National Institute of Science Education and Research

Nonlinear dynamics experiments simulation

P441-P442

Roll no: 1811004

Abhishek Anil Deshmukh *2018-2023*

School of Physical Sciences

October 10, 2021

Abstract

Chua's circuits have shown applications in random number generation leading to applications in cryptography due to the quantity of rather unpredictable numbers it can produce based on just small variations in initial parameters and so they deserve a detailed study. Chua's circuit is a simple electronic network that well-known shows a selection of bifurcation phenomena and attractors. The circuit includes a couple of capacitors, an inductor, a linear resistor, and a nonlinear resistor. This report describes simulation techniques of such circuit and circuits in general.

Contents

List of Figures

Chapter 1

Introduction

Chua circuits are simple oscillator circuit which exhibits a variety of bifurcations and chaos. Chua's circuits. The circuit follows the following state equations

$$C_1 \frac{dv_{C_1}}{dt} = G(v_{C_2} - v_{C_1}) - g(v_{C_1})$$

$$C_2 \frac{dv_{C_2}}{dt} = G(v_{C_1} - v_{C_2}) + i_L$$

$$i_L \frac{di_L}{dt} = -v_{C_2}$$

Where,

$$g(v_R) = m_0 v_R + \frac{1}{2}(m_1 - m_0)[|v_R + B_p| - |v_R - B_p|]$$

- C_1: Capacitance of first capacitor C_2: Capacitance of second capacitor L: Inductance of the inductor
- i_L: current passing through the inductor
- v_{C_1}: voltage across first capacitor
- v_{C_2}: voltage across second capacitor
- G: reciprocal of Resistance
- B_p: breakpoint for the non-linear resistor
- m_0: slope after and before $\pm B_p$
- m_1: sloper between $\pm B_p$

Chapter 2

Numerical Solutions

State equations can be solved numerical as they are just a set of ordinary differential equations. Using Scipy the python scientific package for solving the equations, and plotting using plotly, I created a web interface to try different initial values, and different configurations of the values. Folowing is the code for non-linear resistor and state equations which will be integreated over using scipy.integrate.solveivp

```
 def g(m_0, m_1, v_r, B_p):
    B_p = abs(B_p)
    c = m_0 * B_p - B_p * m_1
    if v_r < -B_p:
        y = m_0 * v_r + c
    elif v_r > B_p:
        y = m_0 * v_r - c
    else:
        y = m_1 * v_r
    return y


def state_equations(t, V, G, C1, C2, L, m_0, m_1, B_p):
    """The State equations."""
    v_c1, v_c2, i_L = V
    d = G * (v_c1 - v_c2)
    d_v_c1 = -(d + g(m_0, m_1, v_c1, B_p)) / C1
    d_v_c2 = (d + i_L) / C2
    d_i_L = -v_c2 / L
    return d_v_c1, d_v_c2, d_i_L
```

Chapter 3

LT-Spice

Two simulation of Chua's Circuit which each utilization an alternate type of the nonlinear resistor are introduced beneath. The subsequent odd attractors that are delivered are comparative, yet somewhat unique. Regardless of whether you are new to utilizing LTspice, you will see it simple to do the simulations.

3.1 Simulation from a saved .net file

To assist the people who are new to LTspice, I will portray a method by which you can simulate Chua's Circuit utilizing circuit document (Appendix 1).

1. From the LTspice toolbar, select "File" and afterward "Open."
2. Go to the organizer containing the record .asc document and select it. The circuit outline should now be noticeable on your screen.
3. Select "Simulate" on the toolbar and afterward "Run."
4. Pick "V(v2)" and the voltage across C2 will show up as an element of time. (It resembles a lot of commotion.)
5. Put your cursor on the level pivot of the plot and left snap. A container will seem named "Horizontal Axis."
6. In that container is a line marked "Quantity Plotted." On that line "time" as of now shows up.
7. Change "time" to "V(v1)" and afterward click "OK." You currently should see your first bizarre attractor.

Recall that the factors for this framework are the voltage (v1) across C1, the voltage (v2) across C2, and the current I(L1) through the inductor. Any of these amounts plotted versus another gives an intriguing perspective on the peculiar attractor.

3.2 Plots

Varying resistance causes the following:

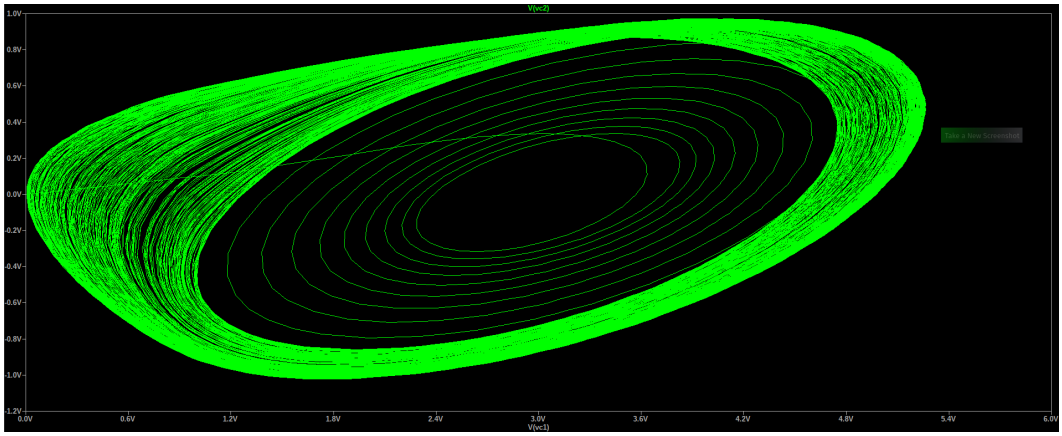

Figure 3.1: *R=1.92k*

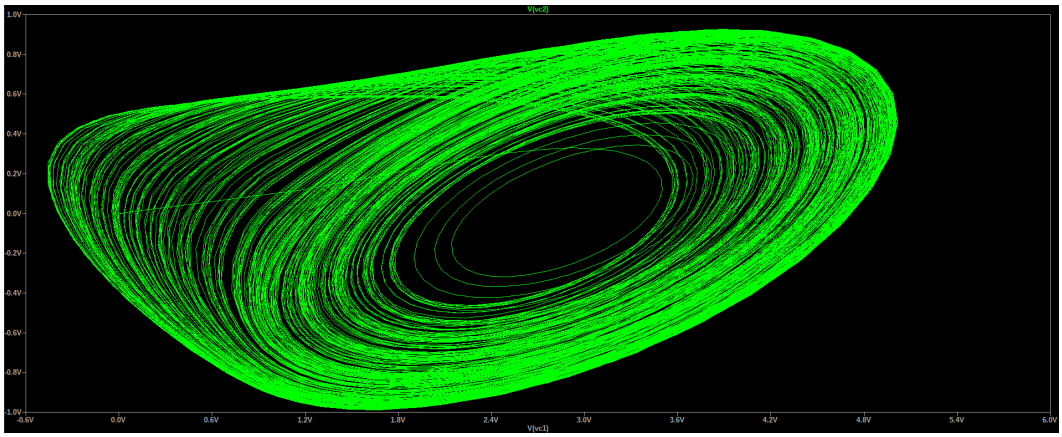

Figure 3.2: *R=1.90k*

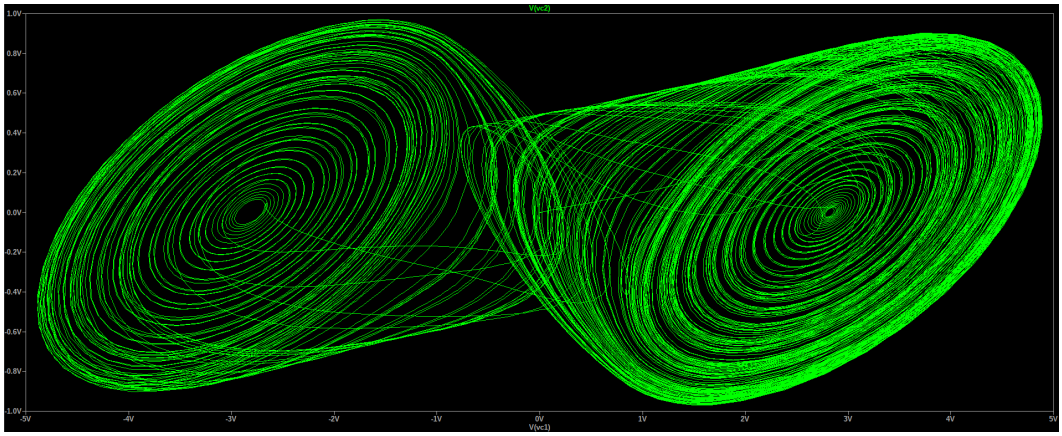

Figure 3.3: *R=1.89k*

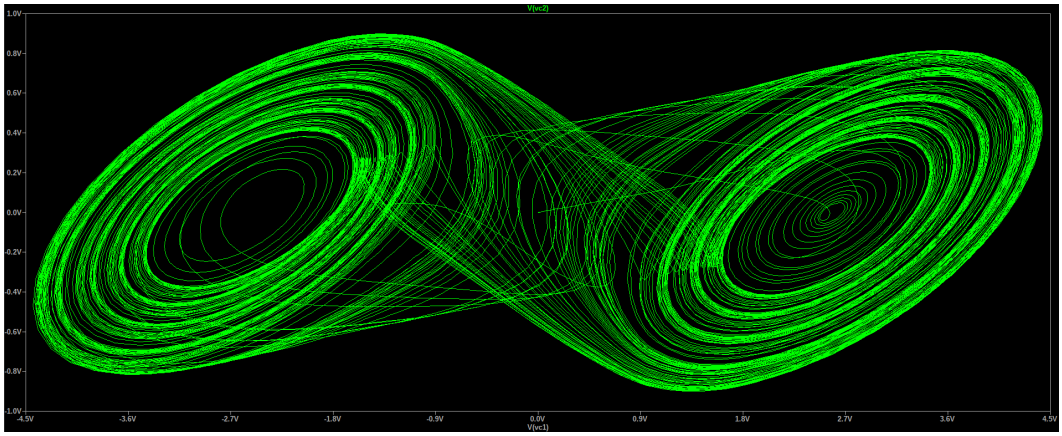

Figure 3.4: *R=1.85k*

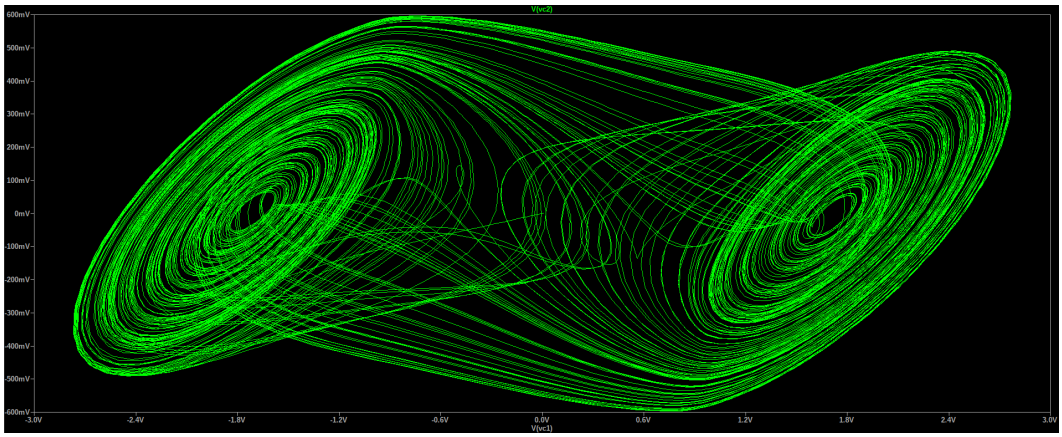

Figure 3.5: *R=1.65k*

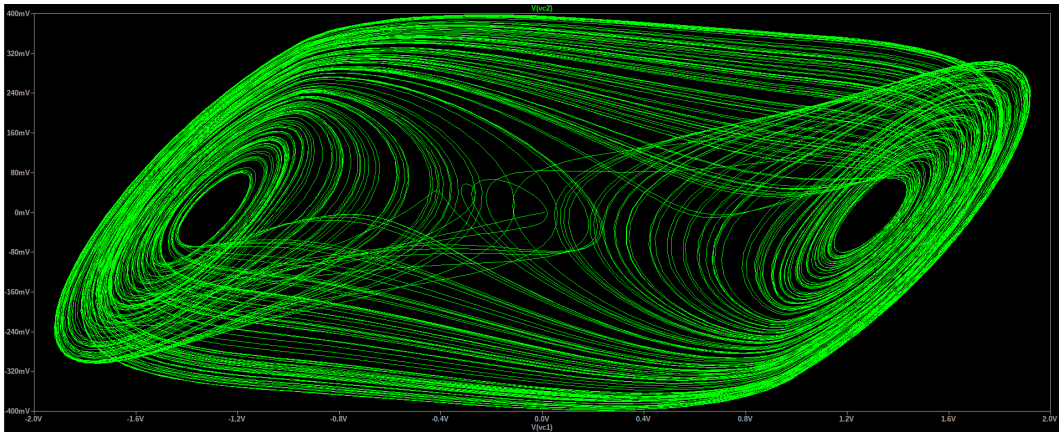

Figure 3.6: *R=1.50k*

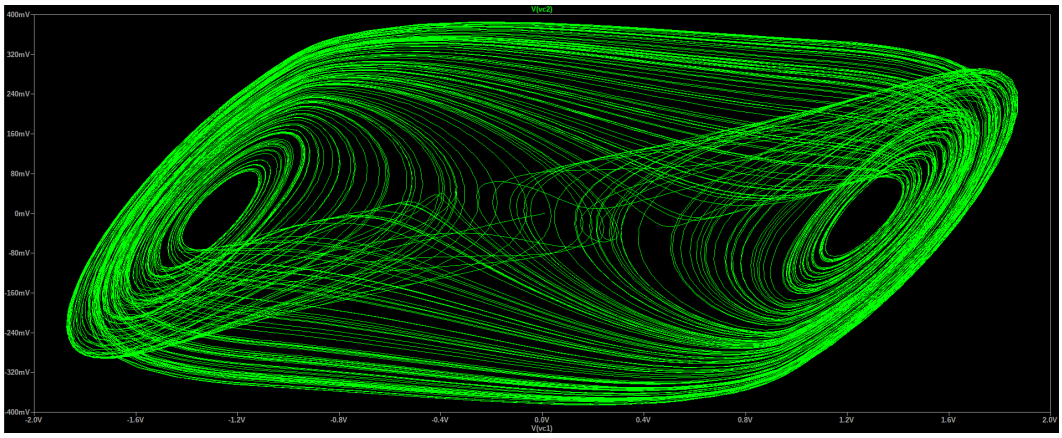

Figure 3.7: *R=1.49k*

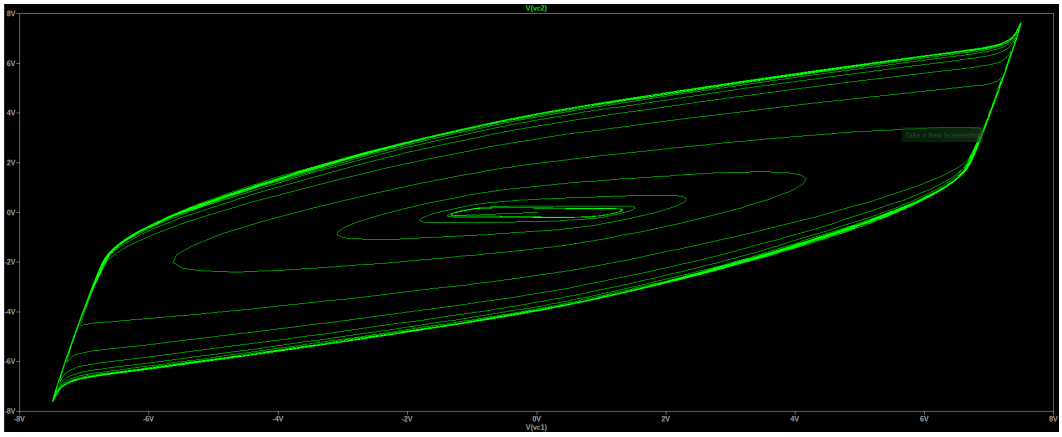

Figure 3.8: *R=1.39k*

Chapter 4

Conclusions

We looked at 2 methods of simulating chua's circuits, numerical and using LTspice. In the numerical method we used scipy to solve the state equations, the interactive plot can be found at Appendix A. When using LT-Spice, we simulated the circuit using net list. Varying values resistance different states of attractors, we go from stable state to double oscillator attractor to Edge case.

Appendices

Appendix A

Scipy and ploty code

A.1 main.py

```
from scipy.integrate import solve_ivp
from numpy import linspace
import plotly.graph_objects as go
from dash import Dash
from dash.dependencies import Input, Output
from layout import LAYOUT
from translator import phy_to_me


app = Dash(
    __name__,
    external_stylesheets=[
        "https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstrap.min.css"
    ],
)

app.layout = LAYOUT

preset_values = [
    [],
    [],
    [],
    [],
    [],
    [],
    [],
    [],
    [],
    [5.56, 50, 7.14, 0.7, 1, -0.5, -0.8],
]
```

```
def g(m_0, m_1, v_r, B_p):
    B_p = abs(B_p)
    c = m_0 * B_p - B_p * m_1
    if v_r < -B_p:
        y = m_0 * v_r + c
    elif v_r > B_p:
        y = m_0 * v_r - c
    else:
        y = m_1 * v_r
    return y


def state_equations(t, V, G, C1, C2, L, m_0, m_1, B_p):
    """The State equations."""
    v_c1, v_c2, i_L = V
    d = G * (v_c1 - v_c2)
    d_v_c1 = -(d + g(m_0, m_1, v_c1, B_p)) / C1
    d_v_c2 = (d + i_L) / C2
    d_i_L = -v_c2 / L
    return d_v_c1, d_v_c2, d_i_L


@app.callback(
    Output(component_id="chua-circuit-graph", component_property="figure"),
    Input(component_id="preset", component_property="value"),
    Input(component_id="v_c1(V)", component_property="value"),
    Input(component_id="v_c2(V)", component_property="value"),
    Input(component_id="i_L(mA)", component_property="value"),
    Input(component_id="tmax", component_property="value"),
    Input(component_id="steps", component_property="value"),
    Input(component_id="C1(nF)", component_property="value"),
    Input(component_id="C2(nF)", component_property="value"),
    Input(component_id="L(mH)", component_property="value"),
    Input(component_id="G(mS)", component_property="value"),
    Input(component_id="B_p(V)", component_property="value"),
    Input(component_id="m_0(mS)", component_property="value"),
    Input(component_id="m_1(mS)", component_property="value"),
)
def update_graph(preset, v_c1, v_c2, i_L, tmax, steps, C1, C2, L, G, B_p, m_0, m_1):

    if preset == 0:
        C1, C2, L, G, B_p, m_0, m_1 = phy_to_me(C1, C2, L, G, B_p, m_0, m_1)
    else:
```

```
        C1, C2, L, G, B_p, m_0, m_1 = phy_to_me(*preset_values[preset])

    # Integrate the equations to get the plot
    soln = solve_ivp(
        state_equations,
        (0, tmax),
        (v_c1, v_c2, i_L),
        args=(G, C1, C2, L, m_0, m_1, B_p),
        dense_output=True,
    )

    # Interpolate solution onto the time grid, t.
    t = linspace(0, tmax, steps)
    v_c1, v_c2, i_L = soln.sol(t)

    fig = go.Figure(
        data=go.Scatter3d(
            x=v_c1,
            y=v_c2,
            z=i_L,
            marker=dict(
                size=4,
                color=i_L,
                colorscale="Viridis",
            ),
            line=dict(color="darkblue", width=2),
        )
    )

    fig.update_layout(
        width=1000,
        height=900,
        autosize=False,
        scene=dict(
            camera=dict(
                up=dict(x=0, y=0, z=1),
                eye=dict(
                    x=0,
                    y=1.0707,
                    z=1,
                ),
            ),
            aspectratio=dict(x=1, y=1, z=1),
            aspectmode="manual",
        ),
```

```
    )
    return fig


if __name__ == "__main__":
    app.run_server(debug=True)
```

A.2 layout.py

```
import dash_html_components as html
import dash_core_components as dcc

# first double attractor
# C1, C2, L, G, B_p, m_0, m_1 = 0.00011 , 0.001, 140, 0.0007, 1, -0.5, -0.8
C1, C2, L, G, B_p, m_0, m_1 = 5.56, 50, 7.14, 0.7, 1, -0.5, -0.8

preset_value_options = [
        {"label": "none", "value": 0},
        {"label": "1-dot", "value": 1},
        {"label": "none", "value": 2},
        {"label": "none", "value": 3},
        {"label": "none", "value": 4},
        {"label": "none", "value": 5},
        {"label": "none", "value": 6},
        {"label": "none", "value": 7},
        {"label": "none", "value": 8},
        {"label": "9-Standard Double Attractor", "value": 9},
]


# initial condition
v_c1, v_c2, i_L = 1, 0, 0

# Maximum time point and total number of time points.
tmax, steps = 100, 10000


def input_div(name, default_value, step_length=0.01):
    return html.Div(
        className="input-group mb-3",
        children=[
            html.Span(className="input-group-text", children=name),
            dcc.Input(
                className="form-control",
```

```
                id=name,
                placeholder=name,
                type="number",
                value=default_value,
                step=step_length,
            ),
        ],
    )

def dropdown_div(name, default_value, options, clearable=False):
    return dcc.Dropdown(
                id=name,
                options=options,
                placeholder=name,
                value=default_value,
                clearable=clearable,
            )


inputs = [
    html.H4(children="Presets"),
    dropdown_div("preset", 9, preset_value_options),
    html.H4(children="Initial values"),
    input_div("v_c1(V)", v_c1),
    input_div("v_c2(V)", v_c2),
    input_div("i_L(mA)", i_L),
    html.H4("Circuit properties"),
    input_div("C1(nF)", C1),
    input_div("C2(nF)", C2),
    input_div("L(mH)", L),
    input_div("G(mS)", G),
    input_div("B_p(V)", B_p),
    input_div("m_0(mS)", m_0),
    input_div("m_1(mS)", m_1),
    html.H4("Simulation properties"),
    input_div("tmax", tmax, step_length=1),
    input_div("steps", steps, step_length=1),
]


LAYOUT = html.Div(
    className="container-fluid",
    children=[
        html.H1(
```

```
            className="text-center align-self-center",
            children="Chua circuit double attractor",
        ),
        html.P(
            className="lead text-center",
            children="Abhishek Anil Deshmukh (1811004)"
        ),
        html.Div(
            className="row",
            children=[
                html.Div(className="col-lg-4 offset-lg-1", children=inputs),
                html.Div(
                    className="col-lg-7",
                    children=[dcc.Graph(id="chua-circuit-graph")],
                ),
            ],
        ),
    ],
)
```

A.3 translator

```
# Units for variables
# c_1   F
# c_2   F
# l     H
# G     S
# B_p   V
# m_x   mS

def phy_to_me(c_1, c_2, l, G, B_p, m_0, m_1):
    c_1 = c_1 * 0.02
    c_2 = c_2 * 0.02
    l = l * 0.02
    return c_1, c_2, l, G, B_p, m_0, m_1

def me_to_phy(c_1, c_2, l, G, B_p, m_0, m_1):
    c_1 = c_1 / 0.02
    c_2 = c_2 / 0.02
    l = l / 0.02
    return c_1, c_2, l, G, B_p, m_0, m_1
```

Appendix B

chua.net