Comprehensive Analysis Report: Circuit Simulator

Arjun Verma
Subhashree Sahoo

November 25, 2024

Abstract

This document provides a comprehensive technical analysis of the Circuit Simulator project. The simulator, implemented in Python, supports both time and frequency domain analyses of RLC circuits. This report delves into the class structure, methods, algorithms, and user interface, alongside detailed explanations of the mathematical and computational foundations.

1 Introduction

The Circuit Simulator is an interactive Python-based tool designed for analyzing electrical circuits. It leverages symbolic computation for solving circuit equations using Kirchhoff's Current Law (KCL) and Laplace transforms. Key features include graphical visualization, time-domain analysis, and frequency-domain analysis, implemented through an intuitive Streamlit interface.

2 Core Components Analysis

2.1 Class: `CircuitSimulator`

The `CircuitSimulator` class is the computational core of the system, responsible for managing components, equations, and solutions. Its attributes and methods are described below:

2.1.1 Attributes

- **components**: A list storing all the circuit components.
- **nodes**: A set containing unique nodes in the circuit.
- **equations**: A list of symbolic equations representing the circuit's behavior.
- **node_vars**: A dictionary mapping nodes to symbolic variables for voltage.
- **current_vars**: A dictionary mapping certain components to symbolic current variables.

- **solutions**: A dictionary storing the solved variables (voltages and currents).
- **s**: The Laplace domain variable used for frequency-domain analysis.
- **t**: The time domain variable for inverse Laplace transformations.
- **component_count**: Tracks the number of each type of component added to ensure unique identifiers.

2.1.2 Methods

`add_component(name, value, n1, n2, ac_params=None)`

- Adds a new circuit component, including validation of values and nodes.
- Tracks components with unique IDs and supports optional AC parameters for voltage sources.
- Updates the `components` list and the `nodes` set.

`setup_node_variables()`

- Creates symbolic variables for node voltages.
- Initializes current variables for inductors and voltage sources.
- Essential for symbolic circuit analysis.

`build_equations()`

- Constructs circuit equations based on Kirchhoff's Current Law (KCL).
- Handles both resistive and reactive components, such as capacitors and inductors.
- Supports AC voltage sources in the frequency domain using Laplace transforms.
- Includes reference node voltage as a constraint.

`solve_circuit()`

- Solves the constructed equations symbolically using SymPy.
- Returns solutions for node voltages and branch currents in the Laplace domain.
- Handles cases where no solution is found with appropriate error messages.

`get_time_domain_response(tmax, points)`

- Computes the time-domain responses of the circuit variables using inverse Laplace transform.
- Handles impulse responses (DiracDelta) and smoothens transitions.
- Implements amplitude limiting to avoid numerical instability.

`get_frequency_response(fmin, fmax, points)`

- Analyzes the frequency response of the circuit variables.
- Calculates magnitude and phase characteristics over a logarithmic frequency range.

`create_circuit_visualization(components, nodes)`

- Generates a professional visualization of the circuit using NetworkX and Matplotlib.
- Annotates components and nodes with labels.
- Highlights AC sources distinctly.

3 Mathematical and Computational Foundations

3.1 Kirchhoff's Current Law (KCL)

- Uses KCL to express the sum of currents entering and leaving a node.
- Forms the basis of the circuit's symbolic equations.

3.2 Laplace Transforms

- Converts time-domain differential equations to algebraic equations in the frequency domain.
- Handles reactive components (capacitors and inductors) efficiently.

3.3 Numerical and Symbolic Computation

- SymPy is used for symbolic computations, including solving equations and Laplace transforms.
- NumPy handles numerical evaluations for time-domain and frequency-domain responses.

4 User Interface Implementation

The Streamlit-based interface provides an intuitive platform for users to interact with the simulator:

- **Component Addition**: Allows users to specify component type, value, and nodes.
- **Circuit Visualization**: Displays the circuit diagram dynamically as components are added.
- **Time-Domain Analysis**: Plots voltage and current responses over time.
- **Frequency-Domain Analysis**: Displays magnitude and phase plots across a frequency range.
- **Error Handling**: Provides feedback for invalid inputs or unsolvable circuits.

5 Flow of Control

1. User interacts with the Streamlit interface to add components.
2. The simulator validates inputs and updates the circuit configuration.
3. Equations are built and solved when the user requests analysis.
4. Results are computed and displayed in graphical formats.
5. Users can iterate by modifying components or analyzing different aspects.

6 Visualization and Output

6.1 Circuit Diagram

- Represents the circuit as a graph with nodes and labeled edges.
- Visualizes AC sources and component connections distinctly.

6.2 Response Plots

- **Time Domain**: Voltage and current variations over time.
- **Frequency Domain**: Magnitude and phase characteristics across frequencies.

7 Visualization and Output

7.1 Circuit Simulation Outputs

8 Error Handling and Robustness

- Ensures all components have valid values and connections.
- Handles edge cases, such as disconnected nodes or missing reference nodes.
- Provides detailed error messages for invalid configurations.

9 Conclusion

The Circuit Simulator is a robust and versatile tool for analyzing RLC circuits. Its integration of symbolic and numerical methods provides a comprehensive platform for both educational and practical applications. Future work could extend the simulator to support additional component types and more complex network analysis.

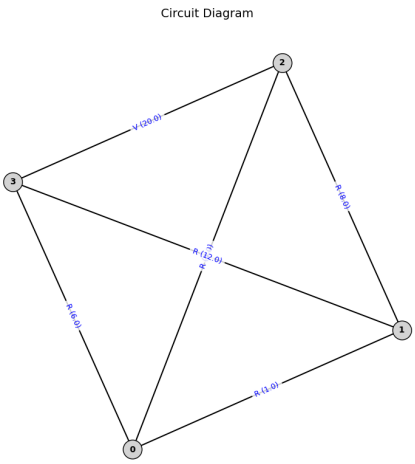

Figure 1: (a) Circuit Diagram of Wheatstone Bridge

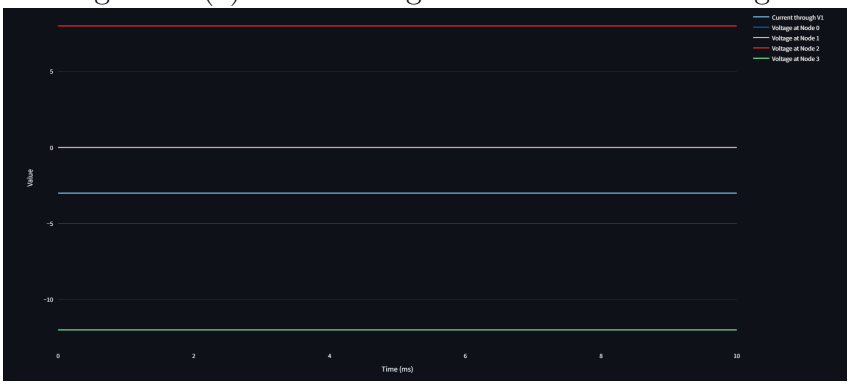

Figure 2: (b) Time Domain Response

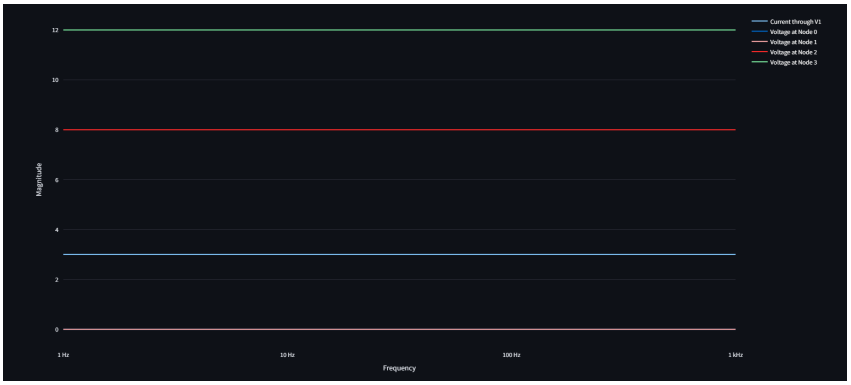

Figure 3: (c) Frequency vs Magnitude

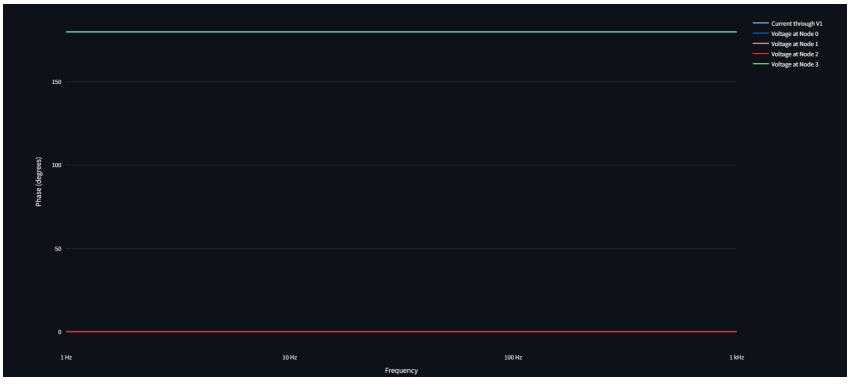

Figure 4: (d) Phase vs Frequency

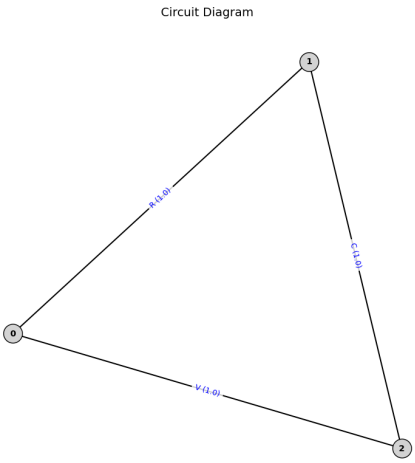

Figure 5: (a) Circuit Diagram of High pass RC circuit

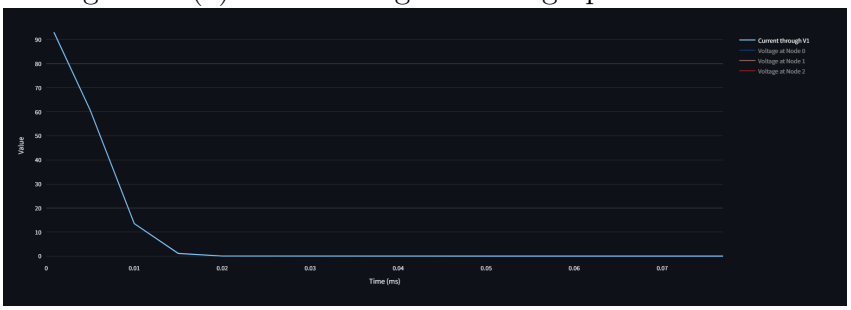

Figure 6: (b) Time Domain Response

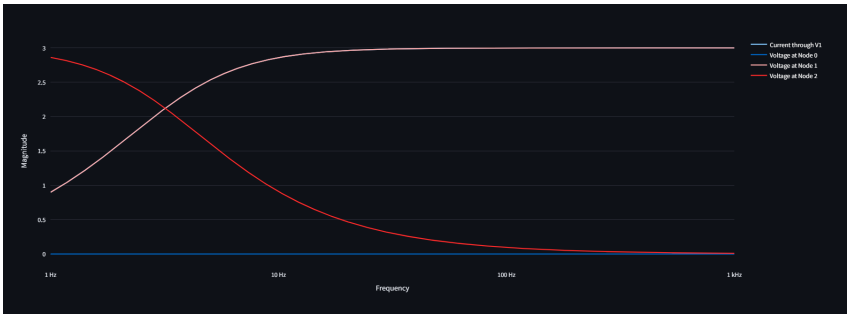

Figure 7: (c) Frequency vs Magnitude

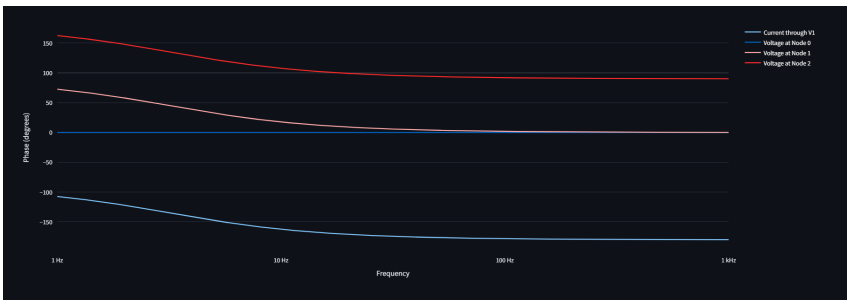

Figure 8: (d) Phase vs Frequency

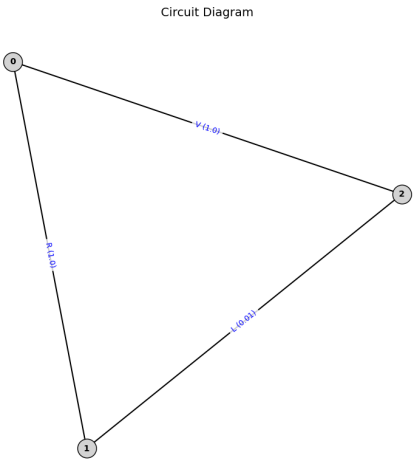

Figure 9: (a) Circuit Diagram of Low pass RL circuit

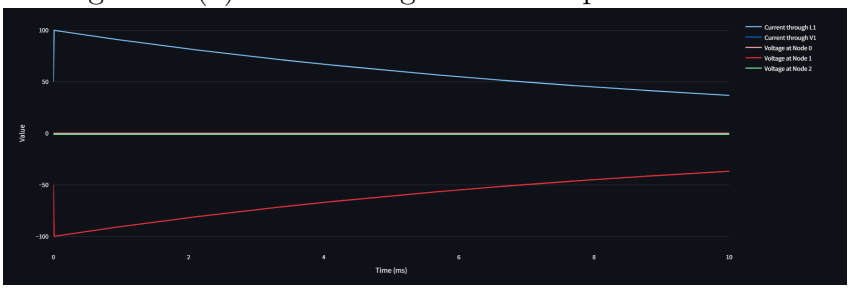

Figure 10: (b) Time Domain Response

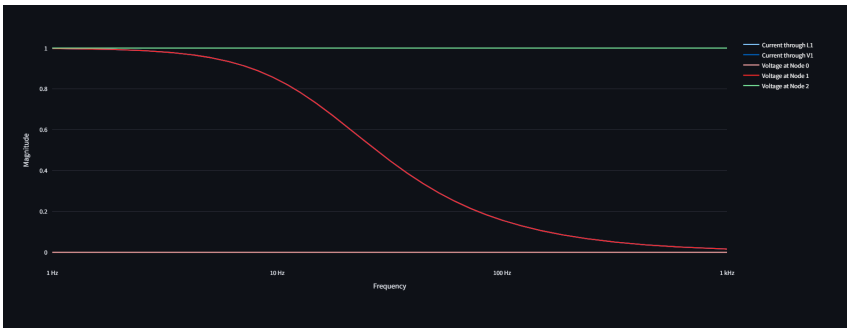

Figure 11: (c) Frequency vs Magnitude

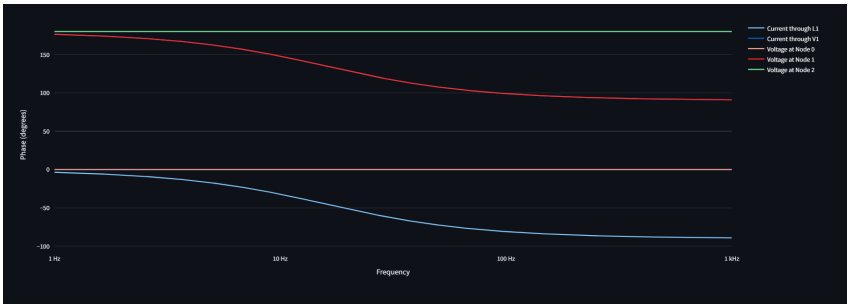

Figure 12: (d) Phase vs Frequency

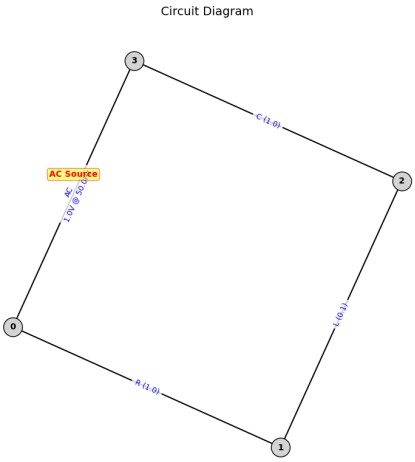

Figure 13: (a) Circuit Diagram of RLC circuit with AC source

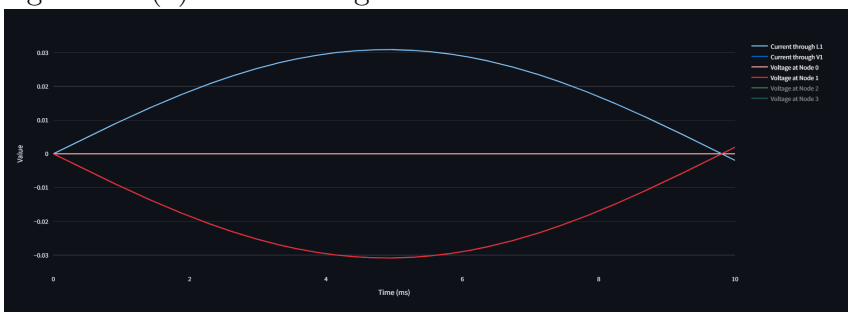

Figure 14: (b) Time Domain Response

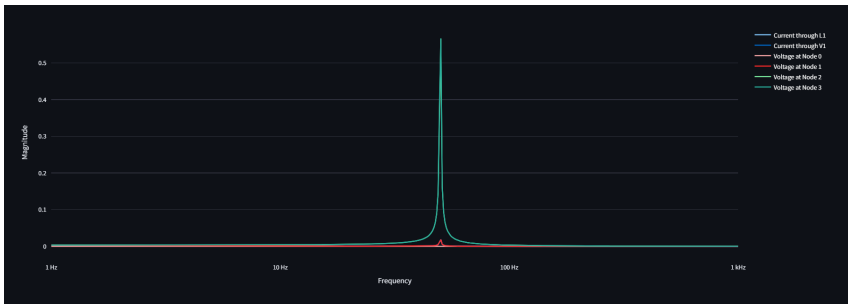

Figure 15: (c) Frequency vs Magnitude

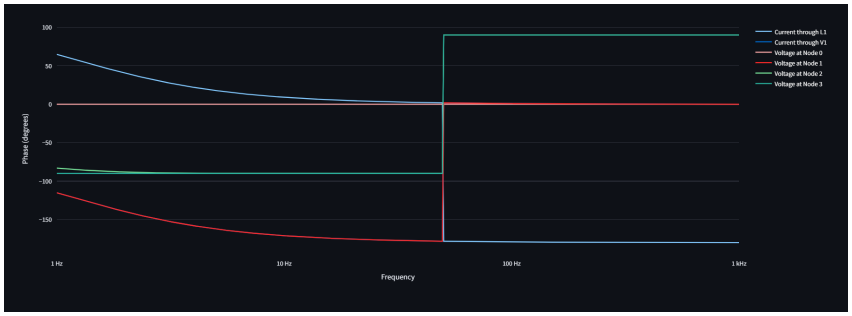

Figure 16: (d) Phase vs Frequency

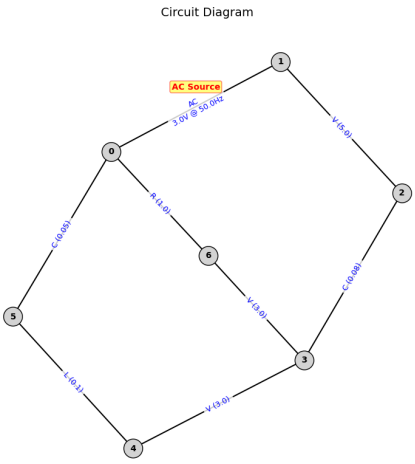

Figure 17: (a) Circuit Diagram of RLC circuit with AC source

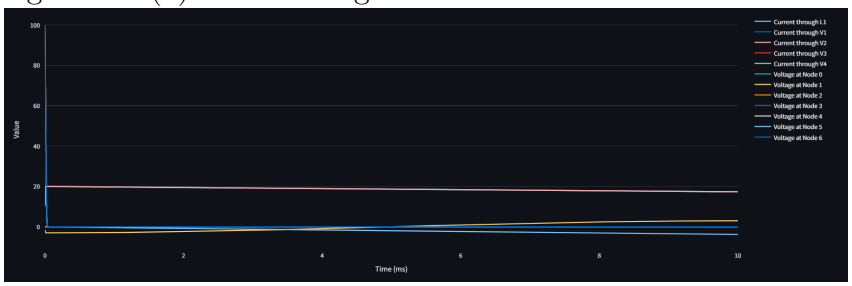

Figure 18: (b) Time Domain Response

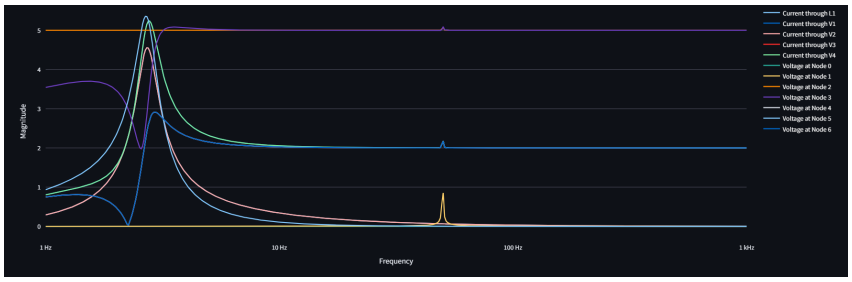

Figure 19: (c) Frequency vs Magnitude

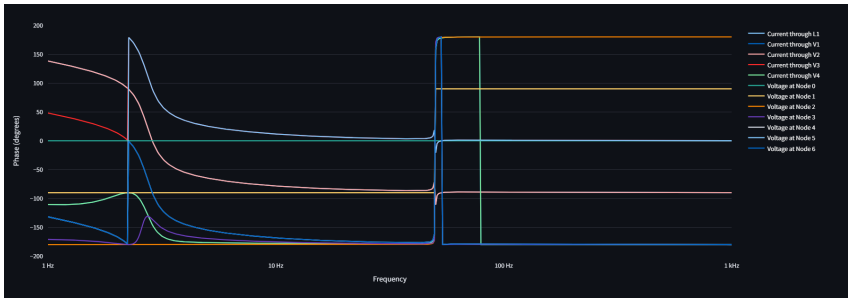

Figure 20: (d) Phase vs Frequency