International Journal of Engineering and Science Invention (IJESI)
ISSN (Online): 2319-6734, ISSN (Print): 2319-6726
www.ijesi.org ||Volume 13 Issue 4 April 2024 || PP 01-09

Stability aspects of Control Systems through Python Programming

Guguloth Rajender Naik[1] , Nikhitha Muthyala[2], Jatoth Venkanna[3] , Thatipalli Manikanta[4] , Thangalapelli Anjani Kumar[5]

[1](*Associate Professor, Electrical and Electronics Engineering, Kakatiya Institute of Technology and Science Warangal, India)*
[2](*Electrical and Electronics Engineering, Kakatiya Institute of Technology and Science Warangal, India)*
[3](*Electrical and Electronics Engineering, Kakatiya Institute of Technology and Science Warangal, India)*
[4](*Electrical and Electronics Engineering, Kakatiya Institute of Technology and Science Warangal, India)*
[5](*Electrical and Electronics Engineering, Kakatiya Institute of Technology and Science Warangal, India)*

ABSTRACT: *Stability analysis is crucial in control systems engineering, ensuring reliability in dynamic environments. However, real-life systems often pose challenges due to complex transfer functions. This project leverages Python technology to enhance stability analysis methodologies, aiming to develop a comprehensive framework for efficient and accurate analyses across diverse domains. Python's versatility and advanced numerical methods make it a leading platform for stability analysis, complementing traditional tools like MATLAB. With libraries such as control, SciPy, and NumPy, Python offers a unified environment for accessing a wide range of stability analysis tools and techniques. The collaborative efforts within Python's user community contribute to the enrichment of the stability analysis toolbox, facilitating ongoing advancements and broadening the array of tools accessible for real-world applications, including FOMCON and other related applications. Through Python's intuitive interface, this project aims to bridge the gap between theoretical concepts and practical applications in control systems engineering. Cooperation within Python's user community enhances the ecosystem, broadening the toolkit available for stability analysis in real-world scenarios.*

KEYWORDS – *control, fomconpy, matplotlib, numpy, scipy*

Date of Submission: 02-04-2024 Date of Acceptance: 12-04-2024

I. INTRODUCTION

Stability analysis and simulation play pivotal roles in assessing the reliability and performance of control systems before real-world deployment. With the advent of computational tools, particularly Python, the methodologies for stability analysis and simulations have undergone a profound transformation. In this paper, we delve into the intricate domain of stability analysis, emphasizing the significance of examining pole placements to gauge system stability. Leveraging Python's computational prowess, engineers and researchers can delve deep into these complex domains, allowing for meticulous analysis and manipulation of system behavior to optimize stability.

Furthermore, Python's versatility extends across both time and frequency domains, offering a robust suite of tools for simulating and scrutinizing system responses. The utilization of numerical simulations, facilitated by libraries like NumPy and SciPy, ensures efficient and accurate stability assessments, thereby enhancing the reliability of control system designs.[2] To visualize and interpret the results, Python harnesses plotting libraries such as Matplotlib, enabling engineers to generate insightful plots like bode plots. These visualizations provide a clear depiction of how the system responds to various inputs and disturbances, facilitating informed decision-making in system design. Additionally, the integration of signal processing libraries enriches the capability to analyze and interpret system behavior across different domains, further enhancing the efficacy of stability assessments.

Moreover, this paper explores the fractional-order calculus is a rapidly developing area with utilization in diverse domains, including signal processing and semiconductor manufacturing. Despite its potential, the adoption of fractional-order systems in industries has been hindered by computational complexities and costly MATLAB tools. Here, we introduce FOMCONpy[6], a Python library designed to address these challenges, offering a cost-effective alternative for fractional-order system analysis in industrial and IoT applications. Lastly, we present our approach wherein we have implemented stability analysis methodologies in Python[1] and compared the results with those obtained through MATLAB and conventional analysis techniques. This

comparative analysis serves to underscore the effectiveness and reliability of Python-based stability analysis tools, paving the way for their widespread adoption in engineering practices.

II. METHODOLOGY

The proposed methodology emphasizes a harmonious blend of theoretical and practical approaches, facilitating a deeper understanding of control systems stability. The entire process is meticulously documented, including code, assumptions.

Validation through peer review ensures robustness and the project concludes with a succinct summary of key findings, potential applications and avenues for future research in control systems stability. This iterative and collaborative methodology ensures a holistic exploration of stability aspects[5], enhancing the reliability and applicability of the project outcomes.

1. Conventional Analysis

1.1 Time domain analysis

Unit step input

Certainly! Conventional analysis of control systems in the time domain involves several steps, typically starting with the system's transfer function and applying specific input signals to study the reaction of the framework. Using a unit step input as an example, the following is a detailed description:

1.Transfer Function: By using the control system's transfer function, also known as G(s), which connects the Laplace transform of the input to the Laplace transform of the output Y(s).

*U(s): Y(s) = G(s)*U(s)*

2.Unit Step Input: Assume a unit step input U(s)=1/s to analyze the system's response to a sudden change.

3.Laplace Transform: Employ the Laplace transform on the unit step input and the transfer function.

Y(s)=G(s)(1/s)*

4.Partial Fraction Decomposition: Decompose the expression using partial fraction decomposition if the transfer function is a proper fraction. Express G(s) as a sum of simpler fractions.

5.Inverse Laplace Transform: Apply the inverse Laplace transform to each term to obtain the response (y(t)) in the time domain.

$$y(t) = inv[y(s)]$$

6.Time Response Analysis: Analyze the time response y(t) to understand the conduct of the framework over time. Study features such as settling time, overshoot, and steady-state response.

Assess stability based on the time response. A stable system settles to a steady-state without unbounded oscillations.

1.2 Frequency domain analysis

Bode plot:

A Bode plot is a graphical depiction of a system's frequency response. Conventional analysis of a Bode plot involves several steps, typically starting with the system's transfer function. Here's an explanation of the conventional analysis of a Bode plot:

1.Transfer Function: Begin with the transfer function G(s) of the control system, where s is the complex frequency variable.

2.Frequency Domain Representation: Express the transfer function in terms of its magnitude G(jω) and phase <G(jω), where w is the angular frequency.

3.Decomposition into Poles and Zeros: Decompose the transfer function into its poles and zeros. Poles contribute to the magnitude and phase, affecting the system's dynamics.

4.Magnitude Plot: Plot the magnitude 20log|G(jω)| on the y-axis against the logarithmic frequency log(ω) on the horizontal axis.

5.Phase Plot: Plot the phase <G(jω) on the y-axis against the logarithmic frequency log(ω) on the x-axis.

6.Analysis: Analyze the Bode plot to understand the system's behavior in the frequency domain. Identify key features such as bandwidth, resonant frequencies, and phase margins.

7. Adjustments and Iterations: If needed, adjust parameters in the transfer function and repeat the process to observe the impact on the Bode plot.

The Bode plot provides valuable insights into a system's frequency response, making it a crucial tool in control system analysis and design.

2. Practical analysis through python

2.1 Time domain analysis

Unit step input

Certainly! Performing the same analysis using Python involves leveraging the control system libraries, such as control in this case.

Control Library: The control library is a Python library designed for analysis and design of linear control systems.[1]It provides functions for working with transfer functions, state-space models, and frequency domain analysis.You can install the library using the command pip install control.

Matplotlib.pyplot Library: The matplotlib.pyplot library is part of the Matplotlib library and is used to create interactive, animated, and static Python visualizations. In this script, it is used to plot the unit step response.You can install the library using the command pip install matplotlib.

Numpy Library: A basic Python module for scientific computing is the numpy library. Large, multi-dimensional arrays and matrices are supported, as are mathematical operations on these arrays.You can install the library using the command pip install numpy.

2.1.1 Algorithm for Stability Analysis of Control Systems in Python.

Import necessary libraries

Step 1: Define Transfer Function[4]

Step 2: Simulate Unit Step Response

Step 3: Plot Unit Step Response

Step 4: Laplace Transform (Implicit in the transfer function definition)

Step 5: Partial Fraction Decomposition (Handled internally by the control library)

Step 6: Inverse Laplace Transform (Handled during system simulation)

Step 7: Stability Analysis (Can be performed based on time response characteristics)

Step 8: Adjust Parameters and Repeat Analysis if needed

Step 9: Display Transfer Function Details

This algorithm outlines the key steps involved in stability analysis using Python for control systems.

2.2 Frequency domain analysis:

Bode plot:

Certainly! Below is a Python script that utilizes the control and matplotlib libraries to perform a Bode plot analysis. Make sure to install the control library before running the script using pip install control.

2.2.1 Algorithm for Stability Analysis of Control Systems in Python

Step 1: Define Transfer Function

Step 2: Generate Frequency Data

Generate logarithmically spaced frequencies

Step 3: Calculate Bode Plot

Step 4: Plot Bode Plot

Plot Stability Criteria: On the Bode plot, it shows the gain crossover frequency, phase crossover frequency, gain margin, and phase margin.

As necessary, change the system parameters or transfer function to see how they affect the Bode plot's stability criterion.

III. PROGRAMMING

3.1 Time domain analysis

Ex: $H(s) = \frac{4}{(s+4)(s+1)}$

```
!pip install control
import control
import numpy as np
import matplotlib.pyplot as plt
# Define Transfer Function
num = np.array([4])
den = np.array([1, 5, 4])
H = control.tf(num, den)
print('H(s) =', H)
# Calculate step response
t = np.linspace(0, 10, 1000)  # Time vector
```

```
t, y = control.step_response(H, T=t)  # Calculate step response
# Plot step response
plt.plot(t, y)
plt.title("Step Response")
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.grid()
plt.show()  # Display the plot
```

3.2 Frequency domain analysis

Ex: H(s) = $\frac{ks^2}{(1+0.2s)(1+0.02s)}$

```
import numpy as np
import control
# Define Transfer Function
num = np.array([1,0,0])
den = np.array([0.004, 0.22, 1])
G = control.TransferFunction(num, den)
# Create a Bode plot
control.bode(G, dB=True, deg=True, margins=True)
# Calculate stability margins and crossover frequencies
gm, pm, w180, wc = control.margin(G)
# Convert gain margin to dB
gm_db = 20 * np.log10(gm)
# Determine system stability
if gm_db > 0 and pm > 0:
   stability = "Stable"
elif gm_db < 0:
   stability = "Unstable"
else:
   stability = "Marginally Stable"
# Print results
print("System Stability:", stability)
print("Crossover frequency (wc) =", f'{wc:.2f}', "rad/s")
print("Phase margin (PM) =", f'{pm:.2f}', "degrees")
print("Gain margin (GM) =", f'{gm:.2f}')
print("Gain margin (GM) =", f'{gm_db:.2f}', "dB")
```

Note: The following code snippet demonstrates a basic implementation. Adjust the parameters and logic as needed to suit your specific requirements and use case.

IV. FRACTIONAL ORDER CALCULUS

A key mathematical technique in system and control engineering, the Laplace integral transform allows a function (f(t)) to be transformed into F(s), where F(s) is a function of a complex variable. The definition of this transformation is

$$F(s) = L[f(t)] = \int_0^\infty f(t)\,dt$$

Applying the reverse Laplace transform, which is described as follows, will return the Laplace transform F(s) to its original function f(t).

$$f(t) = L^{-1}[F(s)] = \frac{1}{j2\pi}\int_{c-j\infty}^{c+j\infty} F(s)ds$$

Laplace transform for definitions in fractional calculus:

Definition 1: The fractional operator of Riemann-Liouville transformed using Laplace.

$$L[D^\alpha f(t)] = s^\alpha F(s) - \sum_{k=0}^{m=1} s^k\,[D^{-k-1}]_{t=0},$$

$$where(m-1 \le \alpha < m)$$

Definition 2: Laplace transformation of the fractional operator of Grundwald-Letnikov.

$$L[D^\alpha f(t)] = s^\alpha F(s)$$

Definition 3: Caputo's fractional operator transformed via Laplace.

$$L[D^{\alpha}f(t)] = s^{\alpha}F(s) - \sum_{k=0}^{m=1} s^{\alpha-k-1}\, f^{k}(0),$$
$$where(m-1 \le \alpha < m)$$

The Laplace transform finds widespread application in system modelling and control.

4.1 Fractional order system model

A fractional differential equation can be used to explicitly depict a dynamic continuous-time system, as seen below:

$$a_n D^{\alpha_n} y(t) + a_{n-1} D^{\alpha_{n-1}} y(t) + \cdots a_1 D^{\alpha_1} y(t) + a_0 D^{\alpha_1} y(t) =$$
$$b_m D^{\beta_m} u(t) + b_{m-1} D^{\beta_{m-1}} u(t) + \cdots b_1 D^{\beta_1} u(t) + b_0 D^{\beta_0} u(t)$$

Where γ∈ R + is an integer base order and αk and βk = k · γ are stated in terms of γ, the system is said to be of commensurate order. The equation represents an expression of a commensurate order system.

$$\sum_{k=0}^{n} a_k D^{\gamma-k} y(t) = \sum_{k=0}^{m} b_k\, D^{\gamma-k} y(t)$$

Using the Laplace transform formula for Grundwald-Letnikov, we get the following:

$$a_n s^{\alpha_n} Y(s) + a_{n-1} s^{\alpha_{n-1}} Y(s) + \cdots + a_1 s^{\alpha_1} Y(s) + a_0 s^{\alpha_0} Y(s)$$
$$= b_m s^{\beta_m} U(s) + b_{n-1} s^{\beta_{n-1}} U(s) + \cdots + b_1 s^{\beta} U(s) + b_0 s^{\beta_0} U(s)$$
$$Y(s)[a_n s^{\alpha_n} + a_{n-1} s^{\alpha_{n-1}} + \cdots + a_1 s^{\alpha_1} + a_0 s^{\alpha_0}] = U(s)[b_m s^{\beta_m} + b_{n-1} s^{\beta_{n-1}} + \cdots + b_1 s^{\beta+} b_0 s^{\beta_0}]$$

A transfer function for a fractional-order system may be created by using the commutative property of the fractional derivative operator.

$$G(s)=\frac{Y(s)}{U(s)}$$

Remember the corresponding sequence. System of fractional orders γ,

$$G(s) = \frac{Y(s)}{U(s)} = \frac{\sum_{k=0}^{m} b_k\, (s^{\gamma})\text{^}k}{\sum_{k=0}^{n} a_k\, (s^{\gamma})\text{^}k}$$

The function above can be seen as a pseudo-rational transfer function H(Λ), given Λ = s γ. TΛ = s γ.

$$H(\Lambda) = \frac{\sum_{k=0}^{m} b_k\, \Lambda^{k}}{\sum_{k=0}^{n} a_k\, \Lambda^{k}}$$

It is possible to build a state-space model that can describe multiple input, multiple output (MIMO) fractional-order systems by employing the pseudo-rational function notion. Nevertheless, exploring the nuances of the state-space model is outside the purview of this study.[7] However, the transfer function model of system dynamics is still relevant and may produce useful results even in the case of modeling multivariate systems since arrays of transfer functions can be used in these situations.

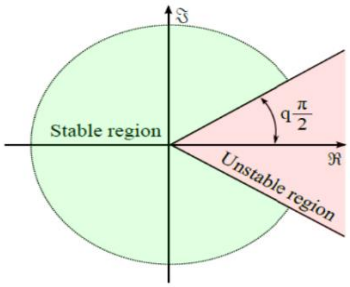

Fig. 1. Stability region of LTI fractional-order system for range: 0 < q < 1

4.2 Fractional order modelling and control

A toolkit for fractional-order modeling and control is called FOMCON. For academics and engineers using fractional calculus in the context of system analysis, identification, design, and optimization, it offers a variety of tools. FOMCON offers features for both theoretical research and real-world applications, making it easier to design fractional-order controllers and analyze fractional-order systems. "Fractional-Order Modeling and Control for Python," or FOMCONpy, is a Python library that has been expanded and is modeled after its MATLAB counterpart, FOMCON. It has extra capabilities designed for the Internet of Things (IoT) and related devices, especially in relation to 5G technology. The creation of a Python library for fractional calculus is anticipated to simplify and make sophisticated fractional-order modeling, identification, and control easier to use in a variety of industrial applications, especially in the Industry.

Graphical user interfaces (GUIs) and backend functions for fractional-order system modeling, identification, design, and optimization, including fractional controllers, are provided by the FOMCONpy package. The goal is

to provide a library for many operating systems that may be used by a variety of users.

There are three separate modules in the library:
1.Module for Fractional-order System Analysis (FOSAM)
2.3System Identification Module for Fractional Order (FOSIM)
.System Control Module of fractional order (FOSCOM)

4.3 FOMCONpy vs. FOMCON

FOMCONpy and FOMCON MATLAB are both tools for fractional-order modeling and control, but they differ in several aspects. With features especially specialized to these situations, such support for 5G technology and interoperability with several operating systems, FOMCONpy was created with the Python environment and the IoT community in mind.Additionally, FOMCONpy[8] provides graphical user interfaces (GUIs) and command-line features, making it accessible to users with different levels of expertise.
On the other hand, FOMCON MATLAB is a toolbox designed for MATLAB, which may require additional licenses for MATLAB's Optimization and Control toolboxes. This can make it costly to use, particularly without a campus-wide subscription. While FOMCON MATLAB offers powerful Fractional-order modeling and control features, as well as compatibility with more recent versions of MATLAB versions and its dependency on additional licenses may pose challenges for users.

4.4 Set up FOMCONpy

For GUI functions, FOMCONpy uses a variety of Python packages and their dependencies, including as numpy, scipy, control, matplotlib, pandas, xlrd, select, addict, and PyQt5. We'll go into depth about the Windows operating system setup procedure.

1. item Install Anaconda. (2019 version was used during development and ensure that python is added to sytem path during installation)
2. verify conda and python is install properly by running the commands below in a terminal
    a. conda --version
    b. python --version
3. Set up Git.
4. Use the command prompt to go to any desired directory.
5. Take FOMCONpy out of the repository (git clone https://github.com/outstandn/fomcon.git).
6. Use "fomconpy" as the current working directory by running cd fomcon
7. Establish a "fomcon" setting:
    a. conda create -n fomcon python=3.7.7 numpy=1.18.1 scipy=1.4.1 pandas=1.0.3 xlrd=1.2.0 matplotlib=3.1.3 pip=20.0.2 pyqt=5.9.2
8. Activate 'fomcon' environemnt
    a. conda activate fomcon
9. install other needed libraries in same 'fomcon' environment using command below
    a. pip install control==0.8.3 addict

4.5 Using FOMCONpy's GUI[6]

Analysis Module for Fractional-Order Systems: Open a command line, go to the directory containing the FOMCONpy code, and type the following command to launch the fractional-order system viewer:

Python pyfomcon.py

4.5.1 Analysis module GUI

The FOMCON GUI, depicted in Fig.2, comprises two group panels:

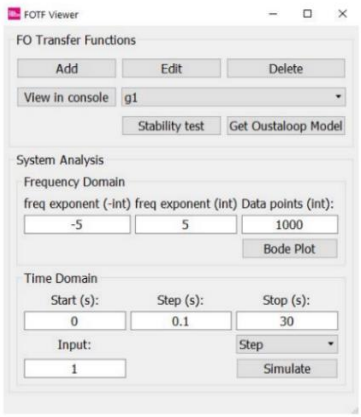

Fig. 2. FOTF viewer

1.FO Transfer Function Panel: The FO Transfer Function Panel makes it easier to add, modify, remove, inspect, verify stability, and convert fractional-order transfer functions. By selecting the Add button, users may add new systems to the drop-down list. This opens a dialog box where users can enter system parameters including delay, zero polynomial, pole polynomial, and unique name. The System Analysis panel's capabilities allow for the management and analysis of several systems

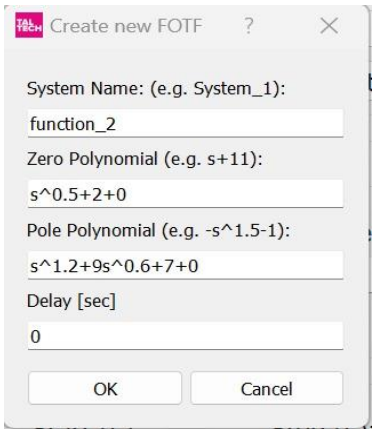

Fig. 3. Create new FOTF

2.System Analysis Panel: Functions for frequency and temporal domain analysis are provided in this section: Frequency Domain Analysis: Plotting amplitude and phase against frequency, the Bode Plot button creates the system's frequency response. Frequency limits may only be specified with integer exponents. Time Domain Analysis: The system's transient response is calculated by clicking the Simulate button. Individuals specify start and stop timings, making that they are both positive integers and in the right sequence.

For example, considering the framework provided by $G(s)=\frac{s^{0.5}+2}{s^{1.2}+9s^{0.6}+7}$, to add this system as g1, users input the system details as shown in Fig.3. Stability can be checked, as illustrated in implementation, where poles inside the red-shaded area represent unstable systems. Furthermore, the step response with a step of 0.1 is calculated in the time interval [0, 30]. It is also possible to analyze the Bode plot in the frequency range [0.00001, 100000] rad/s.

V. IMPLEMENTATION RESULTS

Transfer function:

$$G(s) = \frac{s^{1.2} + 2s^{1.8} + 4s^{2.5}}{s^{0.7} + 3s^{1.4} + 5s^{2.2}}$$

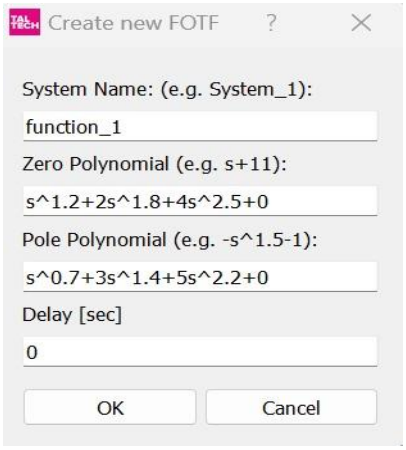

Fig.4. Inputs in new FOTF

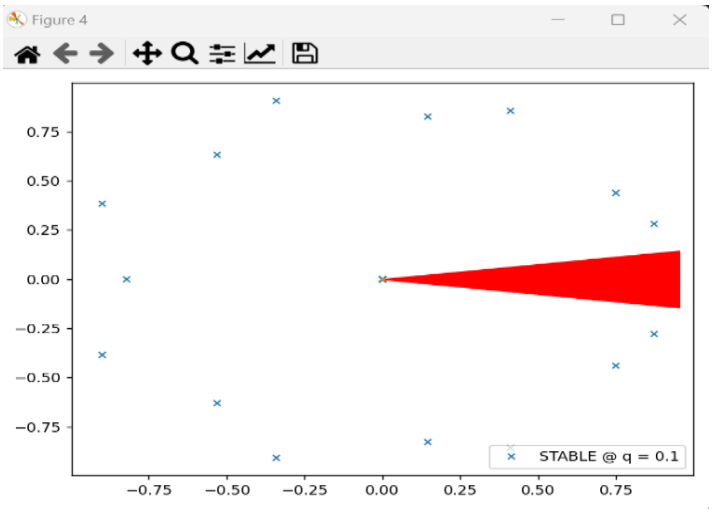

Fig.5. Stability test

Bode plot

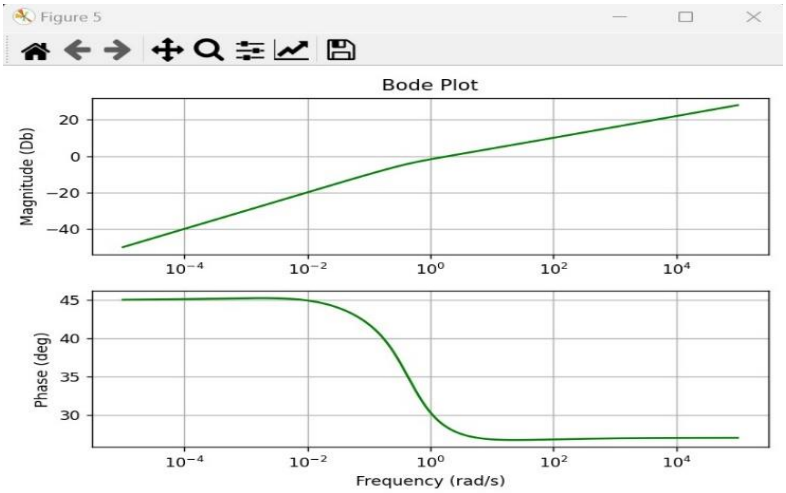

Fig.6. Response through Bode plot

Step response

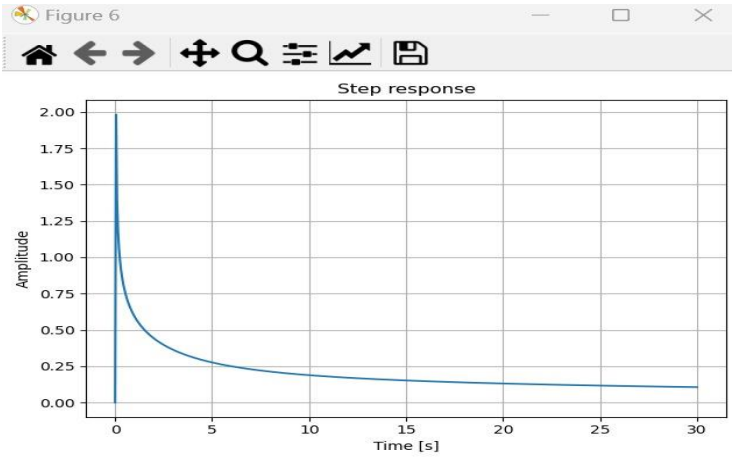

Fig. 7. Stability analysis results of given transfer function

VI. CONCLUSION

The results affirm that stability analysis, facilitated by Python, is both accurate and accessible. The use of emerging technology simplifies complex analyses, showcasing Python's adaptability and ease of implementation. The ease and accuracy of stability assessments, showcased through Bode plot analysis and time domain analysis. The language's versatility and rich ecosystem, exemplified by the control library, enable seamless implementation of complex analyses.

Python's adaptability to handle intricate mathematical operations and because of its simple syntax, stability studies may be conducted by researchers and engineers with ease. [6]Python toolbox for fractional-order systems, was presented, demonstrating the implementation, This presentation served to highlight the practical utility and effectiveness of FOMCONpy in real-world applications involving fractional-order systems. It offers advanced techniques for modelling fractional-order systems. Includes comprehensive features for analysing fractional-order transfer functions. Integrates seamlessly with numpy, scipy, and pandas ensuring efficient data processing algorithms.

REFERENCES

[1] *Sawyer Fuller, Ben Greiner, Jason Moore, Richard Murray, René van Paassen, Rory Yorke ,"Stability Aspects of Control Systems* through Python Programming**"***https://python-control.org*

[2] Stoyan Popov, Nikolay Hinov,"Python Based Electrical Engineering Training for Computer Engineers",16 November 2023,*DOI:10.1109/COMSCI59259.2023.10315903*

[3] L. A. Ribeiro, "Python-based Simulation and Analysis of Electrical Circuits through Transfer Functions," 2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), *Tenerife, Canary Islands, Spain, 2023, pp. 1-6, doi: 10.1109/ICECCME57830.2023.10252502*

[4] Cem Unsalan; Duygun E. Barkana; H. Deniz Gurhan, "Constructing Transfer Function of a System," in Embedded Digital Control with Microcontrollers: Implementation with C and Python , *IEEE, 2021, pp.131-149, doi: 10.1002/9781119576600.ch6.*

[5] Muhammad Nasir Khan; Syed K. Hasnain; Mohsin Jamil; Sameeh Ullah, "6 Control System's Stability," in Electronic Signals and Systems Analysis, Design and Applications: International Edition , River Publishers, *2020, pp.249-282.*

[6] Tobechukwu Onyedi, Aleksei Tepljakov, Eduard Petlenkov, "FOMCONpy: Fractional-order Modelling and Control Library for Python", 2020 43rd International Conference on Telecommunications and Signal Processing (TSP), *pp.239-245, 2020.*

[7] Y. Okuyama, "Stability Analysis of Discrete Event Control Systems Based on Connection Matrices and Graphs," 2019 12th Asian Control Conference (ASCC), Kitakyushu, *Japan, 2019, pp. 955-960.*

[8] J. Mola-Jimenez, J. L. Rueda, A. Perilla, W. Da, P. Palensky and M. van der Meijden, "PowerFactory-Python based assessment of frequency and transient stability in power systems dominated by power electronic interfaced generation," 2018 Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), *Porto, Portugal, 2018, pp. 1-6, doi: 10.1109/MSCPES.2018.8405403.*

[9] T. R. Fernandes, L. R. Fernandes, T. R. Ricciardi, L. F. Ugarte and M. C. de Almeida, "Python Programming Language for Power System Analysis Education and Research," 2018 IEEE PES Transmission & Distribution Conference and Exhibition - Latin America (T&D-LA), Lima, Peru, *2018, pp. 1-5, doi: 10.1109/TDC-LA.2018.8511780.*

[10] A. Tepljakov, E. Petlenkov and J. Belikov, "FOMCON: Fractional-order modeling and control toolbox for MATLAB," Proceedings of the 18th International Conference Mixed Design of Integrated Circuits and Systems - MIXDES 2011, *Gliwice, Poland,08 September 2011, pp. 684-689.*