AC Circuit Builder

Mohammad Kordzanganeh[10137275]

The School of Physics and Astronomy
The University of Manchester

Abstract. The AC circuits have many applications in our everyday lives and a thorough understanding of them is crucial for every physicist. These circuits consist of three types of components: resistors, inductors, and capacitors, and they can be assembled in series or in parallel. This program provides an interface to create these circuits and calculate its overall impedance. It can also load a saved circuit as well as providing a graphical representation of it in HTML. This was made possible by linearising the circuit so it could be stored in a one-dimensional array, and then converting this array into an SVG diagram.

1 Introduction

1.1 Background

The AC circuits are an important part of the everyday life. There can be 3 types of components in AC circuits: resistor, inductor, and capacitor. Often a power supply is also included, but in this work we shall neglect its use. A combination of these components in parallel and series in a closed loop can induce a periodic current in the circuit.

1.2 Motivation

The motivation of this work is to create a C++ program to calculate the overall impedance of a circuit and find the contributions of all constituent components. This work extends this requirement by adding two extra functionalities: 1) saving and loading circuits and 2) outputting a graphical representation of the circuit. The former is achieved using a linearising language which is discussed in more detail in Section 4.2, while the latter uses scalable vector graphics (SVG) to render a circuit.

2 Theory recap

2.1 Relation between voltage and current

Each component provides a unique relationship between the voltage (V) across it and the current (I) through it. Perhaps most familiar is the resistor and how it relates voltage using its resistance

$$V = IR, \tag{1}$$

where R is the resistance of the resistor. An inductor behaves slightly differently

$$V = -L\dot{I}, \tag{2}$$

where L is the inductance and the $\dot{I}$ represents the first time-derivative of the current. Finally, for a capacitor we have

$$V = \frac{1}{C}\int I dt, \tag{3}$$

where C is its capacitance.

2.2 Kirchhoff's laws

To understand a circuit, we shall begin from Kirchhoff's laws to find the relation between the voltage and the current of the entire circuit:

- Current law: the current flowing into a junction must flow out of it.
- Voltage law: the sum of voltages in a closed loop should be zero.

Using the current law, we can determine that in a series circuit, the overall resistance and inductances are additive but capacitance is inverted

$$R_{tot} = R_1 + R_2 + \ldots, L_{tot} = L_1 + L_2 + \ldots, \frac{1}{C_{tot}} = \frac{1}{C_1} + \frac{1}{C_2} + \ldots, \tag{4}$$

and vice versa for the parallel circuits.

From the voltage law, we conclude that

$$-L\ddot{I} + R\dot{I} + \frac{1}{C}I = 0. \tag{5}$$

This equation describes a damped harmonic motion with a natural frequency

$$\omega = \frac{1}{\sqrt{LC}}. \tag{6}$$

2.3 Impedance

The harmonic solution means that the relationship of the current with time is sinusoidal. This means that in all three types of components, the voltages are also sinusoidal.

Let us define two new complex variables $\tilde{I}$ and $\tilde{V}$, which extrapolate the real values of the current and voltage to complex exponentials

$$\tilde{I} = I_{max}exp(i\omega t + \phi), \tilde{V} = V_{max}exp(i\omega t), \tag{7}$$

where ϕ is the phase difference between the two. This means that the complex current of every component now has a linear relationship with the complex voltage across it

$$\tilde{V} = Z\tilde{I},$$

where Z is known as the impedance.

3 Functionality

3.1 Creating a circuit

The user is asked to specify the circuit in a linear manner. At every point, the User can either enter one of the three circuit components (resistor, inductor, or capacitor), add a junction, or finish the series wire.

Adding a component: The user can add a circuit component by entering the number associated with the desired component followed by the value that they have in mind.

Adding a junction: Constructing parallel circuit can be an involved task when it is not done graphically. In this program, this problem is tackled by asking the user to input junctions. When the user chooses to input a junction, she is asked about the number of wires that leave this junction. Then, the user is faced with the initial choices again. There is no limit on the number of nested junctions that can be added, and so it is possible to have nested parallel circuits.

Finishing a wire: It is necessary for every wire to have an end point. For example, in a parallel circuit, it is necessary to know when the user would like to finish adding components to the first wire and start adding to the second one. This can be achieved by signalling when a wire is a finished. In a circuit with two parallel wires,the user can add to the first wire. By signalling finish, she can move to the next wire, and by signalling again she leaves the parallel wires and can add components to the rest of the circuit. In this case, choosing finish would conclude the circuit.

3.2 Save and load

The program can save an input circuit in a text file using a method outlined in Section 4.2. The output file can then be loaded by the program to re-generate the circuit in a future run. This output file can be altered easily if one follows the description in Section 4.2.

3.3 Graphical output

The user can choose to output a graphical diagram of the circuit into an HTML file. The graphic is made using a generated SVG code embedded into this HTML file and its primary function is to inform the user whether the diagram they have created is correct.

4 Implementation

4.1 Class structure

A base class, *circuit_part*, encompasses all elements of a circuit. Two classes of *component* and *wire* are derived from it. It holds 3 double variables: resistance, inductance, and capacitance, and their respective inputs are given in the units of Ohms, milli Henries and nano Farads. It also has a string value that signifies the type of the object, which the derived objects assign a value to when they are initialised.

The component class has three children, *resistor*, *capacitor*, and *inductor*. They overload a component member function to calculate their impedance based on a static frequency variable that is updated by the overall circuit.

The wire class has two children, *series_wire* and *parallel_wire*. The wire class has a member *circ_members* which is a vector of base class pointers to circuit_part. This enables every wire to include both wires and components.

The wire class has a virtual member function, *update_members()*, which was overloaded in both derived wire classes to accommodate for the different addition behaviour of components. This function is called whenever a new part is added to a wire to re-calculate the total resistance, inductance and capacitance.

Figure 1[1] shows this class structure in more detail.

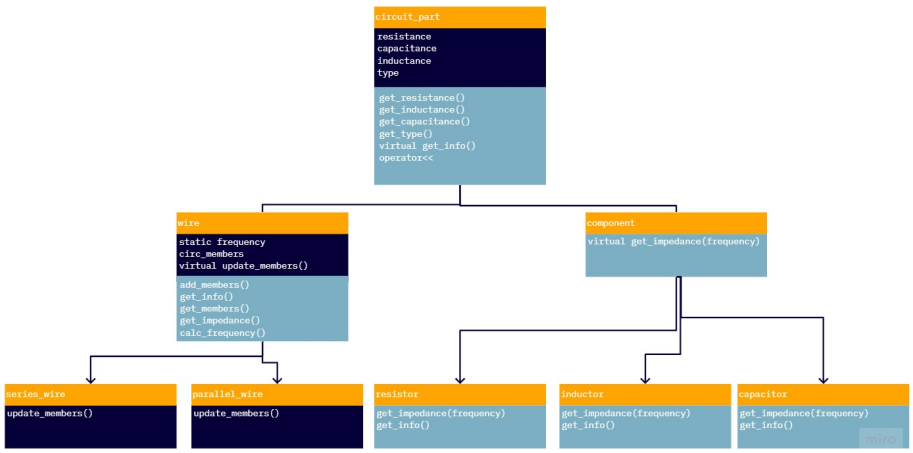

Fig. 1. UML class diagram for the circuit_part base class. Only the series_wire has the frequency and impedance functions as a parallel_wire object cannot make a full circuit on its own.

[1] This diagram was made in Miro.

4.2 Linearising an AC circuit

Describing a circuit in series is simple: all that is needed is a sequential approach to creating it. For example, a circuit with a 100 Ohm resistor followed by a 50 Ohm resistor can be described as: R100 then R50 then finish. However, to make a circuit, it is necessary to find a way to input a parallel circuit, too.

An example of a more complicated circuit could be the above circuit followed by a parallel circuit where the first wire includes a 10 Ohm resistor and a 5 Ohm resistor, and the second wire includes a 20 Ohm resistor. To make the example more general we can follow the parallel circuit by a 15 Ohm resistor. We can use junctions to read this: R100 then R50 then junction of 2 wires then R10 then R5 then finish wire then R15 then finish wire then R15 then finish.

This work uses a string to read such circuits. The above example would be written: `R100R50J2R10R5FR20FR15F`, where F signifies the end of the wire. The graphic for this circuit is draw in Figure 2, and its results are shown in Figure 3. This string is generated whenever the user creates a circuit. The user can

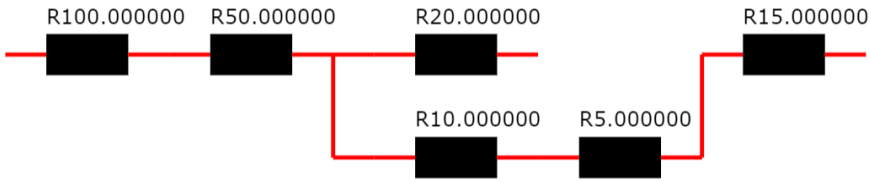

Fig. 2. Graphical representation of the circuit in the example. As evident, there are two major areas of improvement. Firstly, when the parallel wires posses an unequal number of components, the lines do not cover the entire horizontal space and there seems to be a disconnect. Secondly, the numbers have unnecessarily high precisions. This is to account for the variable units of inductance and capacitance, but it also shows that this section could be improved.

then choose to save the circuit, which saves this string to a text file. In future use, the user can choose to load the circuit by reading this text file. The loading function works similarly to the user input function, but instead of taking input from *cin* it transforms the text file into a stringstream and passes it to the input function. The saved text file can be manually altered to enter new diagrams, but to work, it needs to have the correct[2] number of F's.

4.3 SVG diagrams

To make the SVG diagrams, this work uses a combination of rectangles (for components) and lines (for wires). The former requires a beginning coordinate (x_0, y_0), a width and a height, whereas the latter requires beginning and ending

[2] All the opened wires (including the starting branch) must be closed with F's.

```
------------=Welcome to the A.C circuit builder=---------------
Please choose from the below options on how you would like to proceed:
1)Create new circuit
2)Load saved circuit
2
Loaded!
Would you like to see the information for this circuit?(y/n)
y
Circuit:
Total Resistance: 174 Ohms      Total Capacitance: 0 nano Farads        Total Inductance: 0 milli Henries
Impedance:      Resistance = 174         Reactance = 0   ---     Amplitude = 174         Phase = 0
List of Components:
Resistor of resistance: 100 Ohms         Impedance : (100,0)
Resistor of resistance: 50 Ohms  Impedance : (50,0)
Resistor of resistance: 10 Ohms  Impedance : (10,0)
Resistor of resistance: 5 Ohms   Impedance : (5,0)
Resistor of resistance: 20 Ohms  Impedance : (20,0)
Resistor of resistance: 15 Ohms  Impedance : (15,0)

Would you like to draw the circuit?(y/n)
```

Fig. 3. The result of the example circuit in the console. The impedances are shown using the complex¡double¿ class from the standard library.

coordinates. Coordinates are held in the class *coords*, and the base class *svg* is the parent class for the svg elements in the code. The latter has a virtual member function *get_svg_code()* to generate the code for the svg objects. This class has 4 derived classes:

- *horizontal_line* is a horizontal line with a fixed length at a given starting coordinate.
- *box* is a rectangle with fixed width and height at the given coordinate.
- *vertical_line* is a vertical line with an input length at a given coordinate.
- *junction* is a collection of equally spaced horizontal lines with a vertical line connecting their beginnings. It takes as input the number of outgoing wires.

5 Discussions and future work

5.1 Discussion

Choice of circuit_part base class: One of the earliest choices in the class design was whether to separate the wires class from the components class. This route was explored but, depending on the implementation, it was either inefficient or unsuccessful. The following paths were taken to explore this route:

1. In the wires class, use an array of components instead of array of parts. This meant that the parallel wires could only hold on to one component, which was sub-optimal.
2. Use an array of wires, but add a third wire class that holds components. This was highly inefficient, as it would require the user to create a wire and then append the wire to the circuit.
3. Add a template of components or wires for the class. This solved the problem of adding a third class of wire and the parallel wires, but was still inefficient as the user needed to append wires to the circuit as she progressed.

Choice of SVG as a graphical output: Many avenues were explored for a graphical output:

1. The C++ console was incompatible with the way the project was done as it needed to have the foresight of what came next. It could be an excellent extension to adapt the project to redraw the circuit in the console with every user input, but it would be larger than the scope of this project.
2. The graphics.h library was briefly investigated, but as it is not a standard library, it was not deployed.

5.2 Future improvements

Better graphical representation: The graphical representation in this project focussed primarily on conveying to the user the type of circuit that she had built using minimal graphical resources. Although this project achieves this, ideally this representation would also include completed parallel lines, as well as specific shapes for each component.

Graphical user interface: Circuits, and especially their parallel wires, are much easier to input graphically than linearly. The creation of a graphical user interface (GUI) was beyond the scope of this project but it could be an appropriate future addition to the project.

5.3 Exception handling:

The current exception handling of the project deals with most of the issues that can arise from poor input into the program by asking the user to start anew. This is far from ideal, and renders manually inputting longer projects impractical. Implementing this as well as a GUI, could make it very easy for a user to interact with this application.