Cascade Circuit Analyser Final Report

[*]EE20084 - Structured Programming

Jake Stewart
Department of Electrical and Electronic Engineering
University of Bath
Bath, United Kingdom
email: js3910@bath.ac.uk

***Abstract*—This report details the design choices, implementation and testing of the Cascade Circuit Analyser program. Overall it was a resounding success, with low memory footprint, linear time complexity, high speed operation, accurate, modular, scalable and well tested code.**

***Index Terms*—circuit analysis, Python, pytest, regex, software testing**

Contents

I. Overview

This section provides a brief overview of the Cascade Circuit Analyser program, outlining its purpose, functionality, and overall problem-solving structure.

A. `main.py`

This module is the entry point for the application. It is primarily responsible for:

- Handling command line arguments.
- Encapsulating the main execution logic within a try-except block to handle exceptions effectively. This ensures that any unhandled exceptions in the lower modules are caught and managed properly.
- On encountering an exception, it displays the error message, creates an empty CSV file using the specified output argument, and terminates the program with a non-zero exit code to indicate an error state.

B. `net_parser.py`

This module is crucial for interpreting the netlist file into a usable format:

- The `parse_net_file_to_circuit` function reads the input file line-by-line, trimming whitespace and ignoring comments and empty lines.
- It recognizes different sections in the input file (e.g., component definitions or configurations) by matching lines with predefined delimiters.
- It validates the sequence of these sections to prevent processing errors and uses regular expressions to parse relevant data from each line.
- Extracted data is then converted into dictionaries or objects and integrated into a circuit object, facilitating structured access and manipulation in later stages.

C. `circuit.py`

This module defines the `Circuit` class, organizing all circuit-related data:

- The `Circuit` class is designed to store detailed information about the components, terminations, and outputs of a circuit.
- It contains sub-classes for components and outputs, which help in maintaining structured and easily accessible data.
- Its `solve` method plays a pivotal role by sorting components, computing the ABCD matrix for each frequency, and then writing these calculations to an output file.
- This method ensures that the computations adhere to the engineering requirements specified in the output file format, handling data consistently and accurately across various simulations.

D. ***`csv_writer.py`***

Responsible for all CSV file operations, this module:

- Implements functions to write header rows, data rows, and generate an empty CSV file in case of errors.
- Ensures data is formatted in scientific notation, with precise alignment and scaling, whether for displaying results in decibels or linear formats.

E. ***`tests`***

This directory contains all test files, including unit tests and integration tests. The aim was to achieve a high level of code coverage and ensure that the program functions correctly under the typical scenarios proposed in the example netlist files - but also extend to edge cases and cover most failure modes that I could think of.

II. Implementation

This section provides a detailed explanation of the implementation of the Cascade Circuit Analyser program, focusing on the key modules and functions that contribute to its functionality.

A. ***`net_parser.py`***

This module is pivotal for parsing the netlist file and converting its contents into a structured format that the program can manipulate and analyze. Here, we detail the key functions:

- **`parse_net_file_to_circuit`**: This function serves as the starting point for processing the input netlist files.
  - **Functionality**: It initializes an empty circuit instance and begins reading the input file line by line. Each line is stripped of whitespace and checked for comments or emptiness, which are ignored.
  - **Delimiters and Section Matching**: The function uses a match case to identify section delimiters (such as headers for components, outputs, etc.). It ensures that sections are processed in a valid sequence, raising an error if an unexpected section is encountered.
  - **Data Handling**: Lines that do not match section delimiters are considered as data. If data appears outside of its designated section, the function raises an error. Otherwise, it is passed to specialized line processing helper functions for parsing.
- **`process_*_line`**: These functions are tasked with processing specific types of data lines, depending on the section they belong to.
  - **Data Extraction**: Each line of data is matched against a tailored regex pattern that extracts necessary information and returns it in the form of a dictionary.
  - **Component Addition**: The extracted data dictionary is passed to the appropriate `add_*` method of the circuit instance. This method is responsible for creating and appending a new instance of the respective Circuit Component or Output class, or updating the termination dictionary.
  - **Use of Regex**: Regular expressions play a crucial role in ensuring reliable and well-defined data extraction, including handling optional fields like scientific notation, magnitude and decibel prefixes. Magnitudes are applied immediately to standardize data storage format, with multipliers stored in a dictionary for quick access and conversion. Only words from a predefined list are accepted to ensure data integrity.
  - **Error Handling**: If a regex match fails or matches incorrectly, an error is raised to prevent incorrect data handling and ensure the integrity of the circuit data.
  - **Regex Development**: Regex patterns were developed and refined using https://regex101.com/, a tool that allows for meticulous testing and adjustment of regex expressions.

This detailed implementation ensures that the `net_parser.py` module robustly handles the parsing of the netlist file, setting a solid foundation for the subsequent circuit analysis processes.

B. ***`circuit.py`***

The `circuit.py` module encapsulates the `Circuit` class, which is the core data structure for organizing and manipulating the circuit elements and their interactions in the Cascade Circuit Analyser program. This class and its methods are designed to store circuit data, compute necessary parameters, and facilitate the overall solution of the circuit analysis. Below are detailed descriptions of the key methods within this class:

- **Initialization:**
  - **Functionality**: The constructor method, `__init__`, initializes the lists for storing component and output instances, and a dictionary for terminations. This setup is crucial for managing the diverse types of data that are typical in circuit analysis applications.
- **Adding Components:**
  - **`add_component`**: This method accepts parameters defining a component (e.g., type, nodes, value) and creates an instance of a component class (e.g., Resistor, Capacitor). Each component class contains a method to calculate its specific ABCD matrix, which is critical for the network analysis.
  - **Data Organization**: Added components are organized in a list that maintains the order of their insertion, which is important for the sequential processing and analysis of the circuit.
- **Setting Terminations:**
  - **`set_termination`**: This method inputs termination values, such as source and load impedances. It updates the terminations dictionary, which is later used in matrix calculations to apply boundary conditions essential for accurate circuit analysis.

- **Adding Outputs:**
  - **`add_output`**: Specifies which outputs need to be calculated and recorded, such as voltages at specific nodes or overall power dissipation. This method ensures that the output specifications are stored for later use in determining what results are generated and stored.
- **ABCD Matrix Calculation:**
  - **`calculate_abcd_matrix`**: This method is responsible for computing the ABCD matrices of each component at the specified frequencies. It performs matrix multiplications of individual component matrices to derive an overall ABCD matrix for the circuit at each frequency.
- **Circuit Solution Method:**
  - **`solve`**: Acts as the main computational engine of the module. It iterates over the frequency range, computes the total ABCD matrix for each frequency using `calculate_abcd_matrix`, and calculates the circuit's response based on the terminations. The results for each requested output are formatted and passed to the `csv_writer.py` for output to a file.
  - **Efficiency and Accuracy**: The efficiency of this method directly influences the program's performance, ensuring that the calculations are done in a logical sequence based on the accurate mathematical foundations of network theory.

This structured approach within the `circuit.py` module ensures comprehensive management and manipulation of circuit data, facilitating detailed and accurate circuit analysis. The module is designed to be flexible and robust, allowing for future enhancements and adjustments to the methods as new requirements or improvements are identified.

C. *`csv_writer.py`*

The `csv_writer.py` module is dedicated to managing the output of the circuit analysis results into a CSV file format. This module plays a crucial role in ensuring data is accurately and effectively written to facilitate easy review and further processing. Below, we detail the functions implemented within this module and their interactions:

- **Write Header:**
  - **Functionality**: This function writes the header row of the CSV file, which typically includes the names of the variables being recorded, such as frequency, voltage, current, etc.
  - **Dynamic Headers**: Depending on the outputs specified in the analysis, this function dynamically adjusts to include the appropriate headers, ensuring the output file matches the expected format for subsequent analysis.
- **Write Data Rows:**
  - **Functionality**: Following the header, this function is responsible for writing the data rows into the CSV file.
  - **Data Formatting**: The data is scaled to match the output magnitude, and formatted in scientific notation to ensure precision, and each value is appropriately aligned and spaced to maintain a structured and readable format. This is particularly important for ensuring the data can be easily imported and manipulated in data analysis tools, or by a human reader.
- **Write Empty CSV:**
  - **Error Handling**: In cases where the program encounters an error that prevents the completion of the circuit analysis, this function generates an empty CSV file.
  - **Indication of Failure**: The creation of an empty CSV file serves as an indication that the analysis did not proceed as expected, which can be a critical signal for automated systems or further batch processing that relies on the output data.
- **Scientific Notation and Scaling:**
  - **Precision Management**: The module is designed to handle numerical data in scientific notation, allowing for a compact representation of large or small numbers without loss of precision.
  - **Adaptive Scaling**: Depending on the data type (e.g., voltages, currents) and the specified units (e.g., decibels for logarithmicly scaled results, milli, micro), the module scales the data appropriately. This feature is crucial for ensuring the data's usability in various analytical contexts where specific units or scales are required.

This module's design ensures that the results of the circuit analysis are documented in a manner that is both precise and adaptable to various needs, facilitating easy data management and analysis post-simulation.

III. Testing

This section discusses the testing strategies employed for the Cascade Circuit Analyser program, all tests were done using the pytest framework. This was chosen over doc tests as it allows for more flexibility in testing and is more readable. It is also well supported by my IDE, VSCode. A screenshot of all tests passing, as well as profiler icicle graphs can be found in the appendix of this report 3.

A. *`test_net_parser.py`*

This test file focuses on validating the functionality of the `net_parser.py` module:

- **Test Scope**: The tests in this file check the parsing functionality for various input formats and edge cases, ensuring that the module correctly interprets different sections and data types within a netlist file.
- **Error Handling**: These tests also cover cases where input data might be malformed or incorrectly structured, verifying that the module raises appropriate errors or handles the issues gracefully.

- **Regex Validation**: The tests include scenarios to validate the regex patterns used for data extraction, but also ensure that even correctly matched data is valid, e.g. nodes cannot be the same, or greater than 1 apart from each other.
- **Sorting** The tests also check that the components are sorted correctly with manual instances of each component type, validating the order agnostic sorting algorithm.

B. ***test_circuit.py***

This test file targets the `circuit.py` module, which is central to the circuit analysis functionality:

- **Test Scope**: The tests focus on verifying the proper creation of components, outputs, and terminations, as well as the accuracy of the ABCD matrix calculations.
- **Functional Testing**: These tests ensure that the core functionality, such as sorting components and solving the circuit, is accurate and robust.
- **Integration**: This file also includes integration tests to check that the module interacts correctly with other parts of the program, such as the `net_parser.py` and `csv_writer.py` modules.

C. ***test_csv_writer.py***

This test file evaluates the output functionality of the `csv_writer.py` module:

- **Test Scope**: The tests ensure that headers, data rows, and error cases are correctly handled, and that output files conform to the required formats.
- **Edge Cases**: The tests cover edge cases such as empty input data or malformed entries, validating the module's error handling and fallback mechanisms.

D. ***test_performance.py***

This test file focuses on evaluating the performance of the program:

- **Performance Metrics**: The tests measure execution time for different sizes and complexities of input data, ensuring that the program meets performance requirements and handles large datasets efficiently.
- **Optimization**: The tests help identify bottlenecks and areas for optimization, providing feedback for improving the program's efficiency through the amount of cycles and time spent in each function. This was created as the auto-tester had too much overhead in the csv comparison.

  Records of the cProfile runs can be found in the `Tests/Profiles/` directory are included in the repository - I would recommend analysis via the snakeviz package.

  This testing was invaluable for optimising and identifying better ways of solving, mostly by pulling as much out of main loops as possible. I was very pleased with the performance of the program, with the most complex test case `Ext_e_Ladder_400.net` taking only 250ms to run 1, and most others in under 2ms 2, the overall sequential profiling was under 750ms for all tests including profiling overhead, which I am very happy with.

  Ext_e_Ladder_400.net was the most complex test case, due to its larger number of components being re-evaluated at a large number of frequency steps, but this time complexity is still a linear function of the number of components and frequency steps.

  Overall memory footprint is very low as results are not saved, just written as they are calculated to the CSV, which also helps with the speed. Even testing this with bomb.net, a netlist expansion of e_Ladder_400 with 125000 components and an unreasonable number of frequencies, the program was able to chug along nicely at 5 rows written per second - this is a testament to the modularity and efficiency of the program. The frequeny independant calculation poses a good opportunity for parallelisation in the future, but this was not pursued as on these small scales it is not necessary and would likely complicate the code and introduce overhead.

E. ***AutoTest.py***

AutoTest was mostly used for testing the program as a whole, using the `diff` command to compare the output CSV files with the model files, reading the logs by hand, was very helpful for reaching format parity. Sometimes hand calculations were done, but due to the complexity of the calculations, this was arduous not always possible, IDE debugging to step through the program and track variables was the most sane approach here, with the model CSVs being the only real metric of success that the values are correct - and don't just look correct.

Other apporaches used to validate the program were print statements in the net parser to fully regenerate the netlist file from the parsed arguments to validate the integrity of the data - these (and all print statements) are commented out for the final version of the program for performance reasons - but were useful as the structured storage of data is hard to observe from the debugger.

F. ***Error Handling***

At every step of the way, checks and exceptions are used with anything regarding user input data, and this robustly meant empty CSVs were written when appropriate, and assisted in the debugging process by providing clear error messages and stack traces. Some good eamples of this were on the linspace and logspace functions, checking the dictonary keys for the correct values by using the get method to default to None if the key was not present, and then checking for None values to raise an error. Likewise in the termination calculations dictionary, we use this strategy to determine if we are solving for a thevenin or norton equivalent, and if the values are present to calculate them.

IV. Conclusion

Overall this project was a great learning experience on end to end software development, the code is as easy to read as I could make it, with structure and OOP at its core. It passed all the model testing and profiling with flying colours, and I am very happy with the performance and simplicity of the program - a key goal hit.

V. Appendix

[1] `Ext_e_Ladder_400.prof`

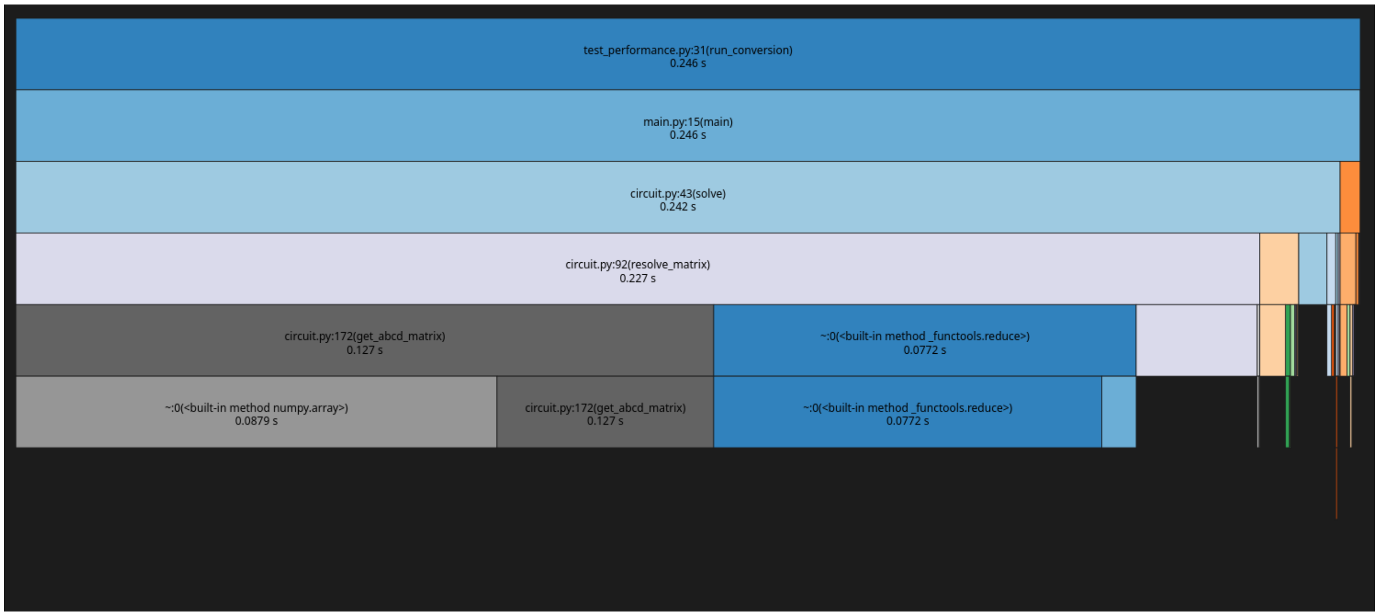


Fig. 1. Icicle graph of the cProfile run for the `Ext_e_Ladder_400.net` test case.

[2] `Ext_mdB_a_Test_Circuit_1.prof`

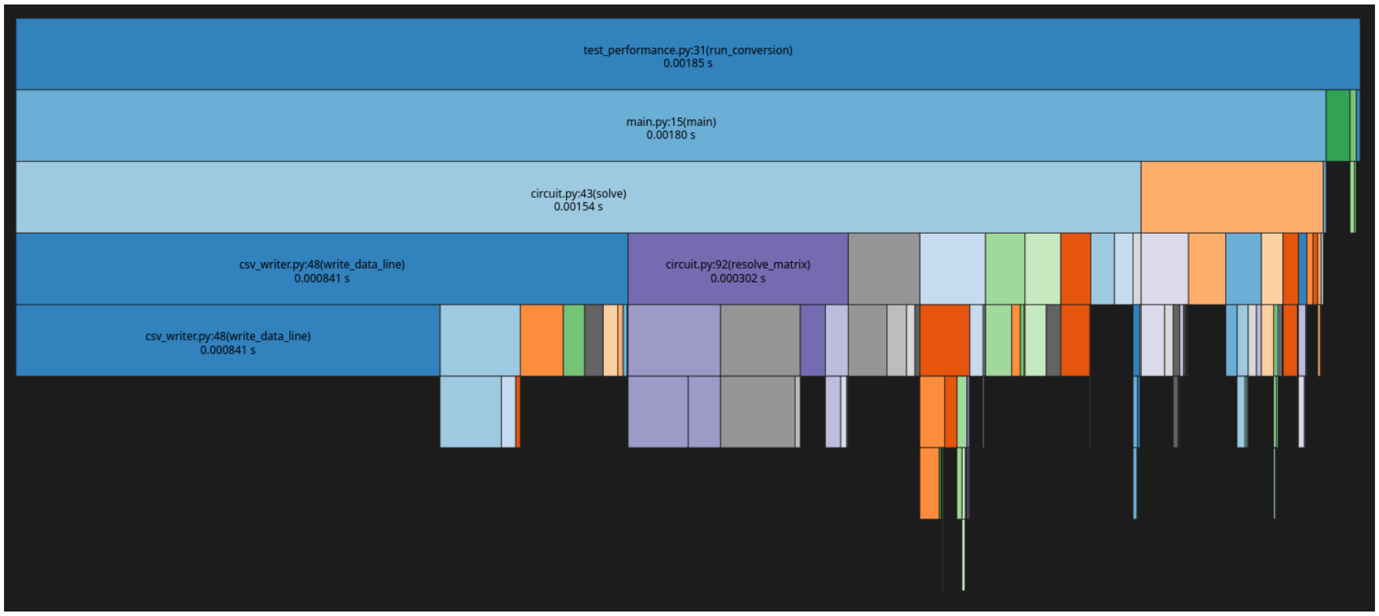
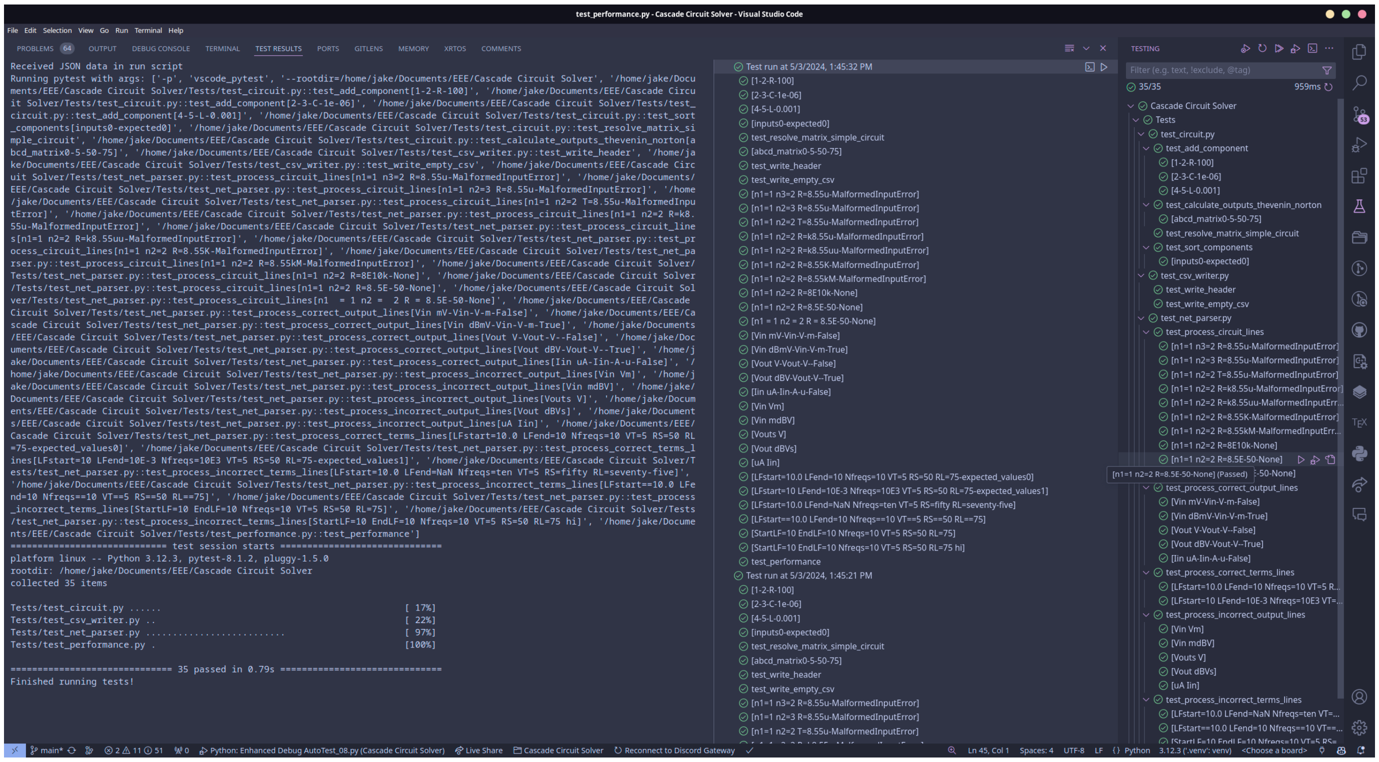


Fig. 2. Icicle graph of the cProfile run for the `Ext_mdB_a_Test_Circuit_1.net` test case.

[3] `Record of Testing`



```
Received JSON data in run script
Running pytest with args: ['-p', 'vscode_pytest', '--rootdir=/home/jake/Documents/EEE/Cascade Circuit Solver', '/home/jake/Docu
ments/EEE/Cascade Circuit Solver/Tests/test_circuit.py::test_add_component[1-2-R-100]', '/home/jake/Documents/EEE/Cascade Circu
it Solver/Tests/test_circuit.py::test_add_component[2-3-C-1e-06]', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_
circuit.py::test_add_component[4-5-L-0.001]', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_circuit.py::test_sort
_components[inputs0-expected0]', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_circuit.py::test_resolve_matrix_si
mple_circuit', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_circuit.py::test_calculate_outputs_thevenin_norton[a
bcd_matrix0-5-50-75]', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_csv_writer.py::test_write_header', '/home/ja
ke/Documents/EEE/Cascade Circuit Solver/Tests/test_csv_writer.py::test_write_empty_csv', '/home/jake/Documents/EEE/Cascade Circ
uit Solver/Tests/test_net_parser.py::test_process_circuit_lines[n1=1 n3=2 R=8.55u-MalformedInputError]', '/home/jake/Documents/
EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_circuit_lines[n1=1 n2=3 R=8.55u-MalformedInputError]', '/home
/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_circuit_lines[n1=1 n2=2 T=8.55u-MalformedInpu
tError]', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_circuit_lines[n1=1 n2=2 R=k8.
55u-MalformedInputError]', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_circuit_line
s[n1=1 n2=2 R=k8.55uu-MalformedInputError]', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_pr
ocess_circuit_lines[n1=1 n2=2 R=8.55K-MalformedInputError]', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_net_pa
rser.py::test_process_circuit_lines[n1=1 n2=2 R=8.55kM-MalformedInputError]', '/home/jake/Documents/EEE/Cascade Circuit Solver/
Tests/test_net_parser.py::test_process_circuit_lines[n1=1 n2=2 R=8E10k-None]', '/home/jake/Documents/EEE/Cascade Circuit Solver
/Tests/test_net_parser.py::test_process_circuit_lines[n1=1 n2=2 R=8.5E-50-None]', '/home/jake/Documents/EEE/Cascade Circuit Sol
ver/Tests/test_net_parser.py::test_process_circuit_lines[n1  = 1 n2 =  2 R = 8.5E-50-None]', '/home/jake/Documents/EEE/Cascade 
Circuit Solver/Tests/test_net_parser.py::test_process_correct_output_lines[Vin mV-Vin-V-m-False]', '/home/jake/Documents/EEE/Ca
scade Circuit Solver/Tests/test_net_parser.py::test_process_correct_output_lines[Vin dBmV-Vin-V-m-True]', '/home/jake/Documents
/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_correct_output_lines[Vout V-Vout-V--False]', '/home/jake/Doc
uments/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_correct_output_lines[Vout dBV-Vout-V--True]', '/home/j
ake/Documents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_correct_output_lines[Iin uA-Iin-A-u-False]', '/
home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_incorrect_output_lines[Vin Vm]', '/home/j
ake/Documents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_incorrect_output_lines[Vin mdBV]', '/home/jake/
Documents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_incorrect_output_lines[Vouts V]', '/home/jake/Docum
ents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_incorrect_output_lines[Vout dBVs]', '/home/jake/Document
s/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_incorrect_output_lines[uA Iin]', '/home/jake/Documents/EEE/
Cascade Circuit Solver/Tests/test_net_parser.py::test_process_correct_terms_lines[LFstart=10.0 LFend=10 Nfreqs=10 VT=5 RS=50 RL
=75-expected_values0]', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_correct_terms_l
ines[LFstart=10 LFend=10E-3 Nfreqs=10E3 VT=5 RS=50 RL=75-expected_values1]', '/home/jake/Documents/EEE/Cascade Circuit Solver/T
ests/test_net_parser.py::test_process_incorrect_terms_lines[LFstart=10.0 LFend=NaN Nfreqs=ten VT=5 RS=fifty RL=seventy-five]',
'/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process_incorrect_terms_lines[LFstart==10.0 LFe
nd=10 Nfreqs==10 VT==5 RS==50 RL==75]', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests/test_net_parser.py::test_process
_incorrect_terms_lines[StartLF=10 EndLF=10 Nfreqs=10 VT=5 RS=50 RL=75]', '/home/jake/Documents/EEE/Cascade Circuit Solver/Tests
/test_net_parser.py::test_process_incorrect_terms_lines[StartLF=10 EndLF=10 Nfreqs=10 VT=5 RS=50 RL=75 hi]', '/home/jake/Docume
nts/EEE/Cascade Circuit Solver/Tests/test_performance.py::test_performance']
============================= test session starts ==============================
platform linux -- Python 3.12.3, pytest-8.1.2, pluggy-1.5.0
rootdir: /home/jake/Documents/EEE/Cascade Circuit Solver
collected 35 items

Tests/test_circuit.py ......                                             [ 17%]
Tests/test_csv_writer.py ..                                              [ 22%]
Tests/test_net_parser.py ..........................                      [ 97%]
Tests/test_performance.py .                                              [100%]

============================== 35 passed in 0.79s ==============================
Finished running tests!
```

```
Test run at 5/3/2024, 1:45:32 PM
  [1-2-R-100]
  [2-3-C-1e-06]
  [4-5-L-0.001]
  [inputs0-expected0]
  test_resolve_matrix_simple_circuit
  [abcd_matrix0-5-50-75]
  test_write_header
  test_write_empty_csv
  [n1=1 n3=2 R=8.55u-MalformedInputError]
  [n1=1 n2=3 R=8.55u-MalformedInputError]
  [n1=1 n2=2 T=8.55u-MalformedInputError]
  [n1=1 n2=2 R=k8.55u-MalformedInputError]
  [n1=1 n2=2 R=k8.55uu-MalformedInputError]
  [n1=1 n2=2 R=8.55K-MalformedInputError]
  [n1=1 n2=2 R=8.55kM-MalformedInputError]
  [n1=1 n2=2 R=8E10k-None]
  [n1=1 n2=2 R=8.5E-50-None]
  [n1 = 1 n2 = 2 R = 8.5E-50-None]
  [Vin mV-Vin-V-m-False]
  [Vin dBmV-Vin-V-m-True]
  [Vout V-Vout-V--False]
  [Vout dBV-Vout-V--True]
  [Iin uA-Iin-A-u-False]
  [Vin Vm]
  [Vin mdBV]
  [Vouts V]
  [Vout dBVs]
  [uA Iin]
  [LFstart=10.0 LFend=10 Nfreqs=10 VT=5 RS=50 RL=75-expected_values0]
  [LFstart=10 LFend=10E-3 Nfreqs=10E3 VT=5 RS=50 RL=75-expected_values1]
  [LFstart=10.0 LFend=NaN Nfreqs=ten VT=5 RS=fifty RL=seventy-five]
  [LFstart==10.0 LFend=10 Nfreqs==10 VT==5 RS==50 RL==75]
  [StartLF=10 EndLF=10 Nfreqs=10 VT=5 RS=50 RL=75]
  [StartLF=10 EndLF=10 Nfreqs=10 VT=5 RS=50 RL=75 hi]
  test_performance
Test run at 5/3/2024, 1:45:21 PM
  [1-2-R-100]
  [2-3-C-1e-06]
  [4-5-L-0.001]
  [inputs0-expected0]
  test_resolve_matrix_simple_circuit
  [abcd_matrix0-5-50-75]
  test_write_header
  test_write_empty_csv
  [n1=1 n3=2 R=8.55u-MalformedInputError]
  [n1=1 n2=3 R=8.55u-MalformedInputError]
  [n1=1 n2=2 T=8.55u-MalformedInputError]
```

```
35/35                                                959ms
Cascade Circuit Solver
  Tests
    test_circuit.py
      test_add_component
        [1-2-R-100]
        [2-3-C-1e-06]
        [4-5-L-0.001]
      test_calculate_outputs_thevenin_norton
        [abcd_matrix0-5-50-75]
      test_resolve_matrix_simple_circuit
      test_sort_components
        [inputs0-expected0]
    test_csv_writer.py
      test_write_header
      test_write_empty_csv
    test_net_parser.py
      test_process_circuit_lines
        [n1=1 n3=2 R=8.55u-MalformedInputError]
        [n1=1 n2=3 R=8.55u-MalformedInputError]
        [n1=1 n2=2 T=8.55u-MalformedInputError]
        [n1=1 n2=2 R=k8.55u-MalformedInputError]
        [n1=1 n2=2 R=k8.55uu-MalformedInputErr...
        [n1=1 n2=2 R=8.55K-MalformedInputError]
        [n1=1 n2=2 R=8.55kM-MalformedInputErr...
        [n1=1 n2=2 R=8E10k-None]
        [n1=1 n2=2 R=8.5E-50-None]
[n1=1 n2=2 R=8.5E-50-None] (Passed)    -50-None]
      test_process_correct_output_lines
        [Vin mV-Vin-V-m-False]
        [Vin dBmV-Vin-V-m-True]
        [Vout V-Vout-V--False]
        [Vout dBV-Vout-V--True]
        [Iin uA-Iin-A-u-False]
      test_process_correct_terms_lines
        [LFstart=10.0 LFend=10 Nfreqs=10 VT=5 R...
        [LFstart=10 LFend=10E-3 Nfreqs=10E3 VT=...
      test_process_incorrect_output_lines
        [Vin Vm]
        [Vin mdBV]
        [Vouts V]
        [Vout dBVs]
        [uA Iin]
      test_process_incorrect_terms_lines
        [LFstart=10.0 LFend=NaN Nfreqs=ten VT=...
        [LFstart==10.0 LFend=10 Nfreqs==10 VT==...
```

Fig. 3. Screenshot from VSCode testing window, showing all tests passing.