User Manual

Welcome to E2SCAPy! In this manual, the following topics are covered:

- Introduction.
- Installation of E2SCAPy.
- Considerations to take into account if you want to obtain a netlist from LTSPICE.
- Obtaining a Netlist from LTSPICE.
- Computing a circuit with E2SCAPy using the layered DDD method.
- Computing a circuit with E2SCAPy using the symbolic method of SymPy.
- Recommendations for obtaining a time-domain analysis, including an example with the Tow-Thomas biquad circuit.

 The reader is also invited to get in touch for recommendations, detected bugs, questions about the functionality, or collaborations through the GitHub contact or directly via email: lui.corl.ing@hotmail.com.

0.1. Introduction

E2SCAPy is an innovative and powerful library designed for the Python programming language, enabling the symbolic resolution of linear circuits. Unlike traditional methods that provide numerical values for voltage or current at a specific element or node, E2SCAPy offers detailed mathematical expressions that describe these quantities in the frequency domain. This approach provides a deeper and more comprehensive understanding of circuit behavior. Additionally, the method for transforming expressions from the frequency domain to the time domain is presented, expanding its usefulness and applicability in the analysis of dynamic systems and the study of transient responses.

This manual includes a series of carefully selected scripts that illustrate various examples of calculation and methods for visualizing results, thereby facilitating the learning and application process of E2SCAPy. These examples not only demonstrate the library's capabilities in terms of symbolic resolution and circuit analysis but also show how to effectively plot system responses. In addition to the examples provided here, a more extensive collection of practical examples is

available for users in the GitHub repository, which can be found at https://github.com/luisCorl. This repository is a valuable resource for any reader who wishes to fully explore and utilize the capabilities of E2SCAPy in various circuit analysis scenarios.

Furthermore, the given repository includes examples of the results shown in this thesis and additional examples such as the EAF model.

0.2. Installation

To use E2SCAPy, it can be obtained directly from the following URLs: https://pypi.org/project/ddd-layer/ To access the download for the DDD algorithm, and the following URL is to access the E2SCAPy algorithm: https://pypi.org/project/e2scapy/, The installation is done via the pip command. Additionally, if this is the first time you are installing it, it is recommended to install or check that the following packages are already installed:

- symengine
- memory_profiler
- sympy
- pandas
- numpy
- matplotlib

All of these libraries can also be obtained through the command *pip*, in the figure 1 shows the procedure for downloading several packages, including ddd_layer and E2SCAPy. Once the installation of all the required libraries is complete, you can perform an import test to ensure that everything is in order. The figure 1 shows the correct import or invocation of the library without any errors:

```
[7] pip install e2scapy

Requirement already satisfied: e2scapy in /usr/local/lib/python3.10/dist-packages (0.0.2)

[8] pip install symengine

Requirement already satisfied: symengine in /usr/local/lib/python3.10/dist-packages (0.11.0)

pip install ddd_layer

Requirement already satisfied: ddd_layer in /usr/local/lib/python3.10/dist-packages (0.0.2)

[12] pip install memory_profiler

Requirement already satisfied: memory_profiler in /usr/local/lib/python3.10/dist-packages (0.61.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from memory_profiler) (5.9.5)

[14] from e2scapy import e2scapy as sc
     #no olvides conectar con Google drive o asegurarte de que
     #se lee correctamente el archivo filtroRC.cir
     #/content/drive/MyDrive/Colab Notebooks/filtroRC.cir

     sc.MNAf("/content/drive/MyDrive/Colab Notebooks/filtroRC.cir")
     A,x,z = sc.formula_sympy()

DDD15V3
estamos verificando..
  elemento  nodo1 +  nodo2 - nodo3 + nodo4 -  miu miu2 miu3   t1   t2
0       R1        1        2     NaN     NaN  NaN  NaN  NaN  NaN  NaN
1       C1        2        0     NaN     NaN  NaN  NaN  NaN  NaN  NaN
2       V1        1        0     NaN     NaN  NaN  NaN  NaN  NaN  NaN
```

Figura 1: Downloading packages using the pip command

0.3. How to Draw a Circuit

Once you have everything needed to start working, it is recommended to have LTspice installed, or obtain it from the following URL: https://www.analog.com/en/resources/design-tools-and-calculators/ltspice-simulator.html Now, we will show how to obtain the symbolic equations for the present circuit, which is an inverting summing amplifier:

Step 1: Draw the circuit in LTspice as shown in the figure 2 shows item (a) on how to draw a circuit considering the number of nodes and remembering that all nodes connected to ground correspond to node number ”0”

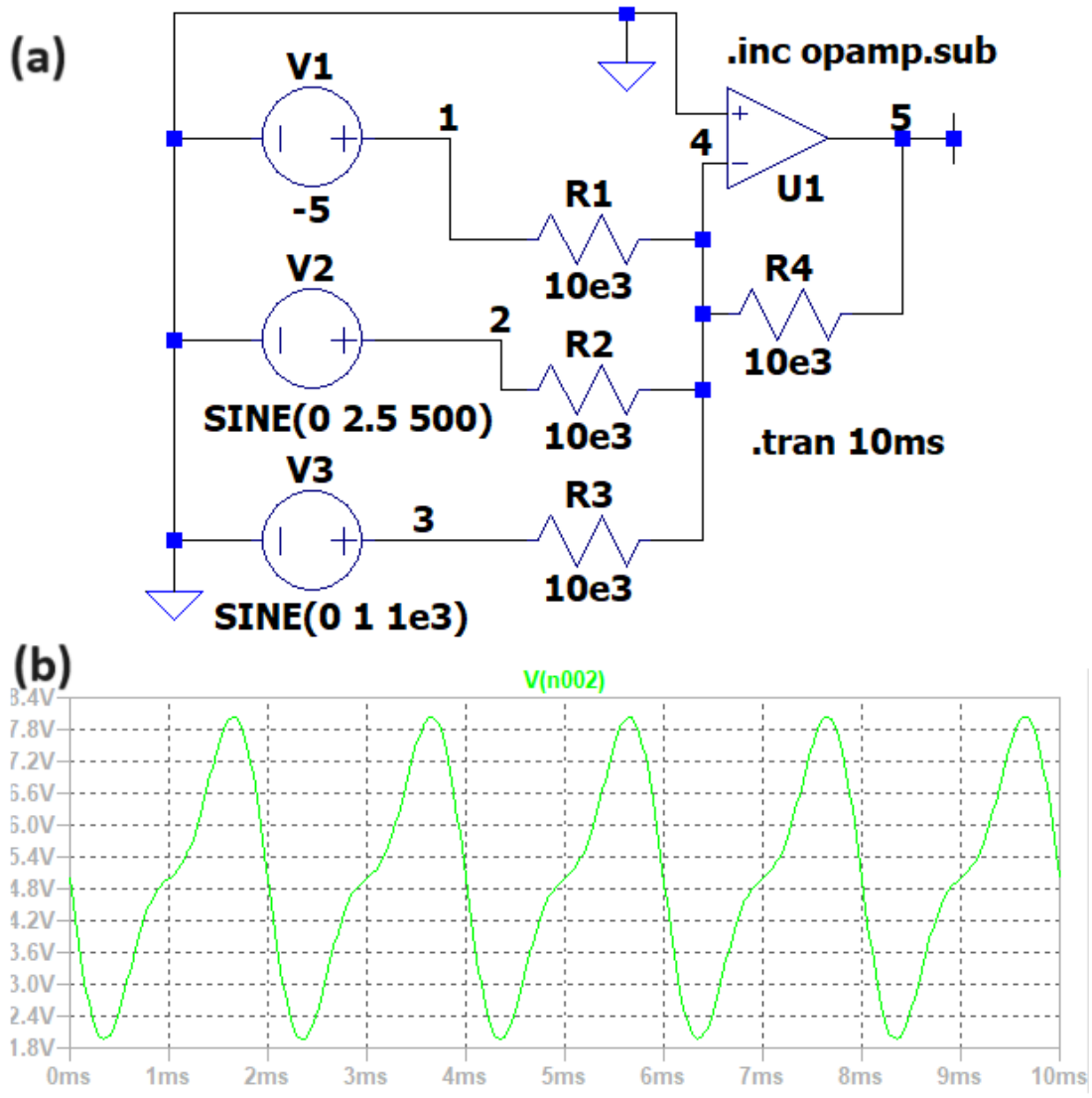

Figura 2: Inverting voltage summing amplifier circuit, (a) shows the circuit drawn in LTspice, (b) shows the output graph at node number 5

0.4. How to Obtain a Netlist from LTspice

To obtain the netlist from the figure 2 (a) To obtain the netlist for a circuit in LTspice, remember that all nodes to which passive components are connected must be numbered. Then, while in the schematic (not in the graph), click on View/SPICE Netlist as shown in the figure. 3

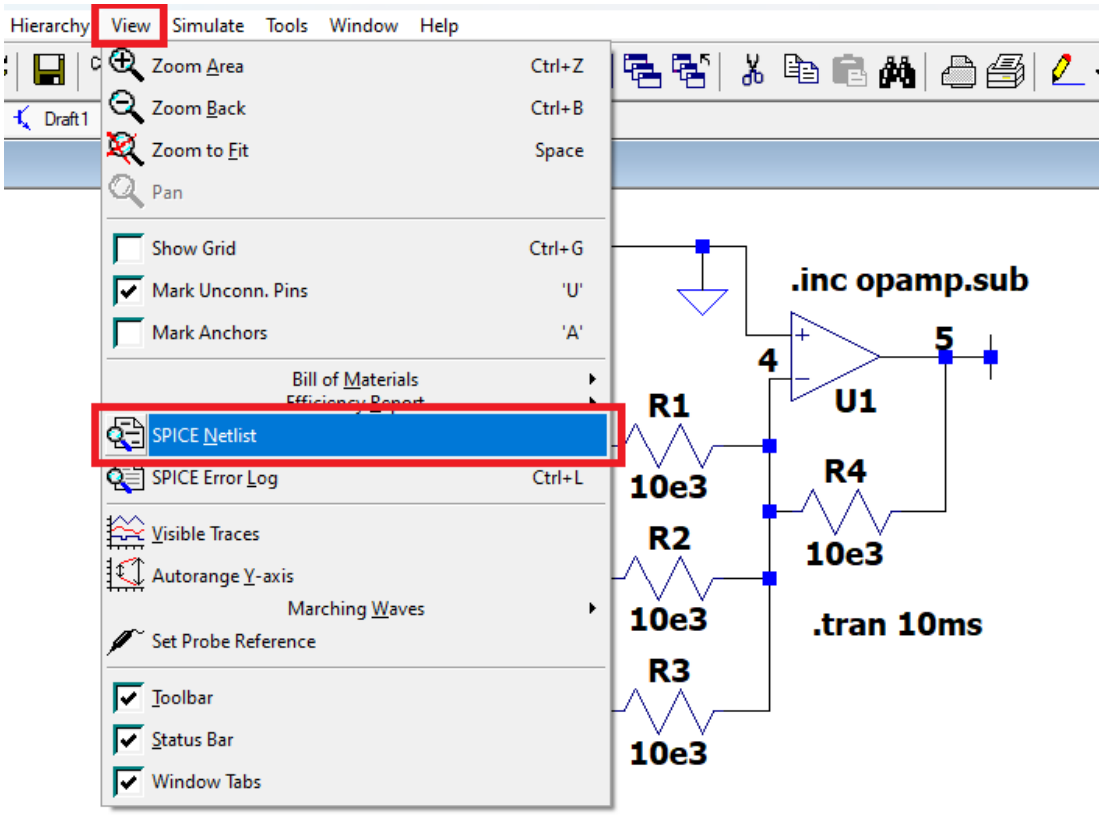

Figura 3: Obtaining a Netlist

The following window shows the netlist, but only the linear elements that make up the circuit should be selected, omitting .inc, .tran, .backanno, and .end, which are additional instructions for LTspice.

```
XU1 4 0 5 opamp Aol=100K GBW=10Meg
R1 4 1 10e3
R2 4 2 10e3
R3 4 3 10e3
R4 5 4 10e3
V1 1 0 -5
V2 2 0 SINE(0 2.5 500)
V3 3 0 SINE(0 1 1e3)
.inc opamp.sub
.tran 10ms
.backanno
.end
```

Figura 4: Netlist LTspice

0.5. How to Prepare a Netlist for E2SCAPy

This step involves saving the netlist in .cir format and naming the elements with a compatible format, keeping in mind that if you want to:

- OpAmp = O
- Resistor = R
- Inductor = L
- Capacitor = C
- Voltage Source = V
- Current Source = I
- Voltage-Controlled Voltage Source (VCVS)= E
- Voltage-Controlled Current Source (VCCS)= G
- Current-Controlled Voltage Source (CCVS)= H
- Current-Controlled Current Source (CCCS) = F
- Transformer = L1, L2, k1, k2
- Gyrator = J

In this way, the netlist is organized as shown in the figure 5:

```
R1 4 1 10e3
R2 4 2 10e3
R3 4 3 10e3
R4 5 4 10e3
O 4 0 5
V1 1 0 -5
V2 2 0 SINE(0 2.5 500)
V3 3 0 SINE(0 1 1e3)
```

Figura 5: Netlist for E2SCAPy

It is always recommended to organize the passive components (R, L, C) and place them in ascending order as shown in the figure. 5 and save the file with a .cir format

0.6. How to Compute a Circuit in E2SCAPy

It is recommended to use the following script as a base for performing symbolic computation using DDD, where the algorithm formulates the mathematical equations and solves the system of linear equations.

```
#importamos la librer a:
from e2scapy import e2scapy as sim

#leemos el archivo .cir con el respectivo
#nombre asignado
sim.MNAf("sum_inv.cir")
#se procede a hacer la formulaci n matematica
#con DDD por capas
A,x,z = sim.formula_DDD()
#se le pide al algoritmo DDD que resuelva
#el sistema de ecuaciones lineales
X = sim.resuelve_serie_DDD(A,x,z)
X = sim.simplifica(X)
```

The node of interest is node 5 with respect to the electrical circuit shown in the figure. 2 Therefore, you can request information about node 5 from the terminal, located in the vector X[4] as shown in the figure. 6

```
tiempo de calculo en serie por DDD:  0.03325080871582031
>>> X[4]
R1·R2·V3 + R1·R3·V2 + R2·R3·V1
-------------------------------
    R1·R2 + R1·R3 + R2·R3
>>>
```

Figura 6: Output at node 5 using the DDD method as shown in the figure 2

It is recommended to use the following script as a base for performing symbolic computation using SymPy, where the algorithm formulates the mathematical equations and solves the system of linear equations.

```
#importamos la librer a:
from e2scapy import e2scapy as sim

#leemos el archivo .cir con el respectivo
#nombre asignado
sim.MNAf("sum_inv.cir")
#se procede a hacer la formulaci n matematica
#con DDD por capas
A,x,z = sim.formula_sympy()
#se le pide al algoritmo GE que resuelva
#el sistema de ecuaciones lineales
X = sim.resuelve_LU(A,x,z)
#soluciones adicionales:
#   X = sim.resuelve_ADJ(A,x,z)
#   X = sim.resuelve_GE(A,x,z)
X = sim.simplifica(X)
```

```
tiempo de calculo en sympy LU:  5.371159791946411
>>> X[4]
R1·R2·V3 + R1·R3·V2 + R2·R3·V1
──────────────────────────────
    R1·R2 + R1·R3 + R2·R3

>>>
```

Figura 7: Output at node 5 using the LU method as shown in the figure 2

0.7. Time-Domain Analysis

In the case of having elements that contain information in the frequency domain directly from the MNA formulation, such as the capacitor C_n and the inductor L_n The result will be in the frequency domain, which allows proper evaluation in that domain. However, in many analyses, it is important to know the symbolic, semi-symbolic, or numerical result in the time domain. To do this, we will analyze a Tow-Thomas biquad circuit in the time domain. For this purpose, the following circuit is prepared:

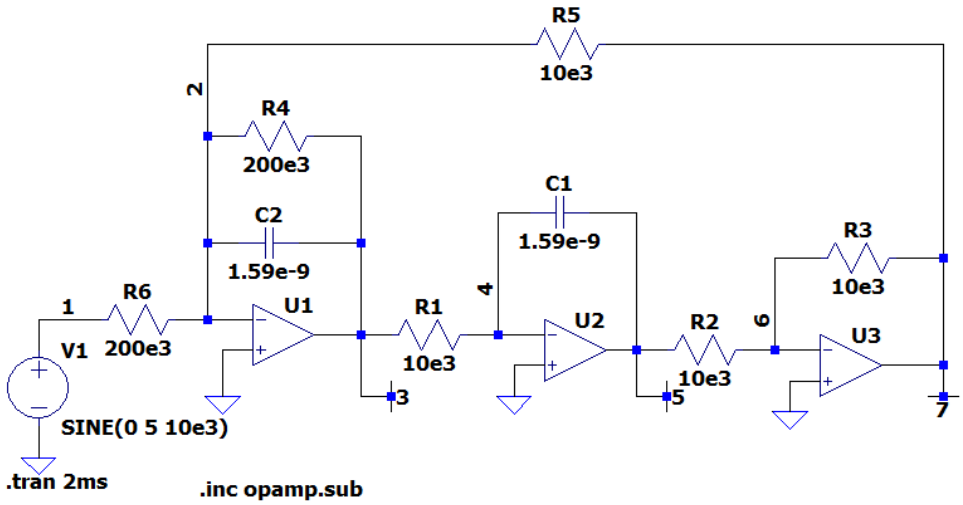

Figura 8: Tow-Thomas biquad circuit

The nodes of interest are nodes 3, 5, and 7. The netlist is prepared as follows:

```
O1 2 0 3
O2 4 0 5
O3 6 0 7
R1 4 3 10e3
R2 6 5 10e3
R3 7 6 10e3
C1 5 4 1.59e-9
C2 3 2 1.59e-9
R4 3 2 200e3
R5 7 2 10e3
R6 2 1 200e3
V1 1 0 SINE(0 5 10e3)
```

The current netlist is saved in .cir format, and the script is prepared by choosing the solution method using DDD:

```
#importamos la librer a:
from escapy import scapy as sim

#leemos el archivo .cir con el respectivo
#nombre asignado
sim.MNAf("TOW THOMAS.cir")
#se procede a hacer la formulaci n matematica
#con DDD por capas
A,x,z = sim.formula_DDD()
#se le pide al algoritmo DDD que resuelva
#el sistema de ecuaciones lineales
X = sim.resuelve_serie_DDD(A,x,z)
X = sim.simplifica(X)
```

In the terminal, transfer functions are prepared by dividing the information from the vector. $H3 = X[2]/V1$, $H5 = X[4]/V1$ y $H7 = X[6]/V1$ as shown in the figure:

```
tiempo de calculo en serie por DDD:  0.5699453353881836
>>> x
    [[V_1]
     [V_2]
     [V_3]
     [V_4]
     [V_5]
     [V_6]
     [V_7]
    [Iv_1]
   [I_OAmp_1]
   [I_OAmp_2]
   [I_OAmp_3]]
>>> print(X[2]/X[0])
C1*R1*R2*R4*R5*s/(R6*(-C1*R1*R2*R5*s*(C2*R4*s + 1) - R3*R4))
>>> print(X[4]/X[0])
R2*R4*R5/(R6*(C1*R1*R2*R5*s*(C2*R4*s + 1) + R3*R4))
>>> print(X[6]/X[0])
R3*R4*R5/(R6*(-C1*R1*R2*R5*s*(C2*R4*s + 1) - R3*R4))
```

Figura 9: Transfer functions for nodes H3, H5, and H7

Based on the transfer functions, it is recommended to use the Laplace transform and the inverse Laplace transform in MATLAB, as they often provide better convergence for large algebraic equations.

```
clc; clear all;
syms s t

R1 = 10e3
R2 = 10e3
R3 = 10e3
R4 = 200e3
R5 = 10e3
R6 = 200e3
C1 = 1.59e-9
C2 = 1.59e-9
```

```
12
13 H3 = C1*R1*R2*R4*R5*s/(R6*(-C1*R1*R2*R5*s*(C2*R4*s + 1) - R3*R4))
14 H5 = R2*R4*R5/(R6*(C1*R1*R2*R5*s*(C2*R4*s + 1) + R3*R4))
15 H7 = R3*R4*R5/(R6*(-C1*R1*R2*R5*s*(C2*R4*s + 1) - R3*R4))
16 w = 2*3.1416*10e3
17 vin = 5*sin(w*t)
18 vin = laplace(vin,t,s)
19 v3s = vin*H3;
20 v5s = vin*H5;
21 v7s = vin*H7;
22 v3t = ilaplace(v3s,s,t)
23 v5t = ilaplace(v5s,s,t)
24 v7t = ilaplace(v7s,s,t)
```

As can be observed, it is important to transform the excitation source to the frequency domain in order to obtain the total voltage in the frequency domain and then calculate the inverse Laplace transform.

To plot all of this in Python, you can follow the procedure outlined in the script presented below:

```
1  from matplotlib import pyplot as plt
2  import numpy as np
3
4  # Generar se al de entrada V1
5  T = np.linspace(0, 0.002, 600)
6  V_1 = []
7  ampl = 0.5
8  f = 10e3
9  w = 2 * np.pi * f
10 for t in T:
11     tempo = ampl * np.sin(w * t)
12     V_1.append(tempo)
13
14
15 from numpy import sin, cos, exp, pi
16
17 def fv3(t):
18     v3 = (25177073981803590908441542337484240471164190720*exp
       (-(4611686018427387904*t)/2933032307719819)*(cos((68719476736*159**(1/2)
       *1144999672862302388011**(1/2)*t)/466352136927451221) +
       (9662018248046651403629057812725 76*159**(1/2)*1144999672862302388011**(1/2)
       *sin((68719476736*159**(1/2)*1144999672862302388011**(1/2)*t)
       /466352136927451221))/160181680102909130250247911627102837954882 01))
       /12975111684348524948804390054269758351775182933 7 -
       (64777702983050847111369040606228680223204835328 0*sin(62832*t))
       /12975111684348524948804390054269758351775182933 7 -
       (25177073981803590908441542337484240471164190720*cos(62832*t))
       /12975111684348524948804390054269758351775182933 7
19     return v3
20
21 def fv5(t):
22     v5 = (25201549726898354590419758006460114534400000000*sin(62832*t))
       /12975111684348524948804390054269758351775182933 7 -
       (64840676247822732549220843889814468034560000000 0*cos(62832*t))
       /12975111684348524948804390054269758351775182933 7 +
```

```
    (648406762478227325492208438898144680345600000000*exp
    (-(4611686018427387904*t)/2933032307719819)*(cos((68719476736*159**(1/2)
    *1144999672862302388011**(1/2)*t)/466352136927451221) -
    (146137487871216261357315111992138518 81*159**(1/2)
    *1144999672862302388011**(1/2)*sin((68719476736*159**(1/2)
    *1144999672862302388011**(1/2)*t)/466352136927451221))
    /450746218921560165635904159781495586339100454551552))
    /129751116843485249488043900542697583517751829337
    return v5
def fv7(t):
    v7 = (648406762478227325492208438898144680345600000000*cos(62832*t))
    /129751116843485249488043900542697583517751829337 -
    (252015497268983545904197580064601145344000000000*sin(62832*t))
    /129751116843485249488043900542697583517751829337 -
    (648406762478227325492208438898144680345600000000*exp
    (-(4611686018427387904*t)/2933032307719819)*(cos((68719476736*159**(1/2)
    *1144999672862302388011**(1/2)*t)/466352136927451221) -
    (146137487871216261357315111992138518 81*159**(1/2)
    *1144999672862302388011**(1/2)*sin((68719476736*159**(1/2)
    *1144999672862302388011**(1/2)*t)/466352136927451221))
    /450746218921560165635904159781495586339100454551552))
    /129751116843485249488043900542697583517751829337
    return v7

# Inicializar listas para V3, V5 y V7
V3 = []
V5 = []
V7 = []
# C lculo del modelo
for t in T:
    v3 = fv3(t)
    v5 = fv5(t)
    v7 = fv7(t)

    V3.append(v3)
    V5.append(v5)
    V7.append(v7)
# Graficar resultados
fig, axs = plt.subplots(1, 1, figsize=(11, 3))
axs.xaxis.set_major_formatter(plt.FormatStrFormatter('%.3f'))
axs.plot(T, V3, color="green", label="v3")
axs.plot(T, V5, color="blue", label="v5")
axs.plot(T, V7, color="red", label="v7")
axs.set_title("Input voltage")
axs.set_xlabel("time s")
axs.set_ylabel("Volts")
axs.legend(loc="lower right")
axs.grid()

plt.show()
```

A continuación se muestra el gráfico de salida

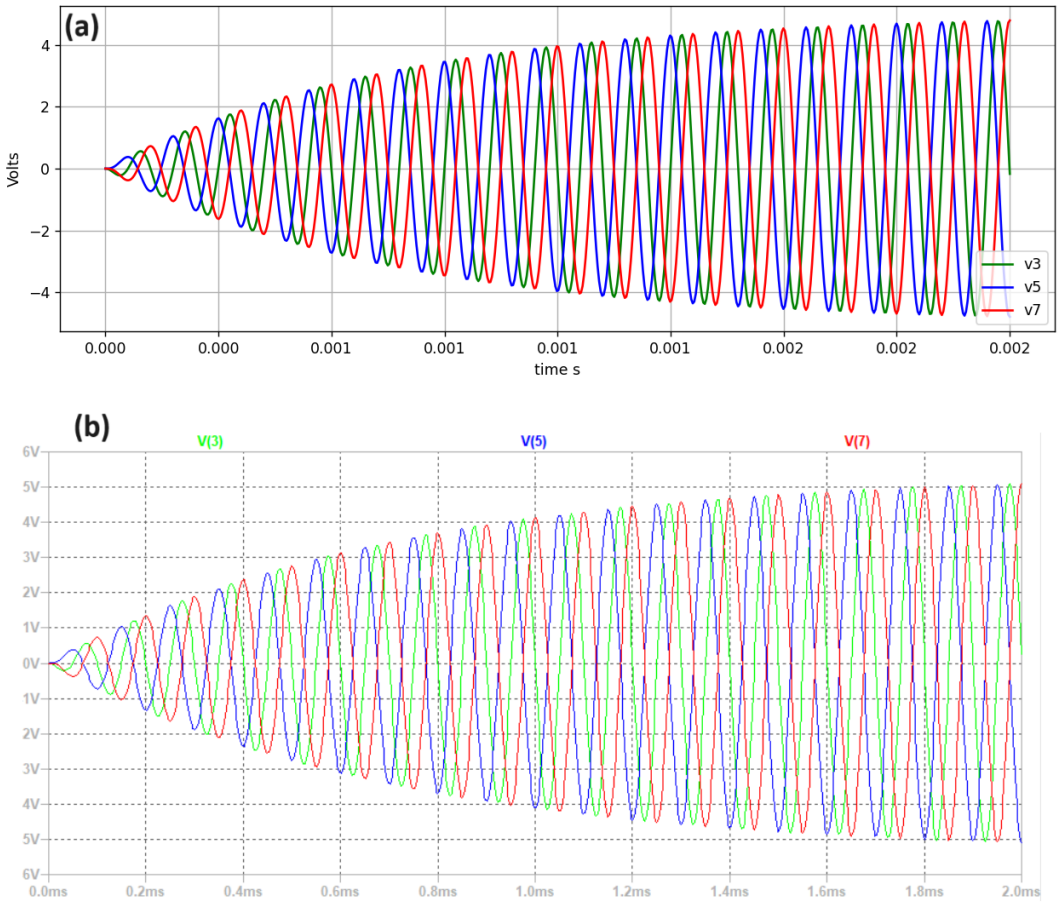

Figura 10: Comparative graph between the graph obtained with the Python script (a) and the graph obtained in LTspice (b)