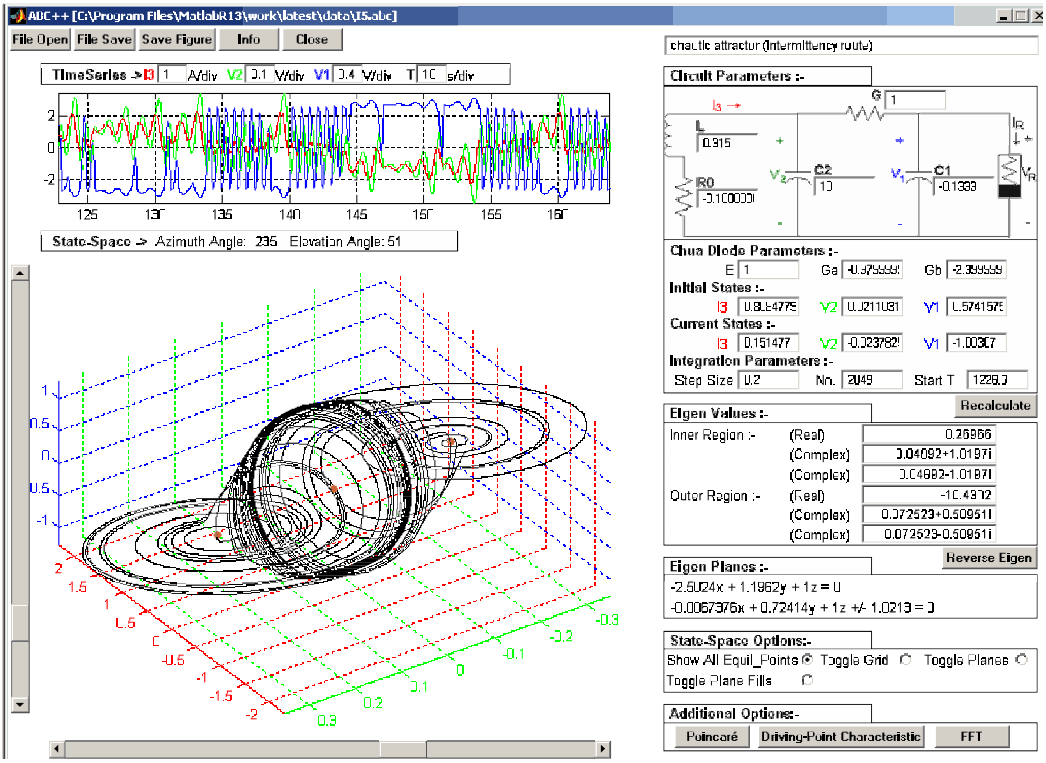

Figure 4.4: ABC++ Screen Shot.

functions directly. This gave the flexability to design a completely modular framework while retaining full control over every stage of the program, see Figure 4.4 where all ui-components were hand coded.

Designing a GUI in Matlab translates to designing an 'event-driven' GUI. A good way of appreciating how an 'event-driven' GUI operates is to imagine that behind every button you see is an 'event function', so when you click on a button an event is triggered calling the associated 'event function' into action. This function then uses data from some, or all, of the GUI's input boxes as arguments[17], it then performs actions on these arguments and returns the results to the main GUI, which in turn updates whichever input boxes or graphics that need to be updated. Care needed to be taken when working with said input boxes as all numerical data displayed must first be converted from numbers to strings, Matlab automatically rounds down numbers when

[17] Function arguments are simply inputs to the function.

converting to strings unless explicitly told not to! (This was a problem due to the sensitivity of chaotic systems to initial conditions).

Designing ABC++ with this modular, 'event-driven' framework allowed each component to be coded, tested and debugged individually, then, once each component was ready, knitted together into a complete application. Even though the application's 1500+ lines of code had to be hand generated, the design approach afforded just the right amount of leverage to get what was required done, done.

4.4.1 GUI Aesthetics

An important part of any GUI is to get it looking professional & tidy; this means careful planning to get the most from your GUI's screen presence[18] aswell as delivering the pertinent information to the end-user in a salient and concise manner. ABC++ uses fully compliant dialog boxes when saving & reading files from its host's hard-drive, its main graphical stage is rendered in 3D and can be viewed from any angle through the use of intuitive slider bars (see Figure 4.6) , and it uses a sublime blend of graphics, text and input boxes to highlight information most relevant to the end-user (see Figure 4.5). The user is also given the option to view the state-space trajectory with or without the eigenplanes[19], the user can also choose to zoom in on the trajectory or view the multi-region (D_{-1}, D_0 and D_1) area as a whole by simple toggling the 'Show All Equil_Points' radio-box.

See Appendix D, for a full tutorial on how to use ABC++.

[18] Using up screen area efficiently.

[19] Which can be filled planes or simply line planes.

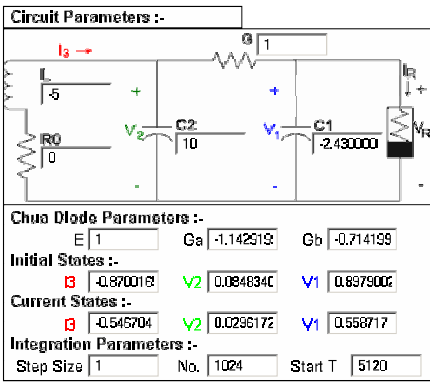

Figure 4.5: ABC++ uses graphics, text and input boxes to highlight information most relevant to the end-user.

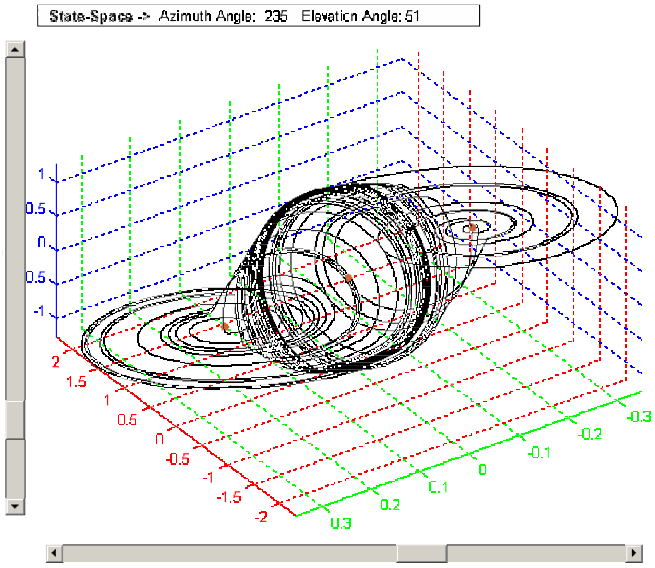

Figure 4.6: State-space trajectory rendered in 3D with slider-bars

Part IV

Project Discussion

5. Project Discussion

5.1 Comparison with other work

ABC++ can be compared directly with two other programs with the same objectives, ABC[6] and INSITE[7].

When compared to ABC, ABC++ fares extremely well, working with ABC++ it is possible to do *everything* the original ABC could, (calculates and draws equilibrium points, eigenvalues, eigenspaces and trajectories). In addition, ABC++ adds some *extra features* (reverse-eigenvalue and poincaré routines) and is far *easier to use*, thanks to its GUI.

ABC++'s visual aesthetics and interactivity excel compared to the surly, cumbersome ABC. Most importantly though, is the fact that ABC++ breathes new life into the valuable 'Initial condition & Parameter Database', all this plus ABC++ is completely portable and OpenSource[1].

INSITE[2] is an integrated suite of programs for the analysis of nonlinear dynamical systems. Its main advantages and disadvantages over ABC++ stem from the fact that it was designed or the analysis of nonlinear dynamical systems in general, and not the Chua Circuit Paradigm specifically. Like ABC++, it too can calculate trajectories and Poincaré maps. INSITE however, can take any dynamical system that can be described by state-equations (discrete- or continuous-time, autonomous or nonautonomous, of any (finite) order). ABC++ on the other hand uses only one set of state-equations (Eqns 3.2, 3.3 & 3.4), which are built in! However, ABC++ has the advantage that it is tailor made for a specific task, to model the Chua Circuit paradigm, and does it well. Also, INSITE does not offer the ability to calculate and draw eigenvectors & eigenplanes as ABC++ does.

[1] Released under GNU Public License, ABC++ is freely available for download at http://www.matlabcentral.com/

[2] **I**nteractive **N**onlinear **S**ystems **I**nvestigative **T**oolkit for **E**veryone.

Although I have not used INSITE (binary cannot be found anywhere); since it dates from before 1987, it is fair to assume that ABC++ has a better user interface than INSITE and is probably easier and more intuitive to use!

5.2 Critical Analysis & Future work

ABC++ has a lot of scope for future work, when evaluating the performance of ABC++ using a profiler[3] the integration routine[4] was found to be a substantial bottle-neck. This is could be overcome by considering the following; in Section 4.1 it was stated that for low iteration latency, code needs to by compiled, so in order to remove the bottle-neck the integration routine code needs to be compiled.

Thankfully, Matlab allows its programs to call compiled C code. These files are called *MEX-files*[5] and once compiled can be called directly as if they were a built in Matlab function. This would definitely eliminate the integration routine bottleneck!

Extra functionality could also be added to ABC++, it would be interesting if ABC++ could, like INSITE, take any system of state-equations. Another facet of INSITE which would complement ABC++ is its ability to reconstruct state-space trajectories from a single time series[18], something which is remarkable and would be fun to implement.

In addition, while ABC++'s core components are fully stable some of the addition components (reverse eigen, fft) are still only βeta quality, sometimes producing meaningless results. If these components could be fine-tuned it would add to the stability and usability of ABC++.

Some other ideas; add the functionality to 'listen to chaos' - the timeseries of the various attractors are within audio frequencies and Matlab can communicate with system speakers on a Windows machine. So why not output these timeseries to your system speakers and 'listen to chaos'.

What about decentralizing the 'Initial condition & Parameter Database', and make use of 21^{th} century technological advances. ABC, and ABC++ for

[3] A program which calculates the time taken for different segments of code.
[4] by its nature highly recursive.
[5] See Matlab Help.

that matter, use a local database[6], it would be interesting to utilize an online database from which people could download and upload trajectories. So if an user finds an interesting state-space trajectory, they could upload it so everybody else with ABC++ could download it! This functionality wouldn't be that hard to implement[7] and is certainly possible with Matlab.

5.3 Conclusion

ABC++ is a user-friendly GUI program, designed to run in a Matlab environment, geared towards the exploration of bifurcations and chaos within the Chua Circuit paradigm, and written on the stanchions of a modular programming framework. Its comes with an extensive database of sets of system parameters and initial conditions, which produces a wide variety of the dynamical behaviors reported for Chua's Circuit.

The main goal of the project was to emulate all the features of ABC[6], and to breathe life into the extensive database that came bundled with it. ABC++ more than delivers on these goals, it also adds some exciting new features, not to mention an easy to use interface.

In addition, since ABC++ was designed using a modular framework, the possiblities when adding extra functionality is endless, and since ABC++ is freely available at http://www.matlabcentral.com, anybody can download the sourcecode[8] and add features as they please.

[6] One stored on the computer's hard-drive.
[7] Especially for those of us with MySQL and PHP exposure.
[8] Which is very well commented.

6. Acknowledgements

Firstly, I would like to thank Professor Michael P. Kennedy for encouraging us to take on the project at the start of the year. Forever approachable and suggestive, Professor Kennedy was always keen to help with the project throughout the year, thank you.

I would also like to thank the members of the Linux community for providing me with a bottomless well, that has satisfied my thirst for knowledge these past years, and without whom typesetting this report with LaTeX would have been impossible.

Bibliography

[1] James Gleick, *"Chaos"*, Vintage, Random House, 20 Vauxhall Bridge Road, London SW1V 2SA, **ISBN:** 0749386061

[2] Simon Singh, *"Fermat's Last Theorem"*, Fourth Estate Ltd, 6 Salem Rd, London W2 4BU, **ISBN:** 1857026691

[3] Sarah Flannery, *"In Code"*, Profile Books Ltd, 58A Hatton Garden, London EC1N 8LX, **ISBN:** 1861972717

[4] Richard P. Feynman, *"The Pleasure of finding things out"*, Penguin Books, **ISBN:** 0140290346

[5] Edward N. Lorenz, "Deterministic Nonperiodic Flow", *Journal of Atmospheric Sciences*, 1963

[6] Michael P. Kennedy, "ABC : A Program for Studying Chaos", *Journal of the Franklin Institute*, vol. 331B, no. 6, pp. 631-658, 1994

[7] Thomas S. Parker and Leon O. Chua, "INSITE - A Software Toolkit for the Analysis of Nonlinear Dynamical Systems", *Transactions of the IEEE*, vol. 75, no. 8, pp 1081-1089, 1987

[8] B. Van der Pol and J. Van der Mark**,** "Frequency demultiplication"*, Nature*, vol. 120, no. 3019, pp. 363-364, 1927.

[9] Christopher P. Silva, "Shil'nikov's Theorem - A Tutorial", *IEEE Transactions On Circuits & Systems*, vol. 40, no. 10, pp. 675-682, 1993

[10] L.O. Chua, M. Komuro, and T. Matsumoto, "The Double scroll family: Part I and Part II", *IEEE Trans. Circuit Syst.*, vol CAS33, pp. 1072-1118, 1986

[11] Michael P. Kennedy, "Three Steps to Chaos - Part II: A Chua's Circuit Primer", *IEEE Transactions On Circuits & Systems*, vol. 40, no. 10, pp. 657-674, 1993

[12] See any good numerical algorithms book such as ... *"Numerical Recipes in C++".*

[13] Erwin Kreyszig, *"Advanced Engineering Mathematics",* 8^{th} Edition, John Wiley & Sons Inc, **ISBN:** 047133328X

[14] Anton·Rorres, *"Elementary Linear Algebra"*, 8^{th} Edition, John Wiley & Sons Inc, **ISBN:** 0471170526

[15] Nagle·Saff·Snider, *"Fundamental of Differential Equations",* 5^{th} Edition, Addison Wesley Longman, **ISBN:** 0201338688

[16] Burden·Faires, *"Numerical Analysis"*, 6^{th} Edition, Prindle Weber & Schmidt, **ISBN:** 0534382169

[17] Haykin·Van Veen, *"Signals and Systems",* 2^{th} Edition, John Wiley & Sons Inc, **ISBN:** 0471164747

[18] Thomas S. Parker and Leon O. Chua, *"Practical Numerical Algorithms for Chaotic Systems"*, Spinger-Verlag, **ISBN:** 0387966897

Appendix A. Logbook

A.1 Period 1

A.1.1 Mon 14^{th} Oct

This week work was spent for the majority defining what we hoped to achieve over the course of the project. In conference with Prof. Michael P. Kennedy, we concluded on emulating all the core features of the original ABC program bring back to life the valuable database of initial conditions and system parameters. Therefore, time was spent discussing what, if any, features would be added and it was agreed that ABC++ would be designed using a modular scalable framework. Of course some time was spent investigating and getting a feel for the old ABC whose binary we still had. This helped us put the project in context.

Of course, to equip ourselves with the necessary tools we spent a lot of time researching nonlinear/chaos theory in general.

A.1.2 Mon 21^{st} Oct

After receiving the paper written by Prof. Michael Kennedy on ABC, with gave me the system of Chua equations I began by modeling this in Simulink to get a grasp on the Chua Circuit paradigm. Simulink was an excellent tool for visualizing these Chua equations, it also allowed is to verify that Matlab could be used to reconstruct the various attractors from the 'Initial condition and system parameter' database.

Of course, this method abstracted the integration routine that was fundamental to the project and something we would have to create ourselves. Time was therefore spent investigating the various mathematical methods of numerical integration. Those that we focused on were the Euler and Runga-Kutta methods.

A.1.3 Tue 29^{th} Oct

Having familiarized ourselves with the desired objectives of the project, it was useful at this stage to formalize this. We sat down and defined that project scope as well as laying out a clear project plan with clear deadlines and specific deliverables.

At this stage we were still contemplating which software development environment we were going to use – C++ with OpenGL for straight Matlab development. C++ functions can be integrated into Matlab and time was spent investigating this – we consultant D. O'Raordain to see if this Matlab C++ extension could be installed on the computers we used.

A.1.4 Mon 4^{th} Oct

At the beginning of this week I coded the Lorenz paradigm in C++ (using Euler) and rendered this out in OpenGL. This process was relatively difficult as one had to code at a low abstraction the graphic routines (unlike Matlab). It was observed that this C++/OpenGL implementation wasn't portable (again unlike Matlab). It was at this stage, in consultation with Prof. Michael P Kennedy, it was decided to run with a pure Matlab implementation.

A.1.5 Mon 11^{th} Oct

This week the majority of me time was spent getting familiar with the Matlab scripting language; which is a high-level language that resembles C.

The best way to learn a programming language is to take a hands on approach. With this in mind I went about coding the engine for our timeseries graph generator. Starting with something relatively easy like this was useful for getting used to the Matlab environment.

A.1.6 Mon 18^{th} Oct

This week it was decided to tackle the main feature of the project; the integration routine.

I started out by designing an Euler integration engine having researching papers in the area. When this was completed it was clear that it would be

able to reconstruct the elusive trajectories from the 'Initial condition and system parameter Database'.

The ABC source code was investigated to reverse-engineer the Runga-Kutta integration routine that it used. This was then coded in Matlab.

A.1.7 Mon 25^{th} Oct

The Runga-kutta integration engine was finished this week and tested shows that it reconstructed the ABC trajectories, which came from initial conditions of up ten decimal place precision, perfectly.

An attempt was also made to encapsulate this chua.m function within a basic Matlab GUI which plotted the attractors in an animated sequence. While this GUI was basic, it did teach me the fundamentals of GUI design in Matlab (such as callback routines etc) this would be important for developing the final GUI.

A.1.8 Mon 2^{nd} Dec

This week was unfortunately a slow one (due to job interviews). I did however get a chance to code the DP Characteristic engine, which would plot the DP characteristic graph for a given set of system parameters.

A.1.9 Mon 9^{th} Dec

To conform to the project plan I lay down for myself I began this week coding the equilibrium point engine. This module was then plugged into the GUI framework, which would draw the equilibrium points, for a given set of system parameters, on the graphics stage.

The latter part of the week was spent modifying and improving the basic GUI I had developed and laying down a conceptualization for what we wanted the final design to look like!

A project review was also carried out setting goals for after Christmas. These include finishing the file routine engine, completing the GUI design and constructing the EigenPlane engine.

A.2 Period 2

A.2.1 Mon 16th Dec

This week was spent for the most part writing up the preliminary report; any other spare time was used in investigating how to build complex GUI in the Matlab environment using the GUIDE tool.

GUIDE is a GUI development environment, which enables you to drag UI components onto a stage to which you can assign callbacks.

A.2.2 Mon 6th Jan

The majority of this week was spent building up our desired GUI look; http://www.matlabcentral.com was hit quite a bit trying to understand how other people approached the job of building GUIs. Reverse engineering what others have done gives you a better understanding of how Matlab GUIs work.

A GUI framework was decided on, I choose not to use GUIDE as it is a bit simplistic and opted to code the GUI aesthetics by hands... it will mean more work but gives that fine tuning ability required when creating a complex GUI. The GUI will be controlled in an 'event-driven' fashion using a switch loop to assign which button has been pressed etc.

A.2.3 Mon 13th Jan

This week further work was done on the GUI, I felt that it would be good idea to get the file routines finished so as the up and coming GUI would have access the initial condition database.

Instead of using the old ABC file format, which used (Carriage Return-Line Feed) delimiters, I decided to change this to comma delimiters something Matlab is more comfortable working with. To do this I wrote a small Perl script, which parsed through the old database and created a new one, neat.

A.2.4 Mon 20th Jan

At this stage we have a usable GUI, you can read in a file from a dialog box which will read in the initial conditions and system parameters, pass these the integration routine which will they generate the resultant timeseries for

the three states and plot the statespace trajectory as well as the individual timeseries.

Tom is working furiously try to get the eigenvalue functionality working and we hope to incorporate this by next week.

A.2.5 Mon 27th Jan

This week the GUI is finally showing some real shape, I have come up with a graphical method of adjusting the Chua Circuit parameters, which was very tedious the code by is intuitive to use and very professional looking. Some of the attractors looked garbled, but when debugged it seems Matlab wanted to round down numbers when converting them to strings for use in the various GUI input boxes.

I also looked into save the various plots as graphic files such as jpeg etc.

A.2.6 Mon 3rd Feb

Tom's EigenPlane function unfortunately still has a bug in it so he asked me to have a look at his code (two eyes being better than one). Thinking that it might be a problem using Matlab's built in roots function (probably numeric), I coded up a quick cardano function to solve for the roots of the characteristic equations, and rewrote the eigenvalues function on my own. It now works and next week we hope to plug it into the GUI.

I also finished the functionality to save the various plots as graphic files, including encapsulated postscript. A good portion of the latter part of the week was taken up preparing for our upcoming seminar presentation, this meant doing up some slides and preparing a 5 min speech about progress to date.

A.2.7 Mon 10th Feb

Now that we have a working eigenvalue function in place, ABC++ looks like catching up very quickly on the original ABC. I investigated plotting the eigenplanes on the main statespace trajectory stage ... it was trickier that I presumed and was quite tedious. Simply for aesthetics I tried adding an alpha channel (transparency) to the plane fill colors, but try as I might I

couldn't get it working for no apparent reason ... so I left it and moved on to the next task.

A.2.8 Mon 17th Feb

This week we, by all accounts, finished the emulation to the original ABC, everything works that was in the original program, save a few bugs that need to be ironed out. This week we chatted with Professor Kennedy about what addition features we could add to ABC++. Tom has been working on a poincaré function for some time now and it was agreed to include that, also put on the table for consideration was a plot of the given systems Driving-Point characteristic, a Fast Fourier Transform function and the ability to work back for user defined eigenvalues to sample system parameters.

Coding the Driving Point characteristic plotting function was trivial and was completed. Tom focused on the reverse-eigen functionality.

A.2.9 Mon 24th Feb

This week was spent polishing up the GUI, at this stage it is prefect without some of the additional features, we furiously tried to remove all the bugs form the reverse-eigen function but it still seems to drive and reverse calculated trajectory off to infinity ... definitely something up here. I also had to design another GUI for Tom's pioncaré function ... I came up with using slider bars to change the plane coefficients in real time, but there still seems to be a bug in the intersection finder which returns some nonsensical results.

A.2.10 Mon 3rd Mar

Started writing up the Final report this week ... considering doing it up with LaTeX on my Linux box at home ... would look good - but very time consuming! I complied LyX on my machine so I could speed up the creation of formulae for use with LaTeX. When not writing up the report I tweaked with ABC++ just to get it looking prefect, and tryed to rid those bugs for the Openday.

A.2.11 Mon 10^{th} Mar

We had to curtail all development work this week, and focus solely on a good coherent presentation for our Openday. It was decided to use my laptop to walk through the project, we also contemplated having an interactive session we people could come in and actually use ABC++ for themselves.

Some posters had to be designed, as well as trying to iron out those last couple of bugs in the reverse-eigen and intersection finder functions.

A.2.12 Mon 17^{th} Mar

Report ... tedious report ....

Appendix B. Simulink Modeling

The Simulink model shown in Figure B.1, is a graphical representation of the system of three differential equations (Eqns 3.2, 3.3 & 3.4). The f(V1) sub-block is shown in Figure B.2.

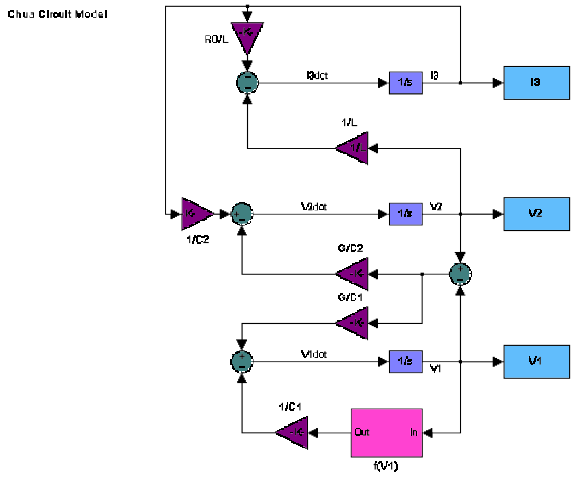

Figure B.1: Simulink Chua's Circuit Model with f(V1) sub-block.

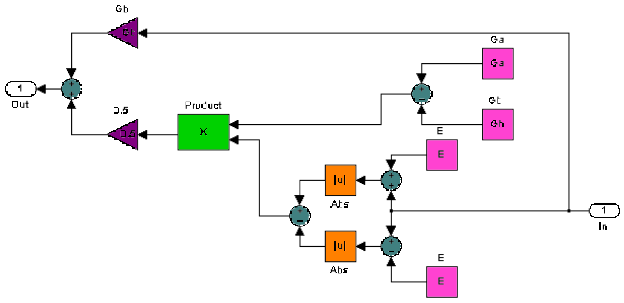

Figure B.2: The f(V1) sub-block.

Appendix C. GUI Conceptualization

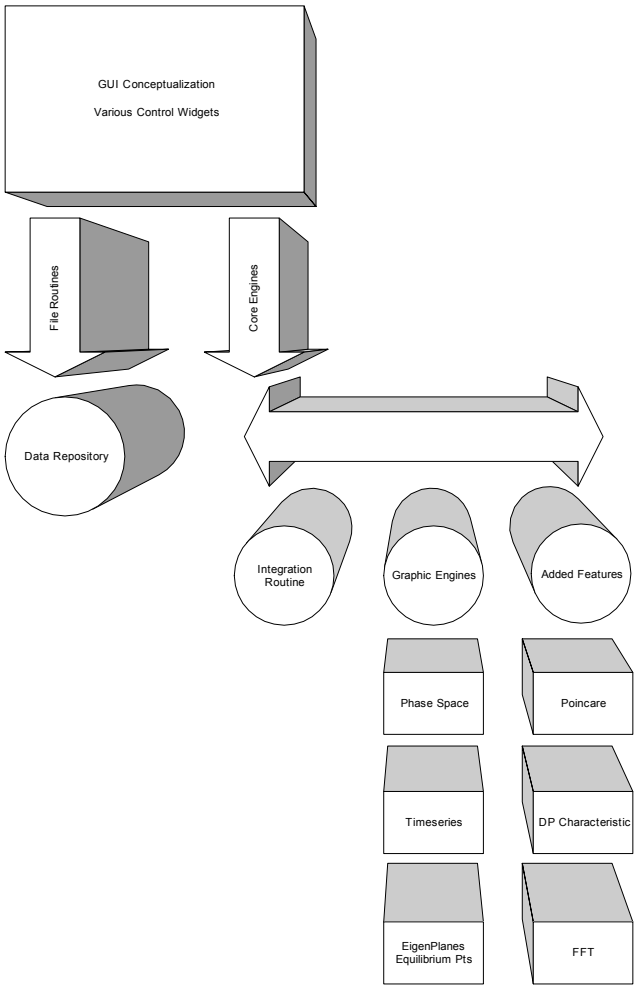

Figure C.1: Graphical User Interface Conceptualization.

Appendix D. ABC++: A Tutorial

> "It may happen that small differences in the initial conditions will produce very large ones in the final phenomena. A small error in the former produces an enormous error in the latter. Prediction becomes impossible ..." - Henri Poincaré.

Like any software program the best way to get a feel for how it works is to try it out for yourself, as such it is encouraged that the reader downloads the program from http://www.matlabcentral.com and gets it up and running himself.

The program comes in a zip file, which needs to be extracted to its own folder somewhere in the Mathlab path[1], once the following requirement is resolved running ABC++ in Matlab is simple a matter of typing **abc_pp** in the command prompt.

You should be then greeted with the GUI, as in Figure 4.4, the first thing you need to do is open a file. This is accomplished by clicking on the 'File Open' button. When opening a file the user is greeted with a dialog box that contains all the files in the 'Initial Condition & Parameter Database' ... pick any file.

After picking the file ABC++ calculates the state-space trajectory and each state's individual timeseries, and plots them for you! ABC++ also automatically calculates the Eigenvalues & Eigenplanes and displays them in a coherent fashion, see Figure D.1. A brief description for each of the hundred or so database files is also displayed in the GUI for user convenience.

The user can then change of fine tune the Chua Circuit parameters, as in Figure 4.5, and simply hit the 'Recalculate' button to see what his new attractor looks like. He can then save this new file back in the 'Initial Condition & Parameter Database' for later use by clicking on the 'File Save' button,

[1] See 'path environment variable', in Matlab help.

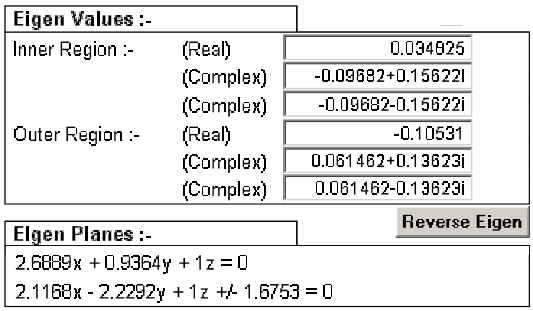

Figure D.1: EigenValues & EigenPlanes calculated automatically, also highlights reverse-eigen functionality.

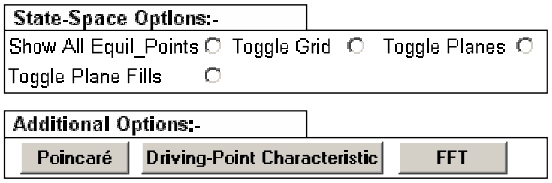

Figure D.2: Additional & State-Space Options.

or save the statespace plot as a graphic file by clicking on the 'Save Figure' button.

Any of the addition options available for ABC++ can be accessed by clicking on the relevant button in the 'Additional Options' frame, see Figure D.2. Go on try it out for yourself!