Manual de usuario

¡Bienvenido a E2SCAPy!. En el presente manual se borda:

- Introducción.
- Instalación de E2SCAPy.
- Consideraciones a tener en cuenta, si se desea obtener un netlist a partir de LTSPICE.
- Obtener un Netlist desde LTSPICE.
- Computar un circuito con E2SCAPy a partir del método DDD por capas.
- Computar un circuito con E2SCAPy a partir del método simbólico de SymPy.
- Recomendaciones si se desea obtener el análisis en el dominio del tiempo con una ejemplificación con el circuito bicuadrado de Tow-Thomas.

SCAPy cuenta con una licencia GNU GPLV3 que puede ser consultada desde el repositorio: https://github.com/luisCorl/e2scapy/blob/main/LICENSE.txt y que no se coloca en este manual por exención de hojas, pero se invita al lector a revisar si así lo desea. También se invita al lector ponerse en contacto por recomendaciones, bugs detectados, preguntas sobre el funcionamiento y o colaboraciones a través del contacto de GitHub o directamente en el correo: lui.corl.ing@hotmail.com.

0.1. Introducción

SCAPy es una librería innovadora y potente diseñada para el lenguaje de programación Python, que permite la resolución simbólica de circuitos lineales. A diferencia de los métodos tradicionales que proporcionan valores numéricos para el voltaje o la corriente en un elemento o nodo específico, E2SCAPy ofrece expresiones matemáticas detalladas que describen estas magnitudes en el dominio de la frecuencia. Esto proporciona una comprensión más profunda y completa del comportamiento del circuito. Además, se presenta el método de transformar las expresiones del dominio de la frecuencia al dominio del tiempo, lo que amplía su utilidad y aplicabilidad en el análisis de sistemas dinámicos y en el estudio de respuestas transitorias.

En este manual se incluyen una serie de scripts cuidadosamente seleccionados que ilustran diversos ejemplos de cálculo y métodos de visualización de resultados, facilitando así el proceso de aprendizaje y aplicación de E2SCAPy. Estos ejemplos no solo demuestran las capacidades de la librería en términos de resolución simbólica y análisis de circuitos, sino que también

muestran cómo graficar las respuestas del sistema de manera efectiva. Además de los ejemplos proporcionados aquí, se ha preparado una colección más extensa de ejemplos prácticos que están disponibles para los usuarios en el repositorio de GitHub, que se puede encontrar en GitHub.https://github.com/luisCorl Este repositorio es un recurso valioso para cualquier lector que desee explorar y utilizar plenamente las capacidades de E2SCAPy en diversos escenarios de análisis de circuitos.

Además en el repositorio dado se pueden encontrar los ejemplos de los resultados mostrados en esta Tesis y algunos ejemplos adicionales como el modelo EAF.

0.2. Instalación

Para poder usar SCAPy se puede obtener directamente de las siguientes URL: https://pypi.org/project/ddd-layer/ para acceder a la descarga del algoritmo DDD, y la siguiente URL es para acceder al algoritmo SCAPy: https://pypi.org/project/e2scapy/, la forma de hacer la instalación es por el comando pip, adicionalmente si es la primera vez que se instala se recomienda instalar o revisar que se tengan las siguientes paqueterías previamente instalas:

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

Todas estas librerías también pueden ser obtenidas a través del comando *pip*, en la figura 1 se muestra el procedimiento para descargar varios paquetes incluyendo ddd_layer y SCAPy, Una vez completada la instalación de todas las librerías requeridas se puede hacer una prueba de importación para comprobar que todo esté en orden la figura 1 muestra la correcta importación o invocación de la librería sin ningún error:

```
[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: Descarga de paquetes mediante el comando pip

0.3. Como dibujar un circuito

Una vez que se tiene todo lo necesario para poder trabajar, se recomienda tener instalado LTspice o bien obtenerlo desde la siguiente URL: https://www.analog.com/en/resources/design-tools-and-calculators/ltspice-simulator.html ahora se muestra como obtener las ecuaciones simbólicas del presente circuito que es un amplificador sumador inversor:

Paso 1 dibujar el circuito en LTSpice como se muestra en la figura 2 muestra el inciso (a) como debe dibujarse un circuito contemplando el número de nodos y recordando que todos los nodos conectados a tierra corresponden con el nodo número ”0”

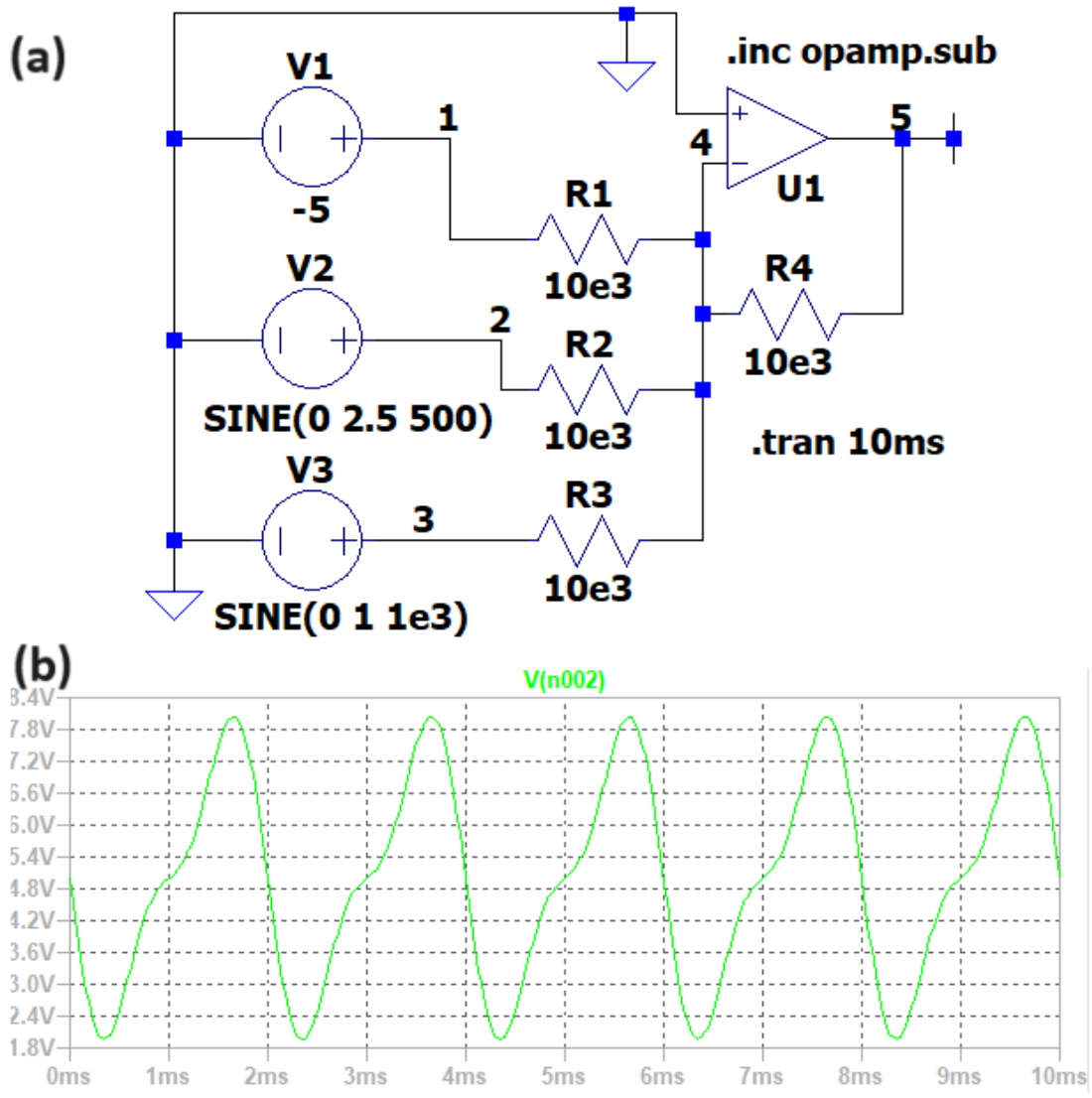

Figura 2: Circuito sumador inversor de tensión, (a) muestra el circuito dibujado en LTspice, (b) muestra el grafico de salida en el nodo numero 5

0.4. Como obtener un netlist de LTspice

Para obtener el netlist a partir de la figura 2 (a) para obtener el netlist de un circuito en LTspice recordando tendrán que estar enumerados todos los nodos a los que se encuentren conectados los elementos pasivos, entonces estando en el esquemático (no en el gráfico) dar clic en view/SPICE Netlist como se muestra en la figura 3

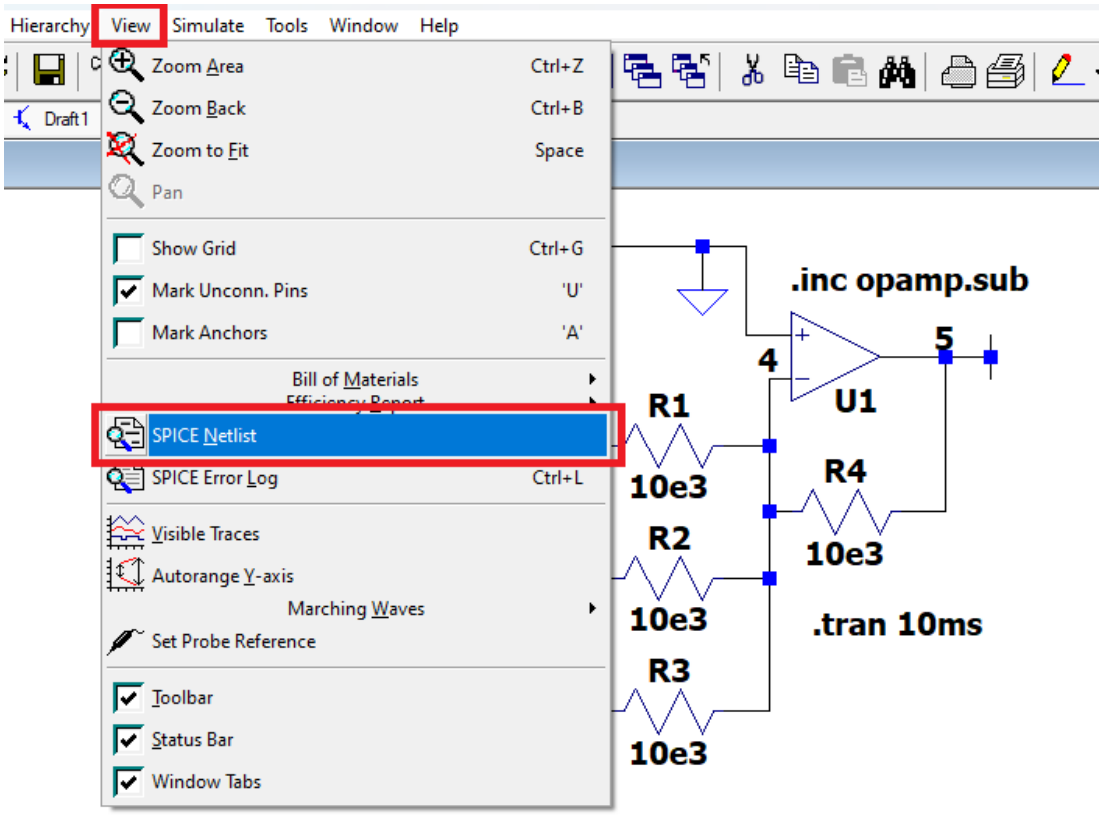

Figura 3: Obtención de netlist

A continuación se muestra la siguiente ventana que devuelve el netlist, sin embargo solo se seleccionan todos los elementos lineales que conforman el circuito omitiendo .inc .tran .backanno .end que son instrucciones adicionales para 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. Cómo preparar un netlist para SCAPy

Este paso consiste en guardar el netlist en formato .cir y asignar el nombre de los elementos con el formato compatible, recordando que si se quiere:

- OpAmp = O
- Resistor = R
- Inductor = L
- Capacitor = C
- Fuente de voltaje = V
- Fuente de corriente = I
- Fuente de voltaje controlada por voltaje (VCVS)= E
- Fuente de corriente controlada por voltaje (VCCS)= G
- Fuente de voltaje controlada por corriente (CCVS)= H
- Fuente de corriente controlada por corriente (CCCS) = F
- Transformador = L1, L2, k1, k2
- Girador = J

De este modo se ordena el netlist de la manera siguiente figura 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 para SCAPy

Se recomienda siempre ordenar los elementos pasivos (R,L,C) y siempre ponerlos en orden ascendente como se muestra en la figura 5 y guardar el archivo con formato .cir

0.6. Cómo computar un circuito en SCAPy

Se recomienda tomar el siguiente script como base para hacer ele cálculo simbólico usando DDD en donde el algoritmo hace la formulación matemática y resuelve el sistema de ecuaciones lineales

```
#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)
```

El nodo de interés es el nodo 5 con respecto al circuito eléctrico de la figura 2 por tanto, que en el terminal se puede solicitar la información del nodo 5 ubicado en el vector X[4] como se muestra en la figura 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: Salida nodo 5 mediante el método DDD de la figura 2

Se recomienda tomar el siguiente script como base para hacer ele cálculo simbólico usando SymPy en donde el algoritmo hace la formulación matemática y resuelve el sistema de ecuaciones lineales

```
#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: Salida nodo 5 mediante el método LU de la figura 2

0.7. Análisis en el dominio del tiempo

En el caso de tener elementos que contengan información en el dominio de la frecuencia directamente desde la formulación MNA como es el caso del capacitor C_n y el inductor L_n el resultado será propiamente en el dominio de la frecuencia con lo se puede evaluar propiamente en dicho dominio; sin embargo, en muchos análisis importa conocer el resultado simbólico, semi-simbólico o numérico en el dominio del tiempo. Para ello se va a analizar un circuito bicuadrado de Tow Thomas en el dominio del tiempo. Para ello se prepara el siguiente circuito:

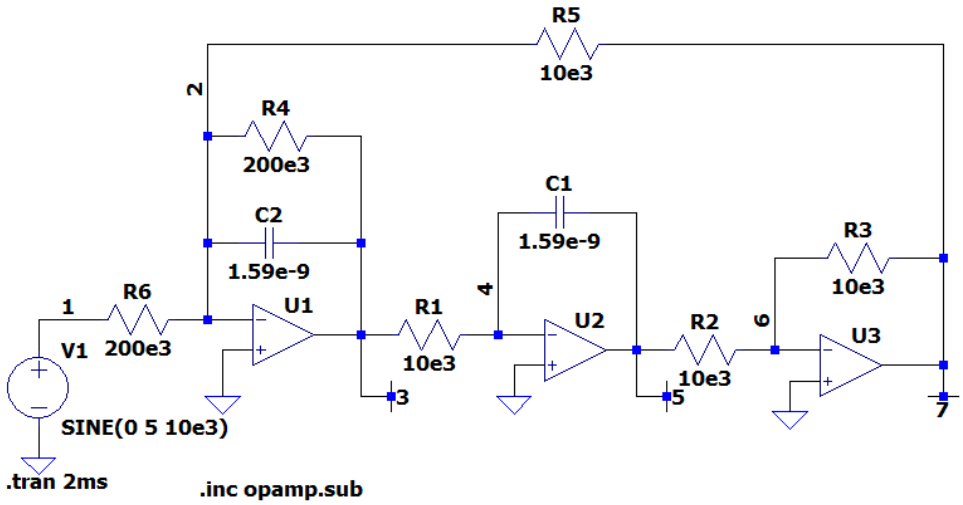

Figura 8: Circuito bicuadrado de Tow-Thomas

Los nodos de interés son los nodos 3, 5, 7 se prepara el netlist de la siguiente forma:

```
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)
```

Se guarda el presente netlist en formato.cir y se prepara el script eligiendo el método de solución por 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)
```

En la terminal se preparan las funciones de transferencia dividiendo la información del vector $H3 = X[2]/V1$, $H5 = X[4]/V1$ y $H7 = X[6]/V1$ como se muestra en la figura:

```
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: Funciones de transferencia para los nodos H3, H5 y H7

A partir de las funciones de transferencia se recomienda usar las transformada de Laplace y la transformada inversa de Laplace de MATLAB, hasta presenta mejor convergencia cuando son ecuaciones algebraicas grandes.

```
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)
```

Cómo puede observarse es importante transformar la fuente de excitación al dominio de la frecuencia para posteriormente obtener el voltaje total en el dominio de la frecuencia y después calcular la transformada inversa de Laplace

Para graficar todo esto en Python se requiere se puede seguir el siguiente procedimiento de acuerdo con el script que se presenta a continuación:

```
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 = (252015497268983545904197580064601145344000000 00*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) -
    (1461374878712162613573151119921385188l*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) -
    (1461374878712162613573151119921385188l*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

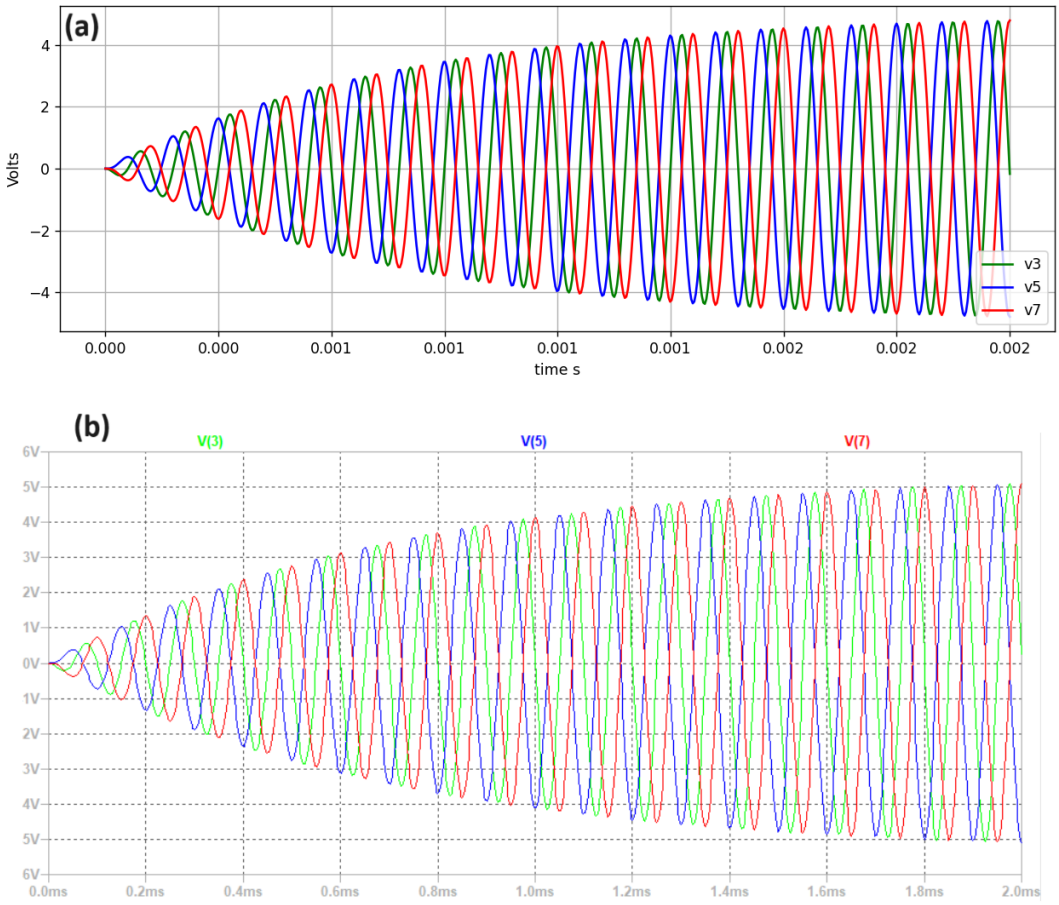

Figura 10: Gráfico comparativo entre el gráfico obtenido con el script de Python (a) y el gráfico obtenido en LTspice (b)