Symbolic Analysis of a Twin-T Active Bandpass Filter with Python

Tiburonboy, juan1543cabrillo@sudomail.com

Last Update or Publication Date: 04 March 2025

Abstract: This report is an analysis of a band pass filter using Symbolic Modified Nodal Analysis implemented with Python in a JupyterLab Notebook then rendered into a PDF. The report builds on the work, *A Bandpass Twin-T Active Filter Used in the Buchla 200 Electric Music Box Synthesizer*, by Aaron D. Lanterman. The report continues with related analysis, thoughts and observations about the filter topology. The use of SymPy makes generating the node equations and obtainig analalytic solutions for the node voltages almost efferortless.

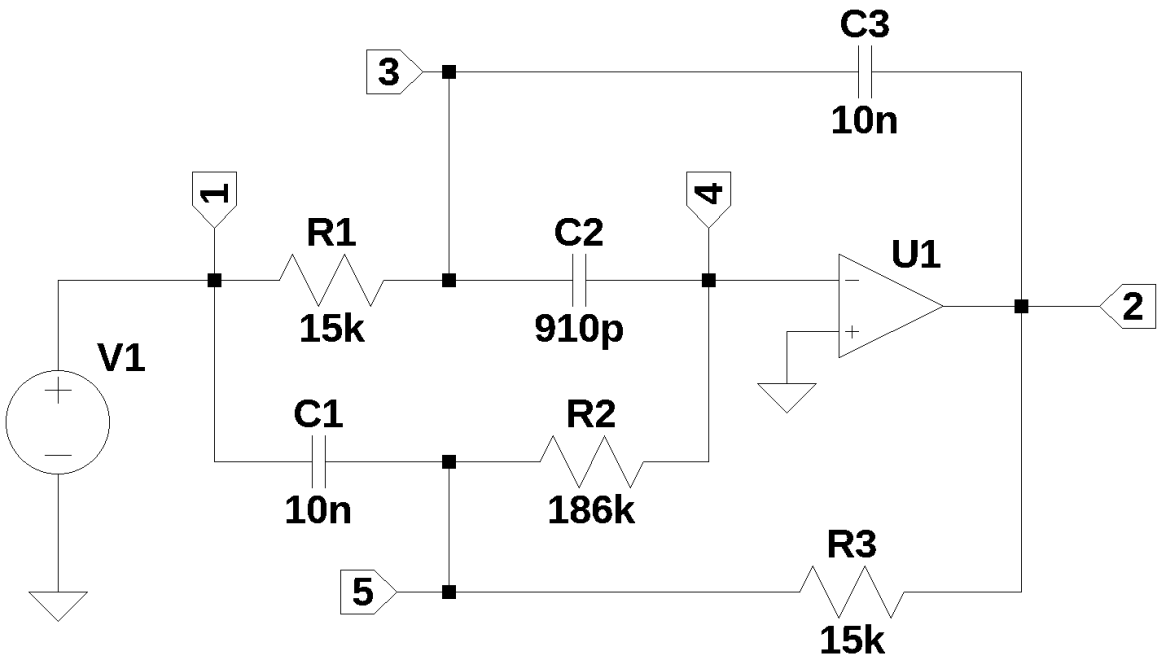

Figure 1: Schematic for Bandpass Twin-T Active Filter with componets for the 1000 Hz band pass filter.

1 INTRODUCTION

This report walks through some of the analysis found in [1], which, describes a third order bandpass filter (BPF) employed in the Buchla Model 295 10 Channel Comb Filter, a synthesizer module developed as part of the Buchla 200 Electric Music Box. The BPF described has a unique arrangement of elements not found in the typical Twin-T configuration, which makes this BPF a interesting candiditate to study. I sometimes watch YouTube videos made by Lantertronics - Aaron Lanterman, the author of [1]. His video, Analysis of the Buchla 295 10 Channel Comb Filter - a Weird Twin-T Topology, describes this filter and some of its characteristics.

A proceddure implemented in the Python programming language called modified nodal analysis (MNA) [2] is used to generate the network equations used in this analysis. The MNA procedure provides an algorithmic method for generating a system of independent equations for linear circuit analysis. Once the scematic is drawn and the net list is generated, the network equations and solutions for the node voltages are eaily obtained as shown in this report. The code in the Jupyter notebook can be used as a template to analyize almost any linear circuit. The schematic for the filter, Figure 1, was drawn using LTSpice and the netlist was exported for the analysis.

This report was written using a JupyterLab notebook and rendered to a PDF document using Quarto which is an open-source scientific and technical publishing system. The source code for this paper is available here and related material is located here.

My motivation for writing this report is: a) my own educational purposes, b) to explore the ability of Python to solve circuit analysis problems, and c) to document the procedure for rendering a JupyterLab notebook into a PDF document.

1.1 Buchla History

Buchla Electronic Musical Instruments (BEMI) was founded by Don Buchla in 1963 under the name Buchla & Associates. The company manufactured synthesizers and MIDI controllers. Buchla's modular synthesizers, like the Buchla 100 and 200 series, were innovative in their design and sound creation approach, differing from Moog synthesizers by employing complex oscillators and waveshaping.

1.2 Related Work

The Python code presented in this report is somewhat unique since Python is open source, free and runs on a variety of platforms. The Python MNA code, [x], is made available under a public domain license and archived in a github repository.

There are other symbolic circuit analysis codes available and some of these are described here. Some of these codes are based on commercial software such as MATLAB, TINA and Maple.

SLiCAP is a symbolic linear analysis tool. SLiCAP is now a Python program, but originally it was written in MATLAB.

TINA is an acronym of Toolkit for Interactive Network Analysis. The TINA design suite is a circuit simulator and PCB design software package for analyzing, designing, and real time testing of analog, digital, HDL, MCU, and mixed electronic circuits and their PCB layouts. TINA has some symbolic analysis capability.

Maple is a mathematical package and there is an application note available describing its use in symbolic circuit analysis. The application note presents a method for evaluating, solving and designing a common, but not so simple pulse-mode high-gain transimpedance amplifier or TIA circuit.

Symbolic Circuit Analysis is a web page devoted to symbolic circuit analysis.

SAPWIN is a windows program package for symbolic and numerical simulation of analog circuits.

Lcapy is an experimental Python package for teaching linear circuit analysis. It uses SymPy for symbolic mathematics. In [3] there is an overview of Lcapy as well as a survey of symbolic circuit analysis packages.

1.3 Circuit Description

The schematic for the 1000 Hz section of the comb filter is shown in Figure 1. The schematic was drawn using LTSpice and the netlist was exported for this analysis. The schematic has component values for the 1000 Hz section of the comb filter. The input node is labeled node 1 and the output of the Op Amp is labeled node 2. The other nodes are sequencially labeled from 3 to 5. The filter circuit has three resisters, three capacitors and one Op Amp. The component values shown in the schematic, Figure 1, where chosen from [1, Table I], which is the 1000 Hz filter. In [4], the Op Amp is listed as N5556V, which is an operational amplifier manufactured by Signetics. The Op Amp used in the analysis is an ideal Op Amp, which is defined in the Python MNA code as a circuit element of type `O` in the circuit netlist.

The Twin-T filter topology is made up of two "T" shaped networks connected together, which is why it's called "Twin-T". The arrangment of the resistors and capacitors in Figure 1 is a bit different than the more common notch (band stop) Twin-T filter which shows up in on-line when you search for "Twin-T".

The filter has three capacitors which sugests that the circuit has a thrird order characteristic polynominal.

1.4 Analysis

The circuit analysis is performed using a JupyterLab notebook running Python code. This report is directly rendered from a JupyterLab notebook and the code cells with results are displayed. The code cells are highlighted in gray and most of the code output are displayed by converting the equations to LaTex and then markdown to make the fonts used by the PDF rendering engine consistent. Some of the code cells have supressed in the PDF output by placing mark down command, `#| echo: false`, in the first line of the code cell. This was done to improve readability since some of the code repeats. The `ipynb` notebook file in the GitHub repository can be examined to view all the code cells, even those hidden by using HTML comment tags.

LTSpice was used to draw the schematic of the filter and generate the netlist. The component values were manually edited to used scientif notation and the Op Amp entry was fixed. The analysis begins with generating the newtork equations from the circuits netlist by calling:

```
report, network_df, df2, A, X, Z = SymMNA.smna(example_net_list)
```

SymPy is capabled of working through the algebra to solved the system of equations and even find the roots of the numerator and denominator polimominals.

The analysis continues with a reduced complexity version of the filter, as described in [1]. A sensitivity analysis for the filter is performed along with a Monte Carlo and worst case tolerance analysis.

New sections

- Combinations of substitutions
- Investigate range of allowable initial guesses
- design plots

New section. Results, Discussion and Conclusion

1.4.1 Reproducibility

The analysis is based on Python modules constsing of SymPy, NumPy, SciPy and Pandas. Code to automatically generate network equations from circuit netlist using Modified-Nodal-Analysis [2]. The function SymMNA is describe in the hyperlink. The results presented in this report should be reproducable if the package versions listed in Table 1 are used.

Table 1: Package or Library versions

Package	version
Python	3.10.9
JupyterLab	3.5.3

Package	version
IPython	8.10.0
NumPy	1.23.5
SymPy	1.11.1
SciPy	1.10.0
Pandas	1.5.3
Matplotlib	3.7.0
Tabulate	0.8.10
LTSpice	17.1.8
Quarto	1.4.553

The JupyterLab notebook used to generate this report was run on a laptop with an Intel i3-8130U CPU @ 2.2 GHz, illustrating that a high performace machine is not required to perform the analysis described in this document.

2 SYMBOLIC ANALYSIS

Symbolic analysis presented in this report employes the MNA technique to generate network equations from the filter's netlist with the circuit's elements are represented by symbols. SymPy is used to solve the system of equations in symbolic form to obtain analytic expressions for the filter's voltage transfer function. The symbolic expressions can in some cases provide a deeper understanding of how each component contributes to the circuit's operation that can complement numerical simulations.

2.1 Circuit Netlist

The netlist generated by LTSpice from the schematic is shown below. Some edits were made to fix up the formating of the component values and the Op Amp declaration. The nodes were labeled in the schematic, otherwise LTSpice will use default labels such as N001, N002 etc. and the smna function wants integer values for the node numbers and these need to be consecutively ordered with no gaps in the numbering.

```
* Bandpass-Twin-T-Active-Filter.asc
R1 3 1 15k
R2 4 5 186k
R3 2 5 15k
C1 5 1 10n
C2 4 3 910p
C3 2 3 10n
```

```
XU1 4 0 2 opamp Aol=100K GBW=10Meg
V1 1 0 1 AC 1
.lib opamp.sub
.ac dec 1000 100 10k
.backanno
.end
```

Generation of the netlist from a schematic caputure program is comvinent and less error propne than generating the netwlist by hand. A visiual inspection of the schematic insurs that the circuit to be analyzed is correct and it follows that the netlist is alos correct. This is espically true or larger more complicated schamitics.

The cleaned up netlist was copied to the cell below. The tripple quotes allow text strings to span multiple lines since the line breaks are preserved.

```
example_net_list = '''
* Bandpass-Twin-T-Active-Filter.asc
R1 3 1 15e3
R2 4 5 186e3
R3 2 5 15e3
C1 5 1 10e-9
C2 4 3 910e-12
C3 2 3 10e-9
O1 4 0 2
V1 1 0 1
'''
```

2.2 Generate Network Equations

The function `SymMNA.smna(example_net_list)` implements the MNA method on the filter's netlist. Stamps which are templates for modifying the B, C and D matrices are used to facilitate the construction of the matrices. The stamps used in this implementation of the MNA follow the stamps of [4]. The function is divided in the following sections.

- The preprocessor reads in the netlist text file and removes comments, extra spaces and blank lines. The first letter of the element type is capitalized to make subsequent parsing of the file easier. The number of lines are counted and the number of entries on each line are checked to make sure the count is consistent with the element type.
- The parser code loads the preprocessed netlist into a data frame. A report is generated which consists of a count of the element types in the netlist.
- Matrix formulation: Each of the matrices and vectors are generated.

- Circuit equation generation: The circuit equations are generated in a `for` loop. Sympy automatically does some simplification according to its default settings. The Laplace variable s is used when inductors and capacitors are included in the circuit.

There is a limited amount of error checking performed. The number of items on each line in the netlist is checked to make sure the count is correct depending on the element type. The node numbering needs to be consecutive with no skipped numbers, otherwise empty rows or columns will be included in the matrix. Unknown element types create an error.

The function takes one argument which is a text string that is the circuit's netlist. The format of the netlist is very similar to the standard spice netlist and any of the required changes can be easily made with a text editor. The function returns six items.

1. report - a text string, which is the netlist report.
2. df - a Pandas data frame, which the circuit's net list info loaded into a data frame.
3. df2 - a Pandas data frame, which contains the branches with unknown currents.
4. A - a SymPy matrix, which is $(m+n)$ by $(m+n)$, where n is the number of nodes and m is the number of current unknowns. A is the combination of four smaller matrices, G, B, C, and D. These matrices are described in the GitHub repository.
5. X - a list that holds the unknown node voltages and the currents through the independent voltage sources.
6. Z - a list that holds the independent voltage and current sources

The netlist can be generated by hand or exported from a schematic capture program and pasted into the JupyterLab notebook.

```
report, network_df, df2, A, X, Z = SymMNA.smna(example_net_list)
```

Convert the lists of unknown node voltages and currents as well as the list of independednt voltage and current sources into SymPy matrices.

```
X = Matrix(X)
Z = Matrix(Z)
```

Use the SymPy function `Eq` to formulate the network equations.

```
NE_sym = Eq(A*X,Z)
```

The equations, `NE_sym`, define the mathematical relationship between voltages and currents in the filter circuit in terms of node voltages, v_1, v_2 etc., components R_1, C_1 etc., the Laplace variable s and the indepenedent voltage source V_1. The equations are shown below:

```
# Put matrices into SymPy
X = Matrix(X)
Z = Matrix(Z)

NE_sym = Eq(A*X,Z)

# display the equations
temp = ''
for i in range(shape(NE_sym.lhs)[0]):
    temp += '<p>${:s} = {:s}$</p>'.format(latex(NE_sym.lhs[i]),
        latex(NE_sym.rhs[i]))

Markdown(temp)
```

$-C_1 s v_5 + I_{V1} + v_1 \left(C_1 s + \frac{1}{R_1}\right) - \frac{v_3}{R_1} = 0$

$-C_3 s v_3 + I_{O1} + v_2 \left(C_3 s + \frac{1}{R_3}\right) - \frac{v_5}{R_3} = 0$

$-C_2 s v_4 - C_3 s v_2 + v_3 \left(C_2 s + C_3 s + \frac{1}{R_1}\right) - \frac{v_1}{R_1} = 0$

$-C_2 s v_3 + v_4 \left(C_2 s + \frac{1}{R_2}\right) - \frac{v_5}{R_2} = 0$

$-C_1 s v_1 + v_5 \left(C_1 s + \frac{1}{R_3} + \frac{1}{R_2}\right) - \frac{v_2}{R_3} - \frac{v_4}{R_2} = 0$

$v_1 = V_1$

$v_4 = 0$

The two lines of Python code below turn the free symbols from the `NE_sym` equations into SymPy variables and the element values contained in the netlist are put into a Python dictionary data structure to be used in the numerical solutions later in this report.

```
var(str(NE_sym.free_symbols).replace('{','').replace('}',''))
element_values = SymMNA.get_part_values(network_df)
```

2.3 Solve the Network Equations

The SymPy function `solve` is used to obtain the node voltages and independent voltage source currents.

```
U_sym = solve(NE_sym,X)
```

The solution to the network equations are the node voltages expressed in terms of the Laplace variable s are displayed below.

```
temp = ''
for i in U_sym.keys():
    if str(i)[0] == 'v': # only display the node voltages
        temp += '<p>${:s} = {:s}$</p>'.format(latex(i),latex(U_sym[i]))

Markdown(temp)
```

$v_1 = V_1$

$v_2 = \frac{-C_1C_2R_1R_3V_1s^2-C_1C_2R_2R_3V_1s^2-C_1C_3R_1R_3V_1s^2-C_1R_3V_1s-C_2R_2V_1s-C_2R_3V_1s}{C_1C_2C_3R_1R_2R_3s^3+C_2C_3R_1R_2s^2+C_2C_3R_1R_3s^2+C_2R_1s+C_3R_1s+1}$

$v_3 = \frac{-C_1C_3R_1R_3V_1s^2+V_1}{C_1C_2C_3R_1R_2R_3s^3+C_2C_3R_1R_2s^2+C_2C_3R_1R_3s^2+C_2R_1s+C_3R_1s+1}$

$v_4 = 0$

$v_5 = \frac{C_1C_2C_3R_1R_2R_3V_1s^3-C_2R_2V_1s}{C_1C_2C_3R_1R_2R_3s^3+C_2C_3R_1R_2s^2+C_2C_3R_1R_3s^2+C_2R_1s+C_3R_1s+1}$

2.3.1 Voltage Transfer Function $H(s) = \frac{v_2(s)}{v_1(s)}$

The voltage transfer function, $H(s)$, for the filter is generated and displayed by the following code:

```
H_sym = cancel(U_sym[v2]/U_sym[v1],s)
Markdown('$H(s)={:s}$'.format(latex(H_sym)))
```

$H(s) = \frac{s^2(-C_1C_2R_1R_3-C_1C_2R_2R_3-C_1C_3R_1R_3)+s(-C_1R_3-C_2R_2-C_2R_3)}{C_1C_2C_3R_1R_2R_3s^3+s^2(C_2C_3R_1R_2+C_2C_3R_1R_3)+s(C_2R_1+C_3R_1)+1}$

This expression agrees with [1 eq. (1)]. The numerator is a second order polynominal and the demominator is third order polynominal in terms of s. Generally, the order of the dominator is equal to the number of reactive elements in the circuit. The roots of the numerator polynominal are called the zeros of the transfer function and the roots of the denominator are called the poles of the transfer function.

```
H_sym_num, H_sym_denom = fraction(H_sym,s) #returns numerator and denominator
```

2.3.2 Numerator Polynominal

The numerator polynominal is:

```
Markdown('$N(s)={:s}$'.format(latex(H_sym_num)))
```

$N(s) = s^2\left(-C_1C_2R_1R_3 - C_1C_2R_2R_3 - C_1C_3R_1R_3\right) + s\left(-C_1R_3 - C_2R_2 - C_2R_3\right)$

The coeficients of each Laplace terms can be equated to the variables b_2, b_1 and b_0 in the expression:

$b_2s^2 + b_1s + b_0$

where b_2, b_1 and b_0 are:

```
b2 = H_sym_num.coeff(s**2)
b1 = H_sym_num.coeff(s**1)
b0 = H_sym_num - b1*s*1 - b2*s**2

Markdown('<p>$b_2={:s}$</p><p>$b_1={:s}$\
    </p><p>$b_0={:s}$</p>'.format(latex(b2),latex(b1),latex(b0)))
```

$b_2 = -C_1C_2R_1R_3 - C_1C_2R_2R_3 - C_1C_3R_1R_3$

$b_1 = -C_1R_3 - C_2R_2 - C_2R_3$

$b_0 = 0$

Notice that ther terms b_2 and b_1 are negative. This follows from the Op Amp being configured as an inverting amplifier. R2 and R3 provide a DC path from the Op Amp's output to the inverting terminal.

The roots of the numerator polynominal can easily be found with SymPy.

This filter has two transmission zeros

```
num_root_sym = solve(H_sym_num,s)
```

There are two solutions, $z_1 = 0$ and another root at:

```
Markdown('$z_2={:s}$'.format(latex(num_root_sym[1])))
```

$z_2 = \frac{-C_1R_3-C_2R_2-C_2R_3}{C_1R_3(C_2R_1+C_2R_2+C_3R_1)}$

2.3.3 Denominator Polynominal

The demoninator polynominal of the transfer function is called the characteristic polynominal. The roots of the denominator, also called poles of the system, determine the system's stability. If any of these roots have a positive real part, the system is unstable, meaning its output will grow unbounded. The roots also influence how the system responds to changes in input (the transient response). They affect things like how quickly the system settles to a new state, whether it oscillates, and the damping of those oscillations. Each root of the characteristic polynominal corresponds to a natural mode of the system.

The denominator polynominal is:

```
Markdown('$D(s)={:s}$'.format(latex(H_sym_denom)))
```

$$D(s) = C_1C_2C_3R_1R_2R_3s^3 + s^2\left(C_2C_3R_1R_2 + C_2C_3R_1R_3\right) + s\left(C_2R_1 + C_3R_1\right) + 1$$

The coeficients of each Laplace terms can be equated to the variables a_3, a_2, a_1 and a_0 in the expression:

$a_3s^3 + a_2s^2 + a_1s + a_0$

where a_3, a_2, a_1 and a_0 are:

$a_3 = C_1C_2C_3R_1R_2R_3$

$a_2 = C_2C_3R_1R_2 + C_2C_3R_1R_3$

$a_1 = C_2R_1 + C_3R_1$

$a_0 = 1$

The roots of the denominator polynominal can found with SymPy. The filter circuit being analyzed in this report has a denominator polynominal that SymPy can quickly solve. This is not always the case. The expressions are long and do not render well when the JupyterLab notebook is converted to PDF, so the expressions were converted to LaTex strings and then to PNG images with latex2png for this report.

```
denom_root_sym = solve(H_sym_denom,s)
```

$$-\frac{\sqrt[3]{\frac{\sqrt{-4\left(-\frac{3(C_2+C_3)}{C_1C_2C_3R_2R_3}+\frac{(R_2+R_3)^2}{C_1^2R_2^2R_3^2}\right)^3+\left(\frac{27}{C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{C_1^2C_2C_3R_2^2R_3^2}+\frac{2(R_2+R_3)^3}{C_1^3R_2^3R_3^3}\right)^2}}{2}+\frac{27}{2C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{2C_1^2C_2C_3R_2^2R_3^2}+\frac{(R_2+R_3)^3}{C_1^3R_2^3R_3^3}}}{3}$$
$$-\frac{-\frac{3(C_2+C_3)}{C_1C_2C_3R_2R_3}+\frac{(R_2+R_3)^2}{C_1^2R_2^2R_3^2}}{3\sqrt[3]{\frac{\sqrt{-4\left(-\frac{3(C_2+C_3)}{C_1C_2C_3R_2R_3}+\frac{(R_2+R_3)^2}{C_1^2R_2^2R_3^2}\right)^3+\left(\frac{27}{C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{C_1^2C_2C_3R_2^2R_3^2}+\frac{2(R_2+R_3)^3}{C_1^3R_2^3R_3^3}\right)^2}}{2}+\frac{27}{2C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{2C_1^2C_2C_3R_2^2R_3^2}+\frac{(R_2+R_3)^3}{C_1^3R_2^3R_3^3}}}$$
$$-\frac{R_2+R_3}{3C_1R_2R_3}$$

Figure 2: Terms for the first pole.

$$-\frac{\left(-\frac{1}{2}-\frac{\sqrt{3}i}{2}\right)\sqrt[3]{\frac{\sqrt{-4\left(-\frac{3(C_2+C_3)}{C_1C_2C_3R_2R_3}+\frac{(R_2+R_3)^2}{C_1^2R_2^2R_3^2}\right)^3+\left(\frac{27}{C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{C_1^2C_2C_3R_2^2R_3^2}+\frac{2(R_2+R_3)^3}{C_1^3R_2^3R_3^3}\right)^2}}{2}+\frac{27}{2C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{2C_1^2C_2C_3R_2^2R_3^2}+\frac{(R_2+R_3)^3}{C_1^3R_2^3R_3^3}}}{3}$$
$$-\frac{-\frac{3(C_2+C_3)}{C_1C_2C_3R_2R_3}+\frac{(R_2+R_3)^2}{C_1^2R_2^2R_3^2}}{3\left(-\frac{1}{2}-\frac{\sqrt{3}i}{2}\right)\sqrt[3]{\frac{\sqrt{-4\left(-\frac{3(C_2+C_3)}{C_1C_2C_3R_2R_3}+\frac{(R_2+R_3)^2}{C_1^2R_2^2R_3^2}\right)^3+\left(\frac{27}{C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{C_1^2C_2C_3R_2^2R_3^2}+\frac{2(R_2+R_3)^3}{C_1^3R_2^3R_3^3}\right)^2}}{2}+\frac{27}{2C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{2C_1^2C_2C_3R_2^2R_3^2}+\frac{(R_2+R_3)^3}{C_1^3R_2^3R_3^3}}}$$
$$-\frac{R_2+R_3}{3C_1R_2R_3}$$

Figure 3: Terms for the second pole.

$$-\frac{\left(-\frac{1}{2}+\frac{\sqrt{3}i}{2}\right)\sqrt[3]{\frac{\sqrt{-4\left(-\frac{3(C_2+C_3)}{C_1C_2C_3R_2R_3}+\frac{(R_2+R_3)^2}{C_1^2R_2^2R_3^2}\right)^3+\left(\frac{27}{C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{C_1^2C_2C_3R_2^2R_3^2}+\frac{2(R_2+R_3)^3}{C_1^3R_2^3R_3^3}\right)^2}}{2}+\frac{27}{2C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{2C_1^2C_2C_3R_2^2R_3^2}+\frac{(R_2+R_3)^3}{C_1^3R_2^3R_3^3}}}{3}$$
$$-\frac{-\frac{3(C_2+C_3)}{C_1C_2C_3R_2R_3}+\frac{(R_2+R_3)^2}{C_1^2R_2^2R_3^2}}{3\left(-\frac{1}{2}+\frac{\sqrt{3}i}{2}\right)\sqrt[3]{\frac{\sqrt{-4\left(-\frac{3(C_2+C_3)}{C_1C_2C_3R_2R_3}+\frac{(R_2+R_3)^2}{C_1^2R_2^2R_3^2}\right)^3+\left(\frac{27}{C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{C_1^2C_2C_3R_2^2R_3^2}+\frac{2(R_2+R_3)^3}{C_1^3R_2^3R_3^3}\right)^2}}{2}+\frac{27}{2C_1C_2C_3R_1R_2R_3}-\frac{9(C_2+C_3)(R_2+R_3)}{2C_1^2C_2C_3R_2^2R_3^2}+\frac{(R_2+R_3)^3}{C_1^3R_2^3R_3^3}}}$$
$$-\frac{R_2+R_3}{3C_1R_2R_3}$$

Figure 4: Terms for the third pole.

As is evident in the analytic expressions for the poles of the transfer fruction shown in the figures above, these expressions are not very intuitive or useful inless some simplication is performed. With the help of the Python code and SymPy, the procedure outlined above can be considerd a 'brute force' analysis, where analyltic expressions can be easily obtained from a circuit's netlist with little effort. If the circuit is small, that is having only a few componets or nodes, the symbolic expressions can be very useful.

2.4 FACTS (Fast Analytical Circuits Techniques)

The FACTS (Fast Analytical Circuits Techniques) method refers to a circuit analysis technique that enables engineers to obtain the transfer functions and input and output impedances of a circuit by inspection without resorting to too much algebra. The circuits offered as illustrations of the technique are small with one or two reactive elements. Middlebrook [x] seems to have been the first proponent of the method and describes what he calls design oriented analysis. He talks about low entropy equations, in which the terms and elements are ordered or grouped in such a way that their physical origin, where they come from in the circuit, and what part of the circuit contributes to this part of the final expression becomes obvious. The method is also described in [x].

The origin of the FACTS circuit analysis method, as stated above, is from [x] and [x] @Middlebrook1991. IEEE technical papers and books by [x] @basso2016 and [x] @Vorpérian2002 have been written on the subject, however these are not free or available to download for free. FACTS is a circuit analysis method where the transfer function is determined by applying the Extra Element Theorem. Christophe Basso, Vatché Vorpérian and others have open source content available:

- Introduction to Fast Analytical Techniques: Application to Small-Signal Modeling
- A series of 22 YouTube videos starting with: Fast Analytical Techniques for Electrical and Electronic Circuits
- Extra Element Theorem: An Introduction (with Examples)

According to the open source FACTS literature:

> The well-known and widely used methods of nodal or loop analysis, while effective for obtaining numerical solutions, are largely ineffective for deriving analytical solutions in symbolic form, except for simple circuits. Attempting to invert a matrix with symbolic entries, even for low-order matrices, results in tedious algebra and complex, high-entropy expressions that provide little meaningful insight.

The FACTS procedure, as illustrated in the open source literature, employs example circuits that are usually of order 1 or 2. The procedure results in equations that keep circuit elements grouped together and thus the equations are more intuitive. For 3rd order and higher circuits, the procedure becomes rather involved and I didn't see any circuit analysis examples beyond a 2nd order circuit.

Traditional circuit analysis techniques usually don't produce analytical answers, i.e. equations in terms of symbols. Both the FACTS procedure and the MNA Python code in this book yield analytical answers, with the FACTS results possibly being more intuitive and the MNA Python code providing an automated solution, which would then have to be algebraically manipulated to be put into a FACTS type solution.

IN the video series start with video 8

Add link to jupyterlab notebook in github.

https://github.com/Tiburonboy/EE_jupyter_notebooks/blob/main/Buchla%20Twin-T%20Active%20Filter/Twin-T_BPF_FACTS.ipynb

Prototype FACTS equations in seperate notebook.

Twin-T_BPF_FACTS.ipynb

references


```
@INPROCEEDINGS{Middlebrook1992,
  author={Middlebrook, R.D.},
  booktitle={Proceedings. Twenty-Second Annual conference Frontiers in Education},
  title={Methods of Design-Oriented Analysis: The Quadratic Equation Revisited},
  year={1992},
  pages={95-102},
  doi={10.1109/FIE.1992.683365}
}

@INPROCEEDINGS{Middlebrook1991,
  author={Middlebrook, R.D.},
  booktitle={Proceedings Frontiers in Education Twenty-First Annual Conference. Engineering I
  title={Low-entropy expressions: the key to design-oriented analysis},
  year={1991},
  pages={399-403},
  doi={10.1109/FIE.1991.187513}
}

@online{basso2016,
    author = {Basso, Christophe},
    title = {Introduction to Fast Analytical Techniques: Application to Small-Signal Modeling
    url  = {https://www.powersimtof.com/Downloads/PPTs/Chris%20Basso%20APEC%20seminar%202016
    year={2016},
    addendum = {accessed 12 Feb 2024}
}

@book{Vorpérian2002,
  title={Fast Analytical Techniques for Electrical and Electronic Circuits},
  author={Vatché Vorpérian},
  isbn={9780521624718},
  year={2002},
  url={https://www.cambridge.org/core/books/fast-analytical-techniques-for-electrical-and-ele
  publisher={Cambridge University Press},
```

```
  addendum = {accessed 1 July 2024}
}
```

3 REDUCED COMPLEXITY

As described in [1], the transfer function for the filter simplifies under the case of $R_1 = R_3$ and $C_1 = C_3$ to a second order polynominal characteristic equation. This simplification works because a zero cancels with a pole in the voltage transfer function. The numerator polynominal reduces to the following when the Laplace term, s, is colleded and factored.

$N_r(s) = -s\left(CRs + 1\right)\left(CR + C_2R + C_2R_2\right)$

In a simular fashon, the denominator polynominal is:

$D_r(s) = \left(CRs + 1\right)\left(CC_2RR_2s^2 + C_2Rs + 1\right)$

You can see the $(CRs + 1)$ terms will cancel in the numerator and denominator.

$\frac{N_r(s)}{D_r(s)} = \frac{-s(CRs+1)(CR+C_2R+C_2R_2)}{(CRs+1)(CC_2RR_2s^2+C_2Rs+1)} = \frac{-s(CR+C_2R+C_2R_2)}{(CC_2RR_2s^2+C_2Rs+1)}$

3.1 Symbolic Analysis for $R1 = R3 = R$ and $C1 = C3 = C$

The following Python code makes the substitution, $R_1 = R_3 = R$ and $C_1 = C_3 = C$, into the network equations.

```
NE_sym_reduced_complexity = NE_sym.subs({R1:R,R3:R,C1:C,C3:C})
```

This produces the following network equations:

$0 = -Csv_5 + I_{V1} + v_1\left(Cs + \frac{1}{R}\right) - \frac{v_3}{R}$

$0 = -Csv_3 + I_{O1} + v_2\left(Cs + \frac{1}{R}\right) - \frac{v_5}{R}$

$0 = -Csv_2 - C_2sv_4 + v_3\left(Cs + C_2s + \frac{1}{R}\right) - \frac{v_1}{R}$

$0 = -C_2sv_3 + v_4\left(C_2s + \frac{1}{R_2}\right) - \frac{v_5}{R_2}$

$0 = -Csv_1 + v_5\left(Cs + \frac{1}{R_2} + \frac{1}{R}\right) - \frac{v_4}{R_2} - \frac{v_2}{R}$

$V_1 = v_1$

$0 = v_4$

The unknown node voltages and current from V_1 can be found by using the SymPy `solve` function.

```
U_sym_reduced_complexity = solve(NE_sym_reduced_complexity,X)
```

The produces the following expresions for the node voltages.

$v_1 = V_1$

$v_2 = \frac{-CRV_1 s - C_2 RV_1 s - C_2 R_2 V_1 s}{CC_2 RR_2 s^2 + C_2 Rs + 1}$

$v_3 = \frac{-CRV_1 s + V_1}{CC_2 RR_2 s^2 + C_2 Rs + 1}$

$v_4 = 0$

$v_5 = \frac{CC_2 RR_2 V_1 s^2 - C_2 R_2 V_1 s}{CC_2 RR_2 s^2 + C_2 Rs + 1}$

3.2 Transfer Function $H(s) = \frac{v_2(s)}{v_1(s)}$

The voltage transfer function for the reduced complexity network is:

$H(s) = \frac{s(-CR - C_2 R - C_2 R_2)}{CC_2 RR_2 s^2 + C_2 Rs + 1}$

The equation above is different than [1, eq (8)], which is missing the s term in the numerator.

3.2.1 Characteristic Polynominal

The characteristic polynominal for $R_1 = R_3 = R$ and $C_1 = C_3 = C$, is shown below.

$CC_2 RR_2 s^2 + C_2 Rs + 1$

Get the coefficients of the Laplace variable, s, from the quadratic equation:

$as^2 + bs + c = 0$

where:

$a = CC_2 RR_2$

$b = C_2 R$

$c = 1$

The poles of the transfer function are:

$\left[\frac{-C_2 R - \sqrt{C_2 R(-4CR_2 + C_2 R)}}{2CC_2 RR_2}, \frac{-C_2 R + \sqrt{C_2 R(-4CR_2 + C_2 R)}}{2CC_2 RR_2}\right]$

The pole are complex if, $C_2 R < 4CR_2$.

3.2.2 Natural Frequency, Q Factor and Damping Ratio

As described in Q factor, a two pole band pass filter has the following transfer function:

$H(s) = \frac{\frac{\omega_n}{Q} s}{s^2 + \frac{\omega_n}{Q} s + {\omega_n}^2}$

The demominator of $H(s)$, also called the characteristic polynominal, and is:

$s^2 + \frac{\omega_n}{Q} s + \omega_n^2$

Rearanging the coefficients of the Lapace variable, we get:

$\frac{s^2}{\omega_n^2} + \frac{1}{Q\omega_n} s + 1$

By equating the coefficients, ω_n, Q Factor and Damping Ratio can be expressed in terms of R_2, R, C_2 and C.

$\omega_n = \sqrt{\frac{1}{CC_2RR_2}}$

$Q\ factor = \frac{1}{C_2 R \sqrt{\frac{1}{CC_2RR_2}}}$

In [1, eq. (12)] the following expression for Q is given:

$\sqrt{\frac{R_2 C_1}{R_1 C_2}}$

which is different than my results.

From Q factor:

$\frac{\omega_n}{Q} = 2\zeta\omega_n = 2\alpha$

Solving for the damping ratio, ζ:

$\zeta = \frac{C_2 R \sqrt{\frac{1}{CC_2RR_2}}}{2}$

The attenuation parameter, α, represents the rate of decay of the oscillations of the system.

$\alpha = \frac{1}{2CR_2}$

3.3 Numerical Analysis

Afer substituting the element values from the netlist into the equations with the following code:

```
equ_N = NE_sym.subs(element_values)
```

we get the following numerical equations:

$$0 = I_{V1} - 1.0 \cdot 10^{-8} s v_5 + v_1 \cdot (1.0 \cdot 10^{-8} s + 6.6667 \cdot 10^{-5}) - 6.6667 \cdot 10^{-5} v_3$$

$$0 = I_{O1} - 1.0 \cdot 10^{-8} s v_3 + v_2 \cdot (1.0 \cdot 10^{-8} s + 6.6667 \cdot 10^{-5}) - 6.6667 \cdot 10^{-5} v_5$$

$$0 = -1.0 \cdot 10^{-8} s v_2 - 9.0 \cdot 10^{-10} s v_4 - 6.6667 \cdot 10^{-5} v_1 + v_3 \cdot (1.1 \cdot 10^{-8} s + 6.6667 \cdot 10^{-5})$$

$$0 = -9.0 \cdot 10^{-10} s v_3 + v_4 \cdot (1.0 \cdot 10^{-9} s + 5.376 \cdot 10^{-6}) - 5.376 \cdot 10^{-6} v_5$$

$$0 = -1.0 \cdot 10^{-8} s v_1 - 6.6667 \cdot 10^{-5} v_2 - 5.376 \cdot 10^{-6} v_4 + v_5 \cdot (1.0 \cdot 10^{-8} s + 7.2043 \cdot 10^{-5})$$

$$1.0 = v_1$$

$$0 = v_4$$

Solve for unknown node voltages, I_{V1} and I_{O1} in terms of Laplace variable s. The node voltages are:

$$v_1 = 1.0$$

$$v_2 = \frac{-1.53414746543779 \cdot 10^{18} s^2 - 1.02276497695853 \cdot 10^{22} s}{117000000000000.0 s^3 + 8.42903225806452 \cdot 10^{17} s^2 + 5.02764976958526 \cdot 10^{21} s + 3.07219662058372 \cdot 10^{25}}$$

$$v_3 = \frac{1.53609831029186 \cdot 10^{24} - 3.45622119815668 \cdot 10^{16} s^2}{5850000000000.0 s^3 + 4.21451612903226 \cdot 10^{16} s^2 + 2.51382488479263 \cdot 10^{20} s + 1.53609831029186 \cdot 10^{24}}$$

$$v_4 = 0.0$$

$$v_5 = \frac{5850000000000.0 s^3 - 2.6 \cdot 10^{20} s}{5850000000000.0 s^3 + 4.21451612903226 \cdot 10^{16} s^2 + 2.51382488479263 \cdot 10^{20} s + 1.53609831029186 \cdot 10^{24}}$$

The network transfer function, $H(s) = \frac{v_2(s)}{v_1(s)}$, is:

$$H(s) = \frac{-1.53414746543779 \cdot 10^{18} s^2 - 1.02276497695853 \cdot 10^{22} s}{117000000000000.0 s^3 + 8.42903225806452 \cdot 10^{17} s^2 + 5.02764976958526 \cdot 10^{21} s + 3.07219662058372 \cdot 10^{25}}$$

The SciPy function TransferFunction is used to represent the system as the continuous-time transfer function.

```
H_num, H_denom = fraction(H) #returns numerator and denominator

# convert symbolic to numpy polynomial
a = np.array(Poly(H_num, s).all_coeffs(), dtype=float)
b = np.array(Poly(H_denom, s).all_coeffs(), dtype=float)

sys_tf = signal.TransferFunction(a,b)
```

3.3.1 Pole zero plot

The poles and zeros of the transfer function can easly be obtained with the following code:

```
sys_zeros = np.roots(sys_tf.num)
sys_poles = np.roots(sys_tf.den)
```

The poles and zeros of the transfer function are plotted on the complex frequency plane.

```
plt.plot(np.real(sys_zeros/(2*np.pi)), np.imag(sys_zeros/(2*np.pi)), 'ob',
    markerfacecolor='none')
plt.plot(np.real(sys_poles/(2*np.pi)), np.imag(sys_poles/(2*np.pi)), 'xr')
plt.legend(['Zeros', 'Poles'], loc=0)
plt.title('Pole / Zero Plot')
plt.xlabel('real part, \u03B1')
plt.ylabel('imaginary part, j\u03C9')
plt.grid()
plt.show()
```

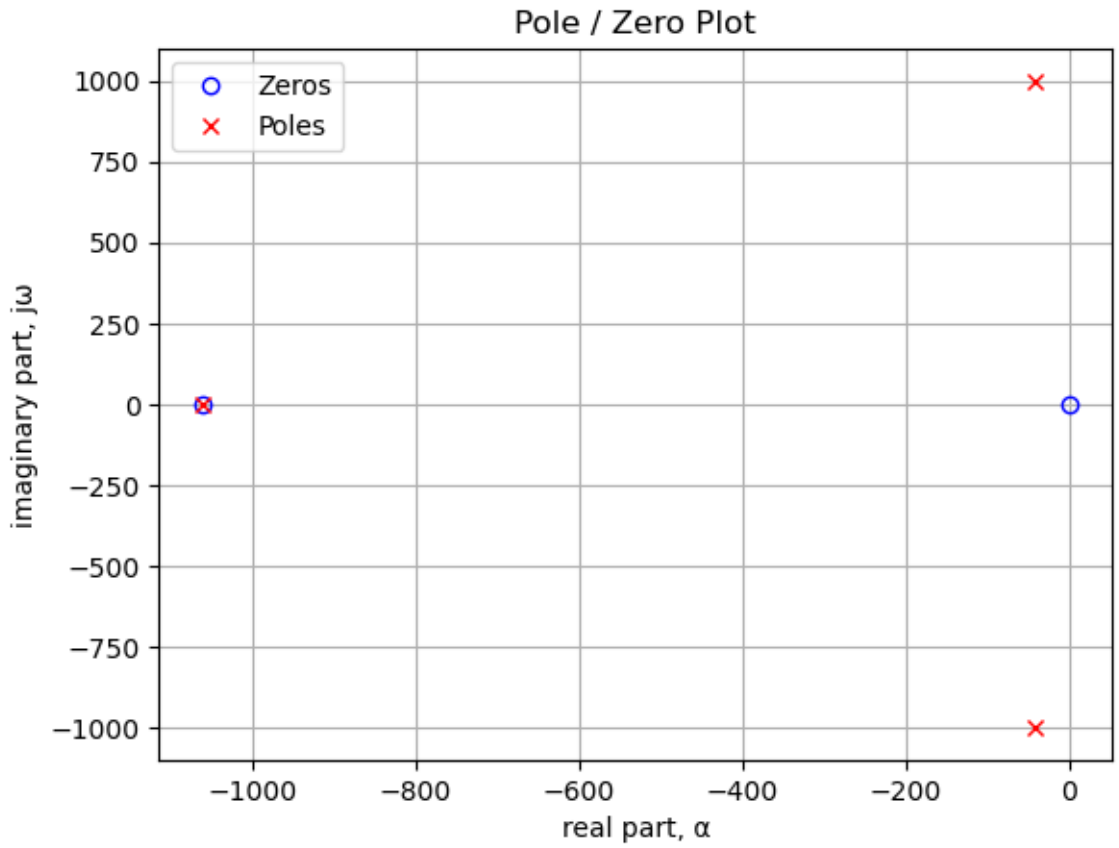

Figure 5: pole zero figure caption

The table below lists the values of the pole and zero locations.

Table 2: table of poles and zeros

Zeros, Hz	Poles, Hz
-1061.03	-1061.03+0.00j
0.00	-42.78+997.93j
	-42.78-997.93j

The zero at -1061.033 Hz and the pole at -1061.033+0.000j Hz cancel.

3.3.2 Magnitude and phase response

The plot of the filter's magnitude and phase response for the transfer function $H(s)$ is shown below.

```
x = np.logspace(1.9, 4.1, 1000, endpoint=True)*2*np.pi # x axis data values
w, mag, phase = signal.bode(sys_tf, w=x) # returns: rad/s, mag in dB and
                                         # phase in deg

fig, ax1 = plt.subplots()
ax1.set_ylabel('magnitude, dB')
ax1.set_xlabel('frequency, Hz')

plt.semilogx(w/(2*np.pi), mag,'-b')    # magnitude plot

ax1.tick_params(axis='y')
plt.grid()

# instantiate a second y-axes that shares the same x-axis
ax2 = ax1.twinx()
color = 'tab:red'

plt.semilogx(w/(2*np.pi), phase,':',color='tab:red')  # phase plot

ax2.set_ylabel('phase, deg',color=color)
ax2.tick_params(axis='y', labelcolor=color)

plt.title('Magnitude and phase plot')
plt.show()
```

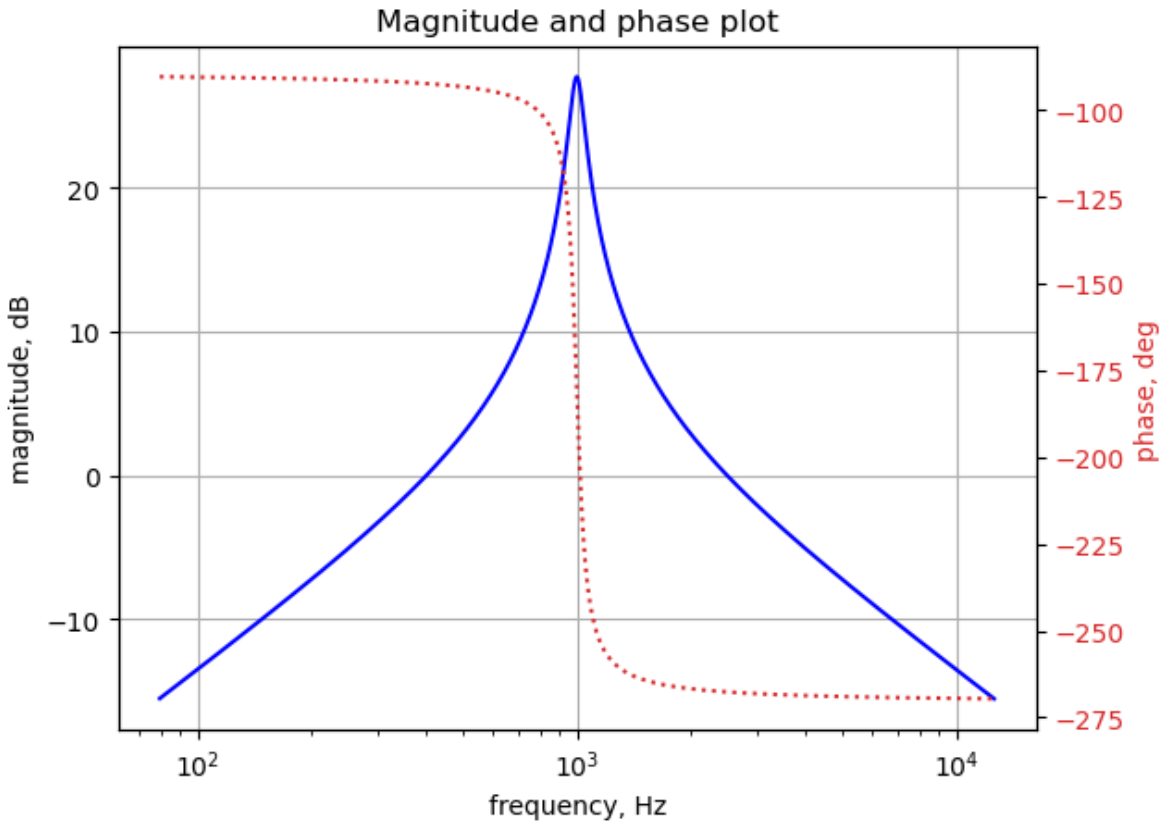

Figure 6: figure caption

The filter has a maximum gain of 27.7 dB at 997.5 Hz and the -3dB frequency is 1044.0 Hz. The attenuation to the right of the peak is 41.3 dB per decade and the filter's quality factor, Q, is 12.32.

The filter Q factor can also be calucated from the element values.

Calculated Q factor: 11.673

The calculated Q factor is a bit different than the the Q measured from the frequency response plot.

The envelope of oscillation decays proportional to $e^{-\alpha t}$, where α is the attenuation parameter calculated above.

```
alpha = atten_param.subs({R:15000,R2:186000,C:1e-8,C2:9.1e-10})
Markdown('$\\alpha$ = {:.3f}'.format(alpha))
```

α = 268.817

3.3.3 Step response

The step response of the is calculated using the SciPy function lsim, which can be used to simulate output of a continuous-time linear system from the continuous-time linear time invariant system base class. The function lsim allows us to evaluate the performance characteristics of the circuit to square wave input.

A square wave with a frequency of 25 Hz and an amplitude of one volt peak to peak is used as the input to the filter.

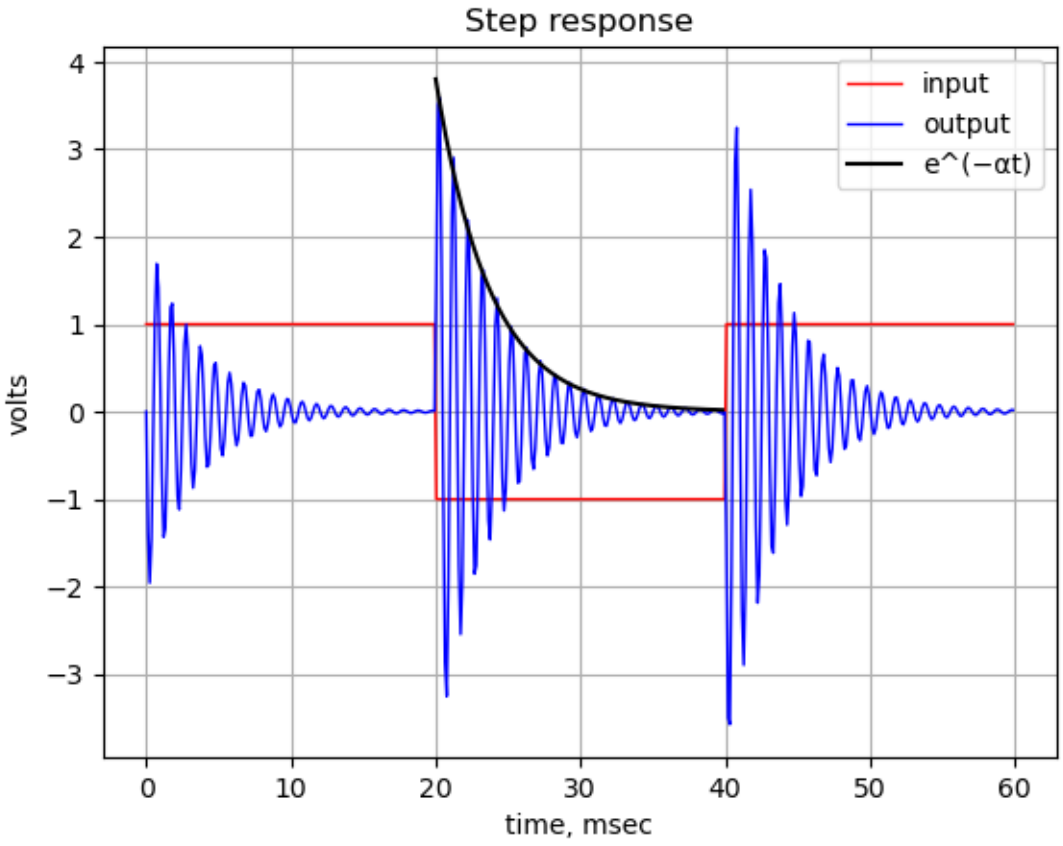

Figure 7: figure caption

The filter's response to a square wave input shows a damped oscillation that occures at each edge of the square wave. The frequency of the oscillation is about 10 cycles over 10 ms, which is 1000 Hz and corresponses to ω_n = 1kHz. The decay time constant is about 10 ms.

3.3.4 Group delay

Group delay is a measure of the time delay experienced by a group of frequencies as they pass through a system. It's essentially the rate of change of the phase response with respect to frequency. Group Delay is important for the following reasons:

- Signal distortion: Variations in group delay across different frequencies can cause signal distortion, affecting the quality of audio, video, and data transmission.
- System design: Understanding group delay is crucial for designing systems with linear phase characteristics, which minimize distortion.
- Pulse propagation: In fields like optics and telecommunications, group delay affects the shape and timing of pulses.

Group delay (τ_g) is calculated as the negative derivative of the phase response (ϕ) with respect to angular frequency (ω):

$\tau_g(\omega) = -\frac{d\phi(\omega)}{d\omega}$

A system with a constant group delay is called a linear phase system. These systems introduce a pure time delay to all frequency components without altering their relative phase relationships, preserving the original waveform shape. According to Audibility of Group-Delay Equalization, the threshold is 2 ms. The abstract for the paper states:

> The audibility thresholds for group-delay variation from several previous related studies are shown in Fig. 1. If not otherwise stated, these studies have been conducted using headphones. Green applied Huffman sequences, or truncated impulse responses of second-order allpass filters, to study the audibility of phase distortion. He found a threshold value for the peak group delay of about 2 ms for center frequencies of 625 Hz, 1875 Hz, and 4062 Hz.

```
plt.title('group delay')
plt.semilogx(w/(2*np.pi), -np.gradient(phase*np.pi/180)/np.gradient(w)*1000,
    '-',label='group delay')

plt.ylim((0,4))

plt.ylabel('Group delay, msec')
plt.xlabel('Frequency, Hz')
plt.grid()
plt.show()
```

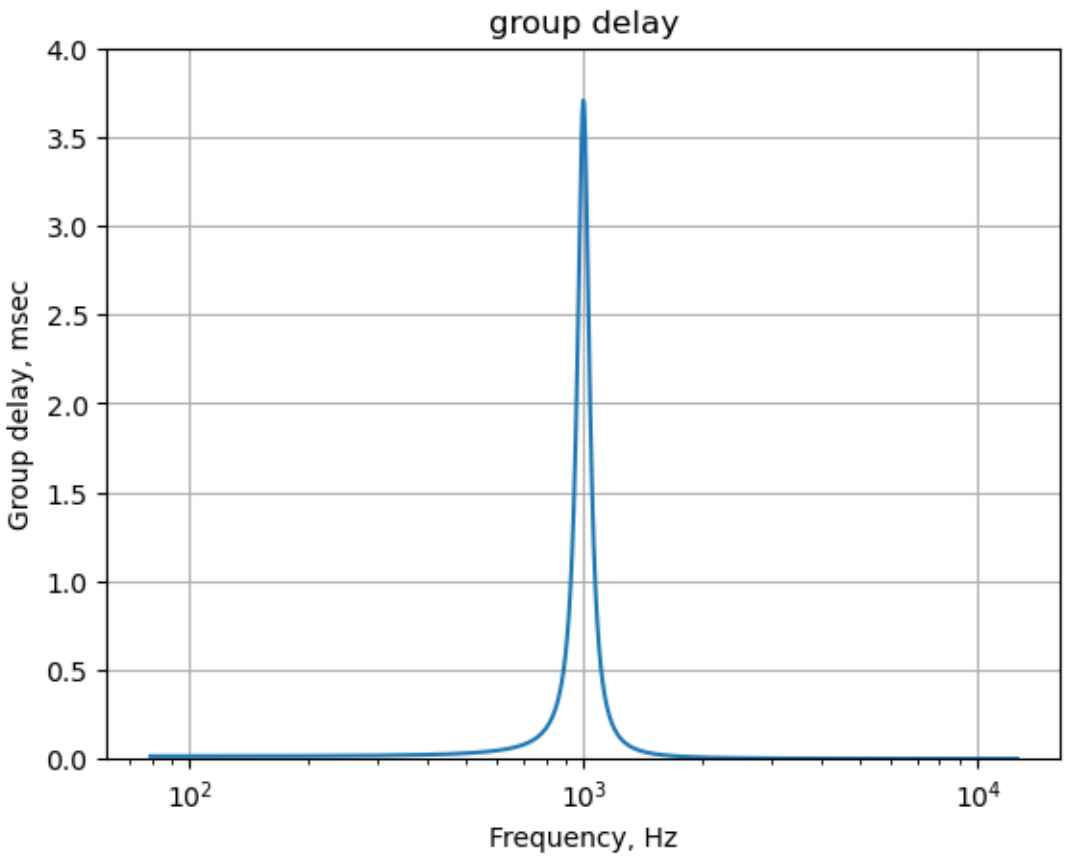

Figure 8: figure caption

3.4 Sensitivity Analysis

When look at a filter, especially in the context of a design review, the circuit designer will want to examine the circuit sensitivity.

Prototype Sensitivity Analysis in seperate notebook.

Twin-T_BPF_Sensitivity_Analysis.ipynb

3.4.1 Monte Carlo Tolerance Analysis

- use 1000kHz filter with values from table I

3.4.2 Worst Case Tolerance Analysis

- use 1000kHz filter with values from table I

Add link to jupyterlab notebook in github.

https://github.com/Tiburonboy/EE_jupyter_notebooks/blob/main/Buchla%20Twin-T%20Active%20Filter/Twin-T_BPF_Sensitivity_Analysis.ipynb

4 FILTER DESIGN EXAMPLES

This section will some through the design steps to implement a filter.

4.1 Example 1, where $R_1 = R_3 = R$ and $C_1 = C_3 = C$

In this example, we want to design a filter with a center frequency of 1.4kHz and a filter Q of approximately 3. The filter in Figure 1 is simplified by using $R_1 = R_3 = R$ and $C_1 = C_3 = C$. Usually it's more convienient to pick capacitor values from standard values and solve for resistors. The standard values for resistors and capacitors follow the E-Series of numbers. The table below can be used to select values for C and R to put the filter response in the ball park for the desired center frequency.

Frequency Range	C	R
10 to 500 Hz	1 μF	10 kΩ
500 to 1 kHz	100 nF	10 kΩ
1 to 5 kHz	10 nF	10 kΩ
5 to 10 kHz	1 nF	5 kΩ

The range of Q varies and the values of C and R will need to be adjusted.

The code below will plot the range of R_2, C_2 and filter Q for a given center frequency, R and C. Note some of the lines of code were to long to display across the page, so I broke a few of the longer lines at commas and periods in the code to control where the lines are wrapped.

```
E6_cap_nF_list = np.array([2.2, 3.3, 4.7, 6.8, 10, 15, 22, 33, 47])*1e-9

C2_plot = E6_cap_nF_list

R2_plot = np.zeros(len(C2_plot))
Q_plot = np.zeros(len(C2_plot))
```

```
freq = 1.4 # kHz
C_value = 10e-9
R_value = 10e3

fig, ax1 = plt.subplots()

# solve for R2 for each C2 in the capacitor list
for i in range(len(C2_plot)):
    R2_plot[i] = solve(omega_n.subs({R:R_value, C2:C2_plot[i],
        C:C_value})-2*np.pi*freq*1e3,R2)[0]
    Q_plot[i] = Q_factor.subs({R:R_value, R2:R2_plot[i],
        C:C_value, C2:C2_plot[i]})
    plt.annotate('{:.1f}nF'.format(C2_plot[i]*1e9), xy=(C2_plot[i]*1e9,
        R2_plot[i]/1e3), xytext=(5,2), textcoords='offset points')

ax1.set_ylabel('R2, k ohms')
ax1.set_xlabel('C2, nF')

# set the color to white to hide the plot
plt.plot(C2_plot*1e9,R2_plot/1e3,'x-w')
plt.xlim((0,55))

ax1.tick_params(axis='y')
plt.grid()

# instantiate a second y-axes that shares the same x-axis
ax2 = ax1.twinx()
color = 'k'

plt.plot(C2_plot*1e9,Q_plot,'x-')

ax2.set_ylabel('Q Factor',color=color)
ax2.tick_params(axis='y', labelcolor=color)

plt.title('C2 vs R2 for \u03C9={:.1f} kHz, R={:.0f}k, C={:,.0f} nF'.
    format(freq, R_value/1e3,C_value*1e9))
plt.show()
```

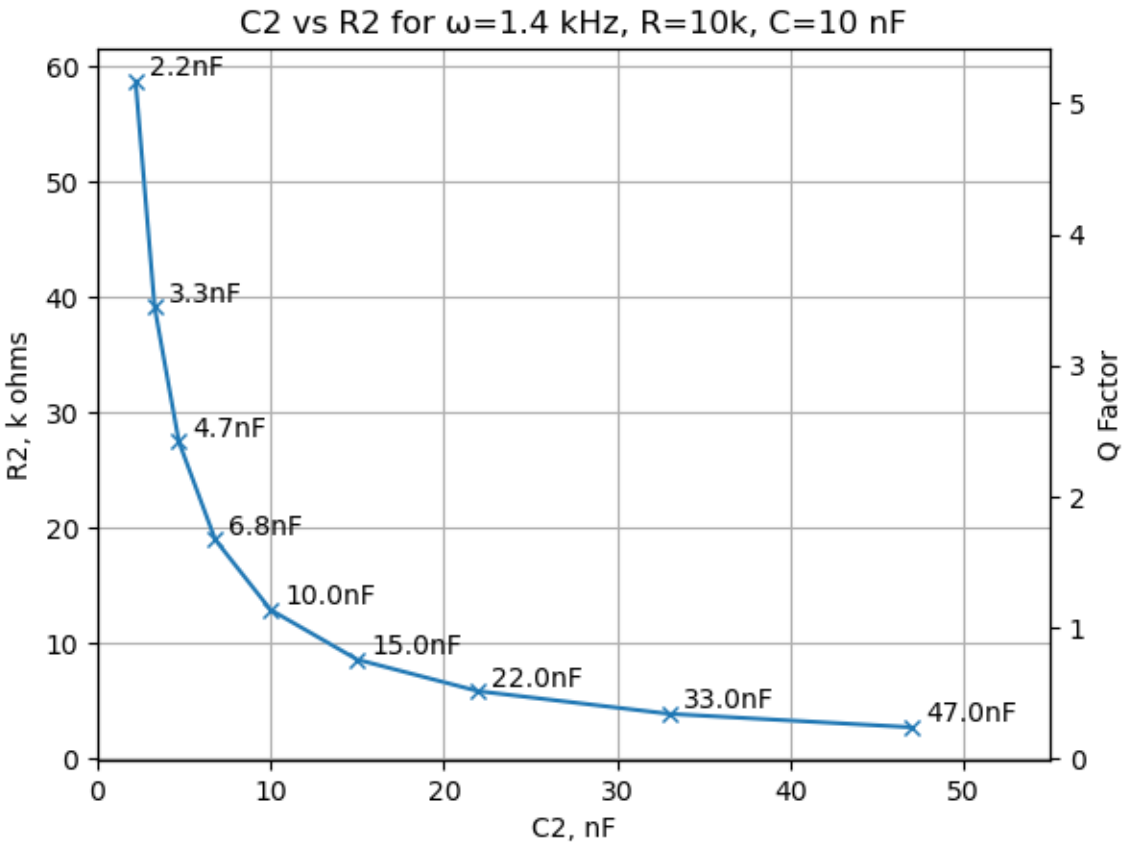

Figure 9: C2 and R2 for frequency and R and C

In the code above the center freqiency is set by assigng a value to the variable, in this case `freq = 1.4`, which has units of kHz. The desired Q is about 3, so $C_2 = 4.7$ nF is chosen.

The plot below shows the filter's peak amplitude versus the value of C_2.

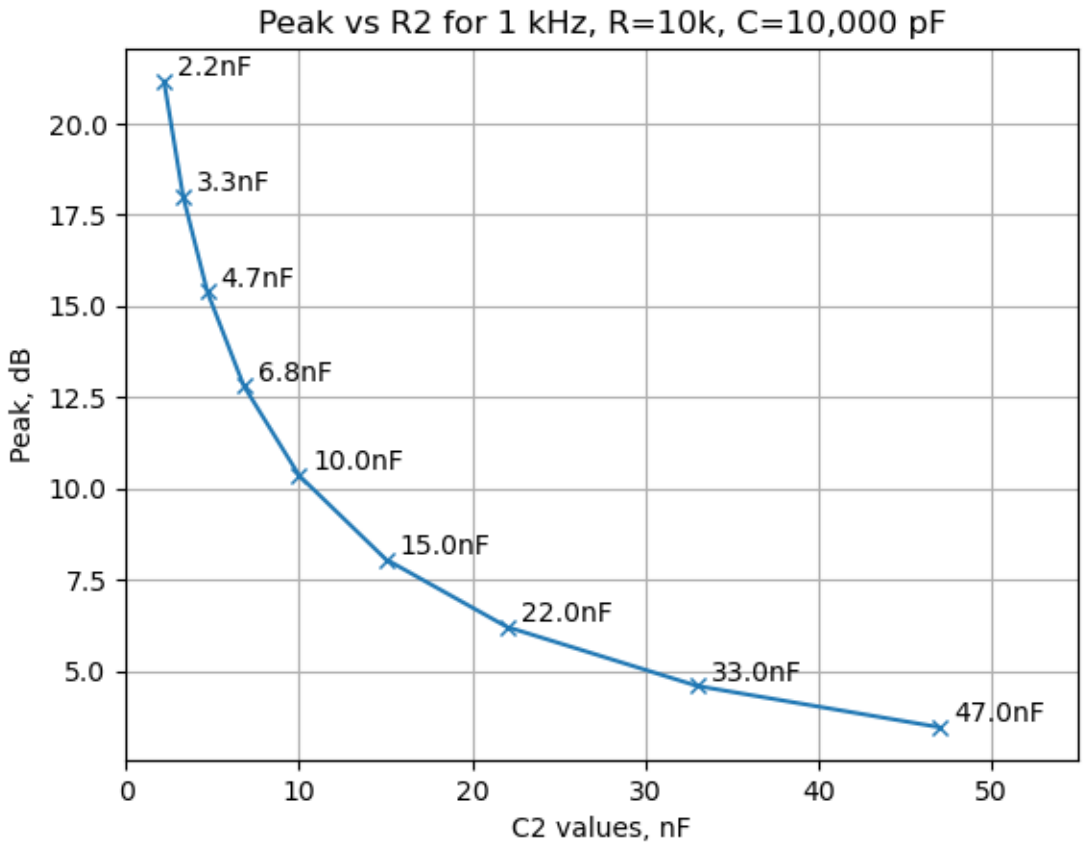

Figure 10: fig cap

We can choose the value of C2, which is $C_2 = 4.7\text{nF}$ and solve for R_2.

```
C2_value = 4.7e-9

R2_value = solve(omega_n.subs({R:R_value, C2:C2_value,
    C:C_value})-2*np.pi*freq*1e3,R2)[0]
Markdown('$R_2={:.3f}k$'.format(R2_value/1e3))
```

$R_2 = 27.497k$

The value happens to be very close to a standard value 1% metal film resistor. Setting R_2 equal to `27,500`.

```
R2_value = 27500
```

The plot of the magnitude and phase are shown below.

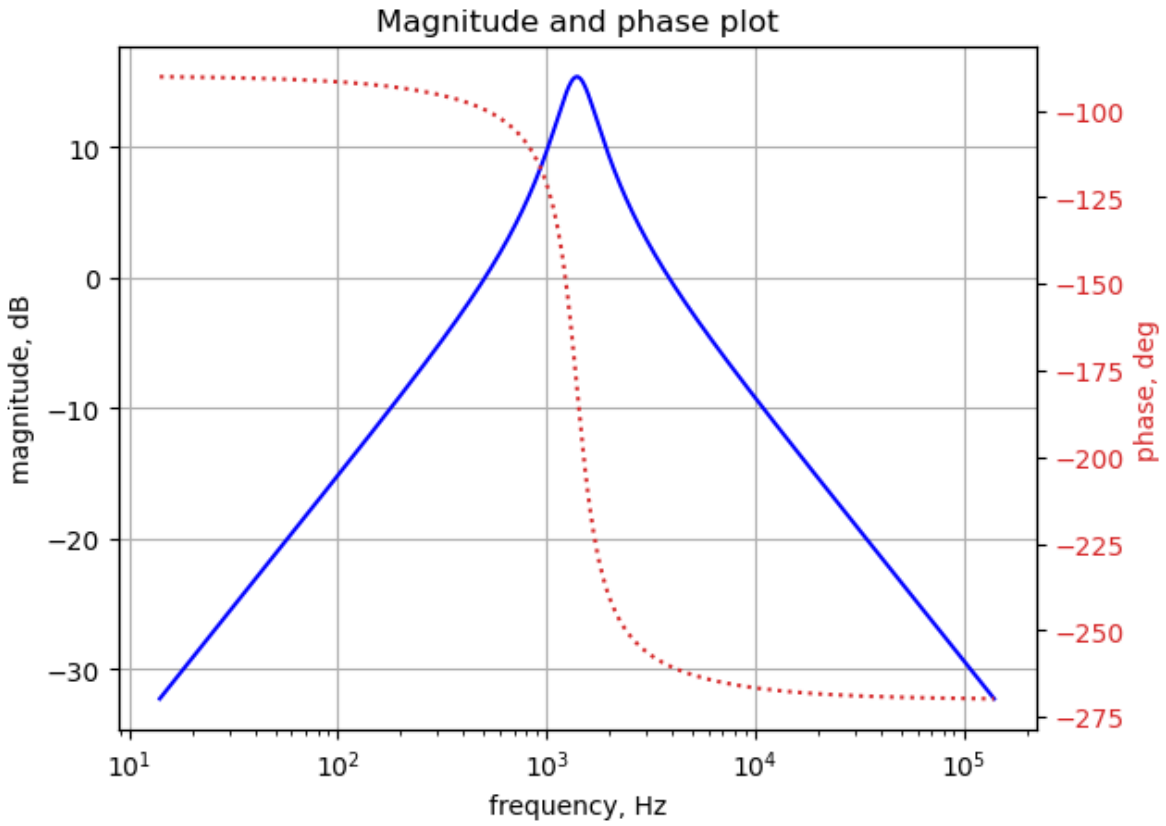

Figure 11: fig cap

The filter has a maximum gain of 15.4 dB at 1393.6 Hz and the -3dB frequency is 1722.7 Hz. The attenuation to the right of the peak is 27.6 dB per decade and the filter's quality factor, Q, is 2.49.

The filter Q factor can also be calucated from the element values.

Calculated Q factor: 11.673

Use the SciPy function TransferFunction is used to represent the system as the continuous-time transfer function.

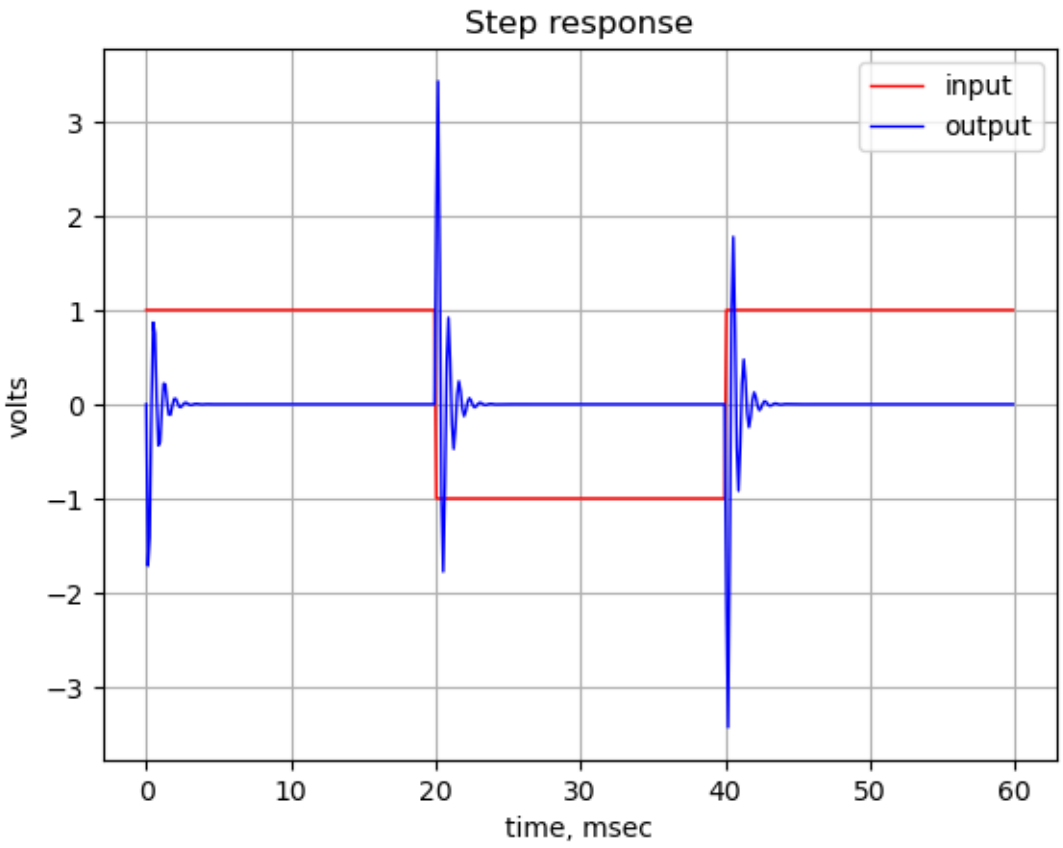

Figure 12: fig cap

4.2 Example 2, Arbitrary Pole Locations

In this example given capacitor values and the pole locations, the values of the resistors are obtained with the help of SymPy's solver function. Three variables p_1, p_2 and p_3 are decleared, which are the three poles of the charactic polyniminal.

I'll write the charactic polyniminal in the form, $(s + p_1)(s + p_2)(s + p_3)$. The code below expands and collects the terms of the equation

```
d_equ = expand((s+p1)*(s+p2)*(s+p3)).collect(s)
Markdown('${:s}$'.format(latex(d_equ)))
```

$p_1 p_2 p_3 + s^3 + s^2 \left(p_1 + p_2 + p_3\right) + s \left(p_1 p_2 + p_1 p_3 + p_2 p_3\right)$

The coefficients of the variable s are:

$d_3 = 1$

$d_2 = p_1 + p_2 + p_3$

$d_1 = p_1p_2 + p_1p_3 + p_2p_3$

$d_0 = p_1p_2p_3$

Divideing each term by d_0, we get the coeeficients in conanical form.

$d_3 = \frac{1}{p_1p_2p_3}$

$d_2 = \frac{p_1+p_2+p_3}{p_1p_2p_3}$

$d_1 = \frac{p_1p_2+p_1p_3+p_2p_3}{p_1p_2p_3}$

$d_0 = 1$

The coeeficients of the filter's charactic equation were obtained above and are:

$a_3 = C_1C_2C_3R_1R_2R_3$

$a_2 = C_2C_3R_1R_2 + C_2C_3R_1R_3$

$a_1 = C_2R_1 + C_3R_1$

$a_0 = 1$

The `scipy.signal` module provides a set of tools for designing and analyzing analog filters. The SciPy functions to design classic analog filters:

- `scipy.signal.buttap`: Butterworth filter
- `scipy.signal.cheb1ap`: Chebyshev type I filter
- `scipy.signal.cheb2ap`: Chebyshev type II filter
- `scipy.signal.besselap`: Bessel filter

These function return (z,p,k) for an analog prototype filter; the second parameter, p, are the poles of the filter. Commenting out the other filter types, we can use `cheb2ap` to calculate the pole locations for this type of filter.

```
N = 3 # filter order
Wn = 1 # normalised frequency

#filter_param = signal.besselap(N)
#filter_param = signal.buttap(N)
rp = 0.8
rs = 5.0
#filter_param = signal.cheb1ap(N,rp)
filter_param = signal.cheb2ap(N,rs)
```

The pole loactions are tabulated below.

Table 4: table of poles and zeros

Pole #	Value, rad/s
1	-0.22-1.02j
2	-2.48-0.00j
3	-0.22+1.02j

In the code below the normalized values of C_1, C_2 and C_3 were chosen arbitraillery. It might be necessary to adjust these values if the resistor values don't end up being positive real numbers. The poles can be a calculated from SciPy filter functions or chosen to be either three postive real numbers or one positive real number and a pair of complex conjegate numbers.

The values for the poles are entered into the code as positive numbers, but …

```
C1_value = 0.7
C2_value = 0.2
C3_value = 0.3
p1_value = -filter_param[1][0] #(1-1j*1)
p2_value = -filter_param[1][1] #(1+1j*1)
p3_value = -filter_param[1][2] #1.2

sub_values = {C1:C1_value, C2:C2_value, C3:C3_value, p1:p1_value,
    p2:p2_value, p3:p3_value}
```

Equatting the coefficients of s we get:

$d_1 = a_1$

$d_2 = a_2$

$d_3 = a_3$

In the expaned version of these equation, we see that there are three equations and three unknowns, which are R_1, R_2 and R_3, since the values for the capacitors and poles have been assigned.

$d_3 = \frac{1}{p_1 p_2 p_3} = a_3 = C_1 C_2 C_3 R_1 R_2 R_3$

$d_2 = \frac{p_1+p_2+p_3}{p_1 p_2 p_3} = a_2 = C_2 C_3 R_1 R_2 + C_2 C_3 R_1 R_3$

$d_1 = \frac{p_1 p_2 + p_1 p_3 + p_2 p_3}{p_1 p_2 p_3} = a_1 = C_2 R_1 + C_3 R_1$

Using the SymPy function `solve`, to solve for the resistor values, the solution is:

```
sol_values = solve(((d3-a3).subs(sub_values), (d2-a2).subs(sub_values),
    (d1-a1).subs(sub_values)),(R1,R2,R3))
Markdown('${:s}$'.format(latex(sol_values[0])))
```

(1.61172938442011, 0.51227330881633, 10.6011406578226)

There are two solutions, so we can just arbitraility chose the first one. Using the second solution means that the value for R_2 and R_3 are swapped.

```
R1_value = sol_values[0][0]
R2_value = sol_values[0][1]
R3_value = sol_values[0][2]
```

The values for the resistors can be put into the newtork equations, which are displayed below.

```
equ_N_ex2 = NE_sym.subs({R1:R1_value, R2:R2_value, R3:R3_value,
    C1:C1_value, C2:C2_value, C3:C3_value})

# display the equations
temp = ''
for i in range(shape(equ_N_ex2.lhs)[0]):
    temp += '<p>${:s} = {:s}$</p>'.format(latex(equ_N_ex2.rhs[i]),
        latex(round_expr(equ_N_ex2.lhs[i],9)))

Markdown(temp)
```

$0 = I_{V1} - 0.7sv_5 + v_1 \cdot (0.7s + 0.620451553) - 0.620451553v_3$

$0 = I_{O1} - 0.3sv_3 + v_2 \cdot (0.3s + 0.094329472) - 0.094329472v_5$

$0 = -0.3sv_2 - 0.2sv_4 - 0.620451553v_1 + v_3 \cdot (0.5s + 0.620451553)$

$0 = -0.2sv_3 + v_4 \cdot (0.2s + 1.952082966) - 1.952082966v_5$

$0 = -0.7sv_1 - 0.094329472v_2 - 1.952082966v_4 + v_5 \cdot (0.7s + 2.046412438)$

$V_1 = v_1$

$0 = v_4$

Solving for voltages and currents in terms of Laplace variable s, we get:

$v_1 = V_1$

$v_2 = \frac{-1.54018451100814 \cdot 10^{44} V_1 s^2 - 2.20352198180951 \cdot 10^{44} V_1 s}{8.4 \cdot 10^{42} s^3 + 2.45569492532383 \cdot 10^{43} s^2 + 1.84138955312286 \cdot 10^{43} s + 2.28498601678766 \cdot 10^{43}}$

$$v_3 = \frac{-8.19874845657021 \cdot 10^{43} V_1 s^2 + 2.28498601678766 \cdot 10^{43} V_1}{8.4 \cdot 10^{42} s^3 + 2.45569492532383 \cdot 10^{43} s^2 + 1.84138955312286 \cdot 10^{43} s + 2.28498601678766 \cdot 10^{43}}$$

$$v_4 = 0.0$$

$$v_5 = \frac{8.4 \cdot 10^{42} V_1 s^3 - 2.34107469483772 \cdot 10^{42} V_1 s}{8.4 \cdot 10^{42} s^3 + 2.45569492532383 \cdot 10^{43} s^2 + 1.84138955312286 \cdot 10^{43} s + 2.28498601678766 \cdot 10^{43}}$$

The network transfer function, $H(s) = \frac{v_2(s)}{v_1(s)}$ is:

$$H(s) = \frac{-1.54018451100814 \cdot 10^{44} s^2 - 2.20352198180951 \cdot 10^{44} s}{8.4 \cdot 10^{42} s^3 + 2.45569492532383 \cdot 10^{43} s^2 + 1.84138955312286 \cdot 10^{43} s + 2.28498601678766 \cdot 10^{43}}$$

Use the SciPy function TransferFunction is used to represent the system as the continuous-time transfer function.

The poles and zeros of the transfer function are plotted below:

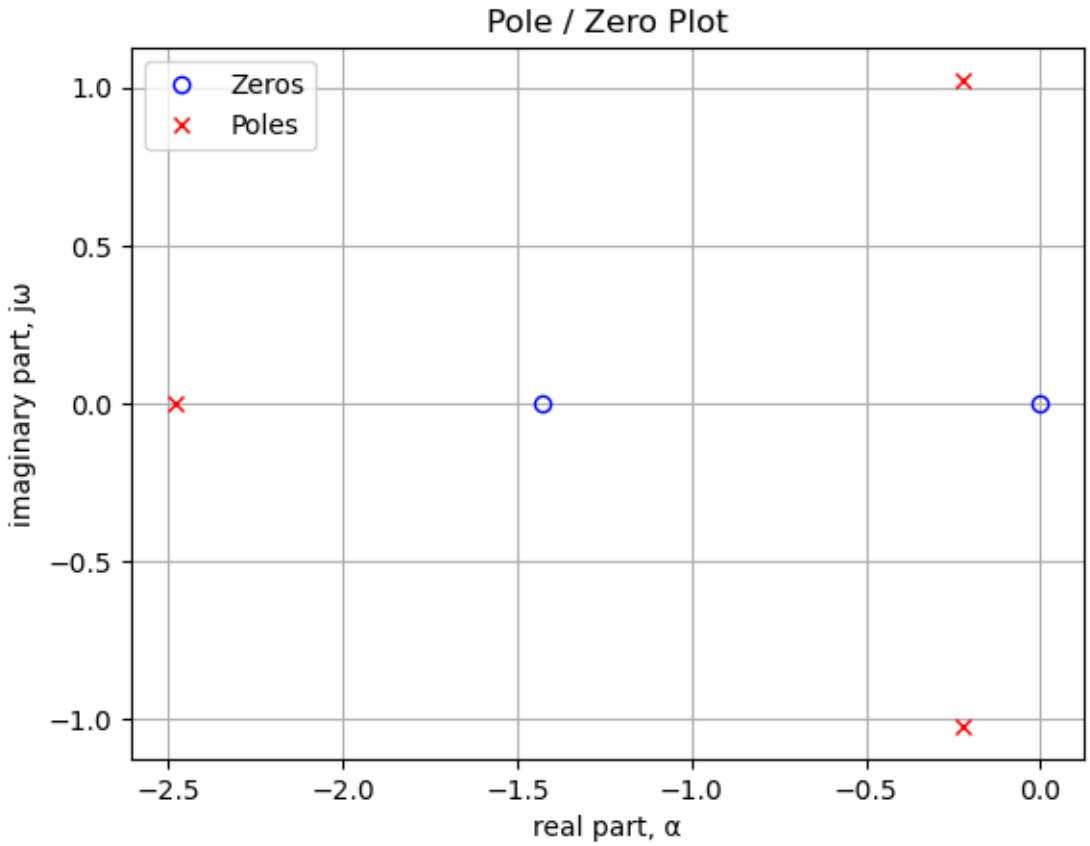

Figure 13: fig cap

The units of the poles and zeros are in radian frequency and are tabulated below.

Table 5: table of poles and zeros

Zeros, rad/s	Poles, rad/s
-1.43	-2.48+0.00j
0.00	-0.22+1.02j
	-0.22-1.02j

The values of the poles network are the same a chosen above, thus validating the solution.

The magnitude and phase of the filter's transfer function is plotted below.

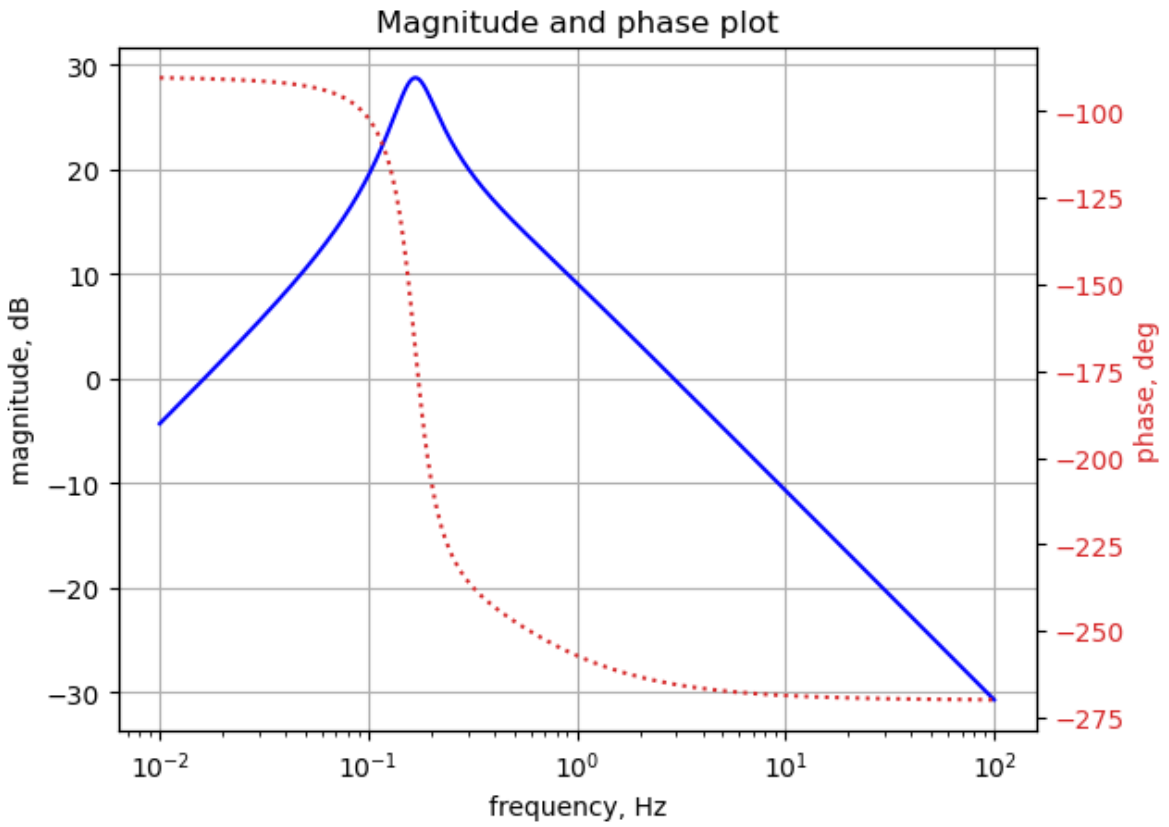

Figure 14: fig cap

The filter has a maximum gain of 28.7 dB at 0.17 Hz and the -3dB frequency is 0.2 Hz. The attenuation to the right of the peak is 24.0 dB per decade and the filter's quality factor, Q, is 2.43.

The step response is plotted below.

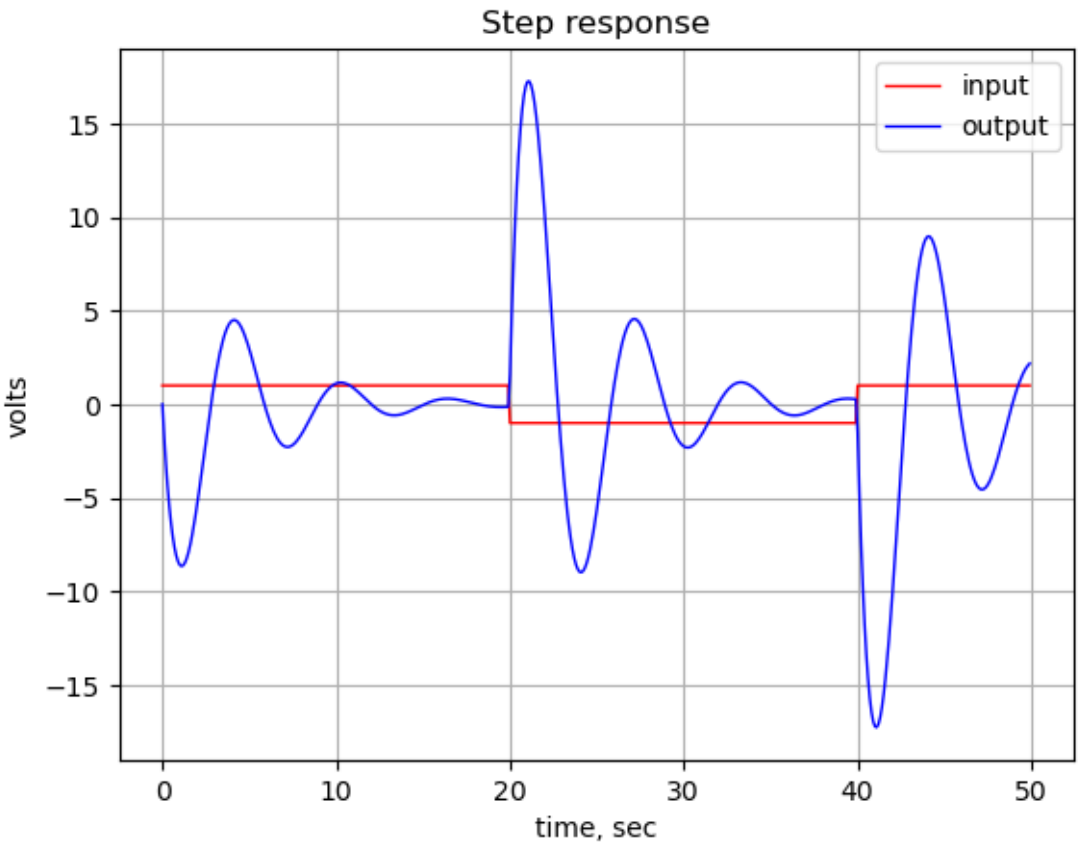

Figure 15: fig cap

The values of the resistors and capacitors can be frequency scaled to put the center frequency of the filter at a different location in the frequency domain.

5 RESULTS

This report has shown that with the help of Python and SymPy electric circuits can be easily analyzed using a mixture of symbolic and numerical methods when following the template presented in this report. A JupyterLab notebook can be used as the "source code" for publishing reports or documents with the help of Quarto.

Analytical expressions were derived for network equations. SymPy was used to solve for the node voltages along with obtaining equations for the poles and zeros. The expressions for the poles in symbolic form are rather long and do not provide much insight into the operation of the filter.

The filter's transfer function naturally has two zeros and three poles which can be reduced to a second order network with constraint of $R_1 = R_3$ and $C_1 = C_3$. The simplified case was analyzed with symbolic coefficients. After substituting the values for the circuit elements the analysis continued.

6 DISCUSSION

The advantages of this filter over other types is than it uses only one Op Amp. By employing the simplification described above, a band pass filter with a given center frequency and Q factor is relatively simple to design sing all of the algebra, calculations and plotting is performed with the help of Python modules. By using the JupyterLab notebook various candatidate filter topologies can be evaluated and compared.

Obtaining analytic expressions for this filter do not appear to be that useful, but are easily obtained with the Python code presented in this report. Most scholarly publications include some analytic expressions and the code presented here can easily generate some of those expressions.

Generating a PDF document from the JupyterLab notebook is easy, however, if particular formatting is required, as for example many journal publications require, additional work would be required. Some online advice is only to use the JupyterLab notebook and Quarto for first and second drafts and then to export the report material to LaTex for publication.

7 CONCLUSION

The use of JupyterLab, Python and Quarto provide a convenient work flow for analyzing electric circuits and presenting the results in a PDF.

add comments on the following:

- analysis of alternatives
- design review documentation
- compare with LTSpice

REFERENCES

[1] Aaron D. Lanterman, "A Bandpass Twin-T Active Filter Used in the Buchla 200 Electric Music Box Synthesizer", https://arxiv.org/abs/2411.11358, last accessed Jan 20, 2025.
[2] C. Ho and A. Ruehli and P. Brennan, "The modified nodal approach to network analysis", IEEE Transactions on Circuits and Systems, 1975

[3] M. Hayes, Lcapy: symbolic linear circuit analysis with Python, PeerJ. Computer science, 2022, https://doi.org/10.7717/peerj-cs.875
[4] Buchla Electronic Musical Instruments, Schematic, 10 Chan. Comb Filter Model 295(A), http://fluxmonkey.com/historicBuchla/295-10chanfilt.htm, last accessed Feburary, 24, 2025
[5] O. Palusinski, Automatic Formulation of Circuit Equations, https://www2.engr.arizona.edu/~ece570/session3.pdf, last accessed January 11, 2024
[x] Tiburonboy, Symbolic Modified Nodal Analysis using Python, https://tiburonboy.github.io/Symbolic-Modified-Nodal-Analysis-using-Python/, last accessed Feburary, 24, 2025

License