

May 14, 2017

AC circuit impedance calculator

Object Oriented Programming final project

Dónal Murray
Student ID: 8381250

University of Manchester, Manchester, United Kingdom

Abstract

A program is presented for the manipulation and storage of alternating current circuits. Series and parallel circuits can be constructed using an arbitrary number of resistors, capacitors, inductors or other circuits composed of the listed components. Impedences of all components in the circuit as well as the total impedance of the circuit are calculated and circuits diagrams can be printed. The program allows the user to save the current project to a file and load old projects from a file. The program is designed to fully comply with the current C++ standard and to make use of advanced features of object oriented programming included in C++11.

1 Introduction

This report aims to outline the design and implementation of a program written in C++ to calculate the impedance of AC circuits consisting of any number of ideal resistors, capacitors and inductors in series or parallel. The program attempts to make best use of the advanced idiomatic features of C++.

C++ is an object oriented programming language that was created by Bjarne Stroustrup in 1983, initially with the aim to introduce object oriented programming into C [?] but which has since diverged to become a programming language in its own right and one of the most popular programming languages in the world [?].

Object oriented programming is centred around classes and their instantiations called objects which contain data and functions which can be either public, allowing the data or function to be accessed from outside of the class, or private, allowing access only within a specified scope. There is a third classification, protected, which will be discussed in section 2.

1.1 AC circuit theory

In contrast with DC circuit theory, where the current through a circuit is opposed only by the resistance, R , of that circuit, in AC circuit theory the current through an AC circuit is opposed by a combination of both the resistance and reactance, X and their combination is called the impedance, Z . Whereas resistance is the opposition to the flow of current, the reactance is the opposition to the *change* in flow of current. This is why the reactance of components is irrelevant with DC circuits, as the current through the circuit is not changing.

Impedance

Mathematically when dealing with AC circuits, complex numbers are used for the impedance with the real part representing the resistance and the imaginary part representing the reactance such that

$$Z = R + jX, \tag{1}$$

where, to agree with convention, $j = \sqrt{-1}$ is used in place of i to avoid confusion with the current which is conventionally denoted I . The program is designed to make all of its calculations with ideal components, which means that a resistor only has resistance and zero reactance and that inductors and capacitors only have reactance and zero resistance. Therefore assuming ideal components, the impedance of a capacitor, inductor and resistor can be found using

$$Z_C = \frac{1}{j\omega C}, \quad Z_L = j\omega L, \quad \text{and} \quad Z_R = R \tag{2}$$

respectively, where ω is the angular frequency of the AC current and C and L are the capacitance and inductance respectively.

With n components connected in series, the total impedance can be found by taking the sum of the impedances of the constituent components with

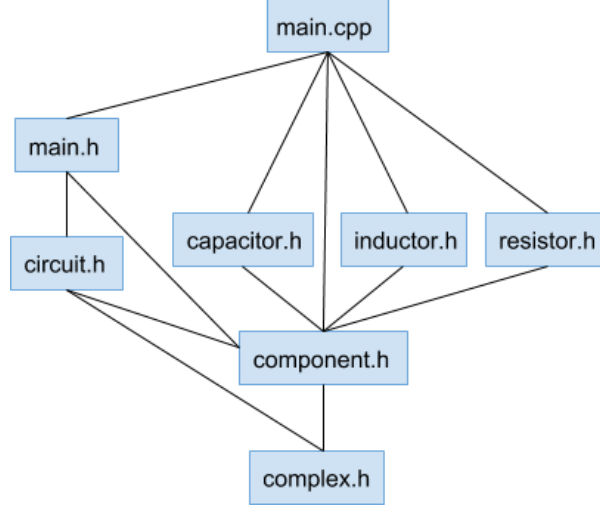


Figure 1: The hierarchy of header files showing the main file at the top and the chain of its dependencies leading down.

$$Z_{total} = Z_1 + Z_2 + \dots + Z_n \quad (3)$$

and with n components connected in parallel, the inverse of the total impedance can be found by taking the sum of the inverse of the impedances of the constituent components with

$$\frac{1}{Z_{total}} = \frac{1}{Z_1} + \frac{1}{Z_2} + \dots + \frac{1}{Z_n} \quad (4)$$

which requires the conjugate and modulus to be found for each of the complex impedances, which is discussed in section 2.

Phase difference

By plotting the impedance on the complex plane, the resulting plot is called a phasor diagram [] and can be used to determine the phase difference, which describes the offset between the voltage and the current. The phase difference is found by taking the argument of the impedance. Capacitors have a phase difference of -90° which means that the voltage lags the current by 90° and inductors have a phase difference of 90° which means that the voltage leads the current by 90°

2 Code design and implementation

In the design of this project, the code was split into seven header files, the hierarchical structure of which is shown in figure 1, which contained class definitions, function prototypes and a namespace definition while the implementation of the classes, the full functions and the `main` function were separated into seven source files in line with the practice of abstraction, whereby the user is only exposed to the parts of the program

which they need to actually use the program, and the details like the member data are hidden from them.

Splitting the code into several files makes it easier to reuse code, limits the `#include` preprocessor directives to what is required only for the small piece of code in the file, and not for the entire program, reducing the risk of name clashes. Separating into different files can also reduce the compile time, as only the parts that have been changed need to be recompiled and the rest can just be linked again by the compiler.

Header guards were put in place to avoid header files being included multiple times. By using the preprocessor directives to define a unique constant for each header file, the compiler could check using a single `if` statement whether the header had already been included. This avoids compiler errors caused by apparent redefinitions of existing functions and other problems.

A custom complex class was used for dealing with complex numbers instead of using the standard library type definition. The class contained member data of the type `double` to store the real and imaginary parts and member functions including functions to return the complex conjugate, modulus and argument of a complex number, all of which were used in the calculation of the impedance and the phase difference.

For the components, an abstract base class `Component` was created, from which three derived classes inherited: `Resistor`, `Capacitor` and `Inductor`. The component class defined the functions that the three derived classes had in common – for instance functions to access and modify the label of the components and a general constructor which was called by the derived classes. The component class had a pure virtual function `get_impedance()` which was then defined in the derived classes and meant that polymorphic arrays could be created so that the derived classes would share a common interface. In this way the same function would carry out a different calculation depending on what type of object it was called on.

A circuit base class was created which was similar to the component base class in that it functions as a shared interface for the series and parallel derived classes. Again, the function to calculate the total impedance was left as a pure virtual function in the base class, to be defined in the correct way within the series and parallel classes. The series and parallel classes also each had specific functions to print out a circuit diagram in the correct layout for the circuit type. This was done using

The aim of the project was to be able to add components to a library of components which could be printed and the components stored in the library could be connected in specified groups to create circuits. The aim was to have a similar functionality for all circuits that had been made so far, so that circuits from the circuit library could be nested inside other circuits. This was achieved by making two polymorphic vectors of base class pointers – one for components and the other for circuits. These polymorphic vectors were put inside the namespace `libs` in `main.h` so that it could be accessed from any function using the namespace and the binary scope operator (`libs::`). If the libraries were used multiple times in the same function, `using namespace libs` could be used once to remove the need to type the namespace and binary scope operator repeatedly within the relevant function.

prototyping functions

vectors

smart pointer

lambdas
namespaces and binary scope operator
function template
static data

3 Results

(illustrates how code is used including input and output details)

4 Discussion

A possible extension of this project would be the ability to create circuits of non-ideal components, taking into consideration that real components have a combination of resistance and reactance and can have an arbitrary phase difference. Changing the circuit class to a class template would mean that circuits could be built using either ideal component objects or non-ideal component objects.

Another possible extension would be a bill of materials calculator for the components that have been used to build the circuit. Using databases for a company like Maplin [1], a list of components with their product codes and an estimate of the total price could be generated to make it easy for the user to translate their plans to a real circuit.

5 Conclusion

The aim of the project was to produce a program which could calculate the impedance and phase difference for a parallel or series circuit consisting of any arbitrary number of capacitors, inductors and resistors. The program can not only perform these tasks but also has additional features including:

- the ability to nest an arbitrary number of series or parallel subcircuits and then recursively calculate the impedance and phase difference of each of the constituent parts to give the total impedance and phase difference for the entire circuit;
- the ability to print circuit diagrams of any complexity through the use of placeholders for the subcircuits;
- the ability to save a project and load it at a later date, allowing for the same component library to be used repeatedly for different projects.

The program demonstrated the use of the C++11 standard library, using many advanced features of object oriented programming in their proper context and using `throw/catch` exception handling to deal with errors that would occur at runtime.

References

- [1] Maplin, *Maplin electronics is a company registered in england no. 1264385*,
<http://www.maplin.co.uk/>.