AC Circuit in C++

Lili Kovacs
10735793

PHYS30762 Object-Oriented Programming in C++

Department of Physics and Astronomy
The University of Manchester

Abstract

The aim of this project was to create a program in C++ that allows the user to create an analogue circuit by adding arbitrary components. The program is then able to calculate its impedance (with magnitude and phase information) and display circuit and component information.

The source code includes an abstract base class for components and a circuit class, through which the operations are performed.

Initial properties of the circuit, given by the user determine the range of available actions for that circuit: more options are available for simple circuits where components are all in series or parallel, while for more compound ones, only the main functionalities of the program can be carried out.

1 Introduction

1.1 Impedance

Impedance, Z is a complex number, representing the ratio of voltage to current across an element, part or the entirety of an alternating current (AC) circuit. For components in series, the impedance is just the sum of the impedances of the individual components:

$$Z_{total} = \sum_i Z_i, \tag{1}$$

while for components in parallel, the reciprocal of the total impedance is the sum of the reciprocals of the individual components' impedances:

$$\frac{1}{Z_{total}} = \sum_i \frac{1}{Z_i}. \tag{2}$$

The impedance of each circuit element involved in this project can be calculated using the following formulas:

$$Z_R = R, Z_C = -\frac{i}{C\omega}, Z_L = iL\omega, \tag{3}$$

where R is resistance, C is capacitance, L is inductance and ω is the angular frequency of the circuit and the subscripts stand for resistor, capacitor and inductor respectively [1]. As for any complex number, the magnitude of Z can be calculated as $|Z| = \sqrt{Z(re)^2 + Z(im)^2}$ while the phase angle is $\theta = \arctan \frac{Z(im)}{Z(re)}$. These can be calculated for individual components and the whole circuit alike.

1.2 Project overview

The aim of this project was to create a C++ program for the user to be able to construct an arbitrary AC circuit from a range of components and then calculate the impedance, magnitude of impedance and phase of the whole circuit and its components individually. Threw choices for components are available for the user to choose from: resistor, capacitor and inductor.

Upon starting the program, the user is asked to choose what type of circuit they want to build and to set the frequency of the circuit. A menu will come up, with its range of options decided by the type of circuit the user has decided on. From here, the user is be able to choose what to do next. Once an action has been carried out, the user is asked again to enter their choice for the next one. The menu for an all-series or all-parallel circuit can be seen of Figure 1.

For any circuit, the user can decide to add a new component, display circuit or component information (impedance, magnitude of impedance and phase angle) or quit the program at any

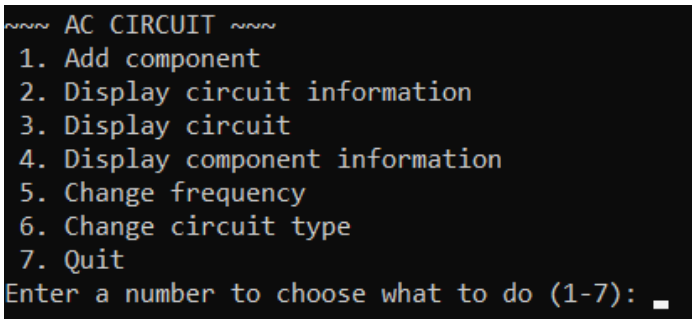

Figure 1: Starter menu for a simple circuit and the message asking the user to choose their next action.

point. For the more compound circuits, when the user decides to add a new component, they are asked how they want to connect it to the existing part of the circuit. For simple circuits (all-series or all-parallel circuits), there are also options to display a schematic diagram of the circuit, change the initial frequency set upon starting the program or change the circuit type.

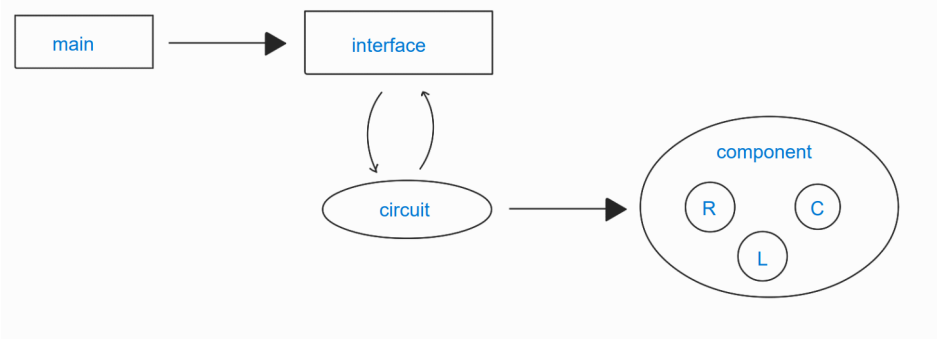

Figure 2: Flowchart summarising the interactions between the files. Main calls functions from interface, which uses member functions of the circuit class. Circuit components are created using the component class.

2 Code design and implementation

The code is split into main.cpp, interface .h and .cpp, circuit .h and .cpp, component .h (as abstract classes do not have any function definitions) and three .h and .cpp files, one for resistor, capacitor and inductor each.

main.cpp is responsible for the program running, but most processes are defined in the interface files, using the class members of the circuit and component classes.

A schematic diagram showing the relationships of these files can be seen on Figure 2.

2.1 Classes

2.1.1 Circuit

An object of the circuit class is characterised by its structure (defined by the user upon staring the program), impedance and components. It is initialised by specifying its structure only. Components can be added though a member function and are stored as a vector of smartpointers to the individual components for easy addition and an efficient use of memory. Frequency is a static data member of the class, also set by the user when starting the program, therefore it is easy to use to set the frequency of components when creating them or when modifying the frequency of the circuit.

Apart from the member functions allowing to access circuit data from outside the class (or for simpler circuits, modify its type and the frequency of the circuit and its components), the circuit class only has one more member function: the template function on Figure 3 to add components.

```
template <class ctype> void add_component(double value)
{
    components.push_back(std::make_shared<ctype>(value));
    std::cout << "Component added!" << std::endl;
}
```

Figure 3: Template function in the circuit class to add components. A smartpointer to a "ctype" object will be added to the vector of components and "value" will take the place of resistance, capacitance or inductance in the constructor.

This function allows to add a new component of any class to the vector containing smartpointers to the existing circuit elements.

2.1.2 Components

```
class component // abstract base class for components
{
public:
    virtual ~component() {}
    virtual void set_component_frequency(double freq) = 0;
    virtual double get_component_frequency() const = 0;
    virtual std::complex<double> get_component_impedance() const = 0;
    virtual double get_component_magnitude() const = 0;
    virtual double get_component_phase() const = 0;
    virtual std::string component_type() const = 0;
    virtual std::string get_component_symbol() const = 0;
};
```

Figure 4: Abstract base class for components from which the resistor, capacitor and inductor classes are derived. Class definition contains virtual functions only, which are overridden for each component in their respective classes.

The three component types allowed in the circuit (resistor, capacitor and inductor) all have their own class, derived from and abstract base class for components, defined as on Figure 4.

The abstract class contains virtual get functions and a function to set the frequency of a component.

For each derived class, there are members for the name (type), symbol, frequency and the physical quantity describing the component: resistance for resistor, capacitance for capacitor and inductance for inductor. These are initialised when a component is added to the circuit.

2.2 Main and user interface

When starting the program, main.cpp calls the welcome() function from interface. The user decides what type of circuit they want and the circuit is created in main. The user is then asked to set the frequency of the circuit - this is important, as for compound circuits, the impedance value is updated as new components are added instead of it being calculated upon request of the user. (For simpler circuits, the user is able to change the frequency and the impedance will be recalculated.)

Interface defines two namespaces: one for simple circuits and one for more compound ones. Depending on the user's initial choice (which determines the namespace used), the menu function from interface is called and the recently created circuit object is passed to it. This is the last command in main.cpp.

Interface includes the welcome function mentioned before, and two general input validation functions (one for a set of allowed values, string_input() and one for positive doubles, number_input()). It also defines two namespaces, "simple" and "compound" and within them, sets of accepted values for user input, the menu and a function to add components: this calls the correct form of the template function of the circuit class, depending on user input.

```
namespace compound
{
    void menu(circuit c);
    std::tuple<circuit, std::complex<double>> add_component(std::tuple<circuit, std::complex<double>> pair);
    const std::vector<std::string> menu_choices = { "1", "2", "3", "4" };
}
```

Figure 5: Menu function, function to add components and the acceptable user input values for choosing an option in the menu declared in the compound namespace.

2.2.1 namespace: simple

This namespace is used for all-series or all-parallel circuits. Since the impedance can be calculated with a single formula, there is no need to store its value. At any point, if the user requests, the get_impedance() function is able to return the correct value depending on the current components. This means that modifying the circuit (frequency or structure) does not require several recalculations, but can be done easily. Displaying the circuit is also possible as its layout is predictable.

The menu function in this namespace therefore presents more options for the user (which are as listed on Figure 1).

Adding a component is also a lot simpler in this namespace: add_component() allows the user to select the type of component to be added and the value that characterises it and returns a modified circuit that includes the new component.

2.2.2 namespace: compound

In this namespace add_component() also asks the user to specify how they would like to connect the new component as this information is needed to calculate the total impedance of the circuit. Instead of keeping track of this for every component (which would require modifying the circuit or component classes), this function takes in a tuple of the original circuit and its impedance and returns a tuple of the new circuit and its updated impedance, depending on the initial value and the way of connecting the new component, specified by the user. Because impedance is stored as a complex number (using the complex class from the Standard Template Library) and not just calculated from the components at any time, when changing the properties of the circuit, it is not be possible to update its value accordingly. Due to the freedom in structuring the circuit, it is also more difficult to format the display of the circuit in an appealing way. The menu in this namespace therefore has less options for the user.

3 Results

This section contains the output of the code for some example user inputs.

```
Enter a number to choose what to do (1-7): 1
Please specify what type of component you'd like to add. Type 'resistor' or 'R' for a resistor,
'capacitor' or 'C' for a capacitor or 'inductor or 'I' for an inductor. R
Enter the resistance (in Ohms): 2
Component added!
```

Figure 6: Example of user adding a resistor of 2 Ohms to the circuit.

Choosing "Add component" in the menu asks the user to enter the parameters for the component they would like to add. On Figure 6, we can see the user input necessary to add a resistor with a resistance of 2 Ohms to the circuit and the confirmation that it has been added.

For another example circuit including an inductor and a resistor in series, the circuit information can be printed out by choosing "Display circuit information" in the menu. Choosing "Display component information" shows the information on the components that the user has added so far. An example of these outputs is shown on Figure 7.

For simple circuits, the user can also display a schematic diagram of the circuit they have

constructed. An example of a series RLC-circuit is shown on Figure 8.

```
Enter a number to choose what to do (1-6): 2
Circuit type: series.
The frequency of the circuit is 1 Hz.
Impedance of circuit Z(real, imaginary) = (5,18.8496)
Magnitude of impedance = 19.5014
Phase angle of impedance = 1.31151
Number of components: 2
Enter a number to choose what to do (1-6): 4
Component information:
component: inductor
        impedance: (0,18.8496)
        magnitude of impedance: 18.8496
        phase: -1.5708
component: resistor
        impedance: (5,0)
        magnitude of impedance: 5
        phase: 0
Enter a number to choose what to do (1-6):
```

Figure 7: Example of the program displaying circuit and component information.

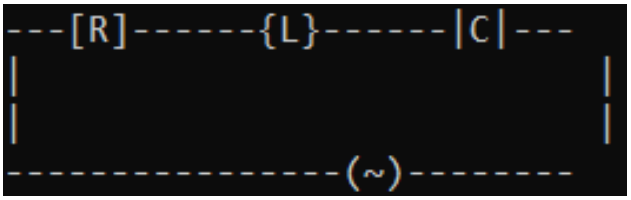

Figure 8: Output of display_circuit() function for a circuit with a resistor, capacitor and inductor connected in series.

4 Discussion and conclusions

The main objectives of this project have been completed - the user is able to construct a circuit by adding an arbitrary number of components in series or in parallel and the program successfully calculates the impedance, magnitude of impedance and the phase angle for both the circuit and the individual components. Along with this, for simple circuits the program is able to display the circuit in a schematic way and there are also options to modify the initial frequency set or change how the components are connected.

There are various ways in which the program could be improved, such as creating more generalised class for circuits that that does not need the user to specify its type in advance (for example by using nested circuits). The ability to select, modify or remove components already added to the circuit would make the program more convenient for the user as well. The component classes could be extended for various other types of components and non-ideal elements could be accounted for by adding relevant members to each class to describe their deviations from ideal elements.

References

[1] Roger A Freedman Hugh D Young. *University Physics with Modern Physics, 15th edition.* Pearson, 2019. Chapter 31, Alternating Current.