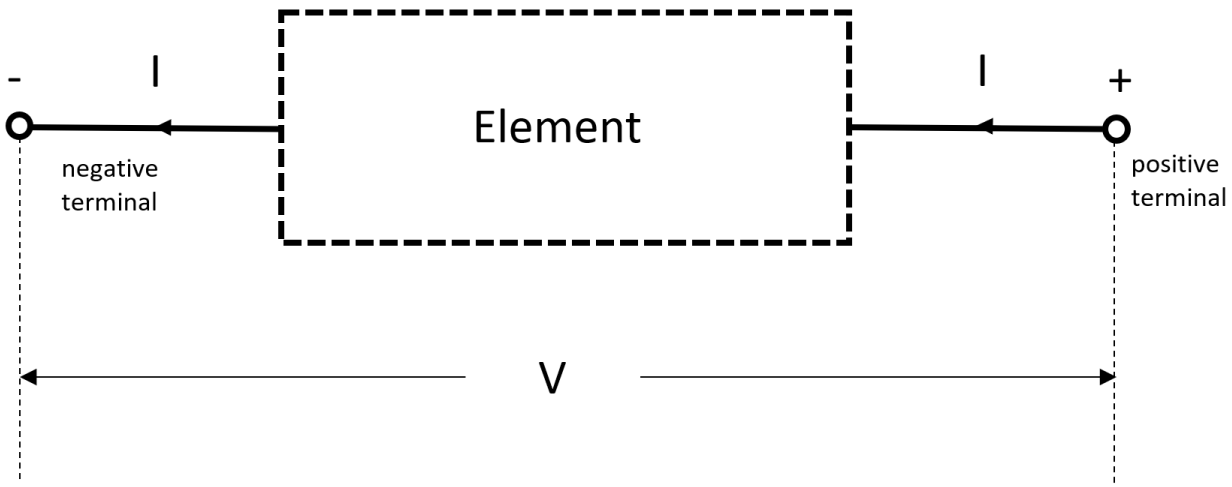
An OPP C++ implementation on A.C. Circuits

- ***Shiling Liang***
  School of Physics and Astronomy
  University of Manchester

Abstract

An OPP C++ base program is written to simulate a general series-parallel AC circuit. Three ideal components(resistor, inductor and capacitor) are derived from an abstract class of component. A circuit class is defined to let user construct series-parallel A.C. circuits of any size and any allowed structure. The circuit construction is achieved via a recursive definition of circuits, which allows two circuits to be connected in series or in parallel to construct a larger circuit. Information of circuits and components can be output, including impedance, phase shift and magnitude. The topology structures of circuits are drawn through an ASCII-based tree diagram.

Introduction



In an alternating-current(AC) circuit, an AC source applies oscillation voltage on a two-terminal circuit element. Then an alternating current is generated in the circuit. Here only the parallel-series is discussed, in which the circuit components are connected either in parallel or series. Three kinds of ideal components, resistor, inductor and capacitor, are used to construct the circuit. The relation between voltage and current is measured by the impedance of the circuit element as

$$V = ZI$$

This equation is called impedance Ohm's Law. Where V is the voltage applied, I is the current., Z is the impedance. If the voltage is a sinusoidal function of time, the impedance Z externally depends only on the frequency of the voltage. Hence both voltage and current are sinusoidal, and they are usually represented as complex-valued functions

$$V = |V|e^{i(\omega t + \phi_V)}$$

$$I = |I|e^{i(\omega t + \phi_I)}$$

where U_0 and I_0 are the amplitude of voltage and current, respectively. ω is the angular frequency. ϕ_V and ϕ_I are the phases. The impedance is also a complex number

$$Z = R + iX = |Z|e^{i\arg(Z)}$$

where R and X are both real number. R is called resistance and X is called reactance. $\arg(Z)$ is the argument of Z , defined by $\arg(Z) = \arctan(R/X)$. Substituting above equations in to the impedance Ohm's law gives

$$|V| = |Z||I|e^{i(\phi_I - \phi_V + \arg(Z))}$$

$$\Rightarrow |Z| = \frac{|V|}{|I|}, \quad \Delta\phi = \phi_V - \phi_I = \arg(Z)$$

it is shown that the magnitudes obeys classical Ohm's law. The phase difference between current and voltage equals to the argument of impedance. From Kirchhoff’s laws, the impedance of n series or parallel connected circuit elements can be calculated as

$$Z_{series} = \sum_{i=0}^{n} Z_i$$
$$Z_{parallel} = \frac{1}{\sum_{i=0}^{n} \frac{1}{Z_i}}$$

Ideal Components

As mentioned above, three kinds of basic circuit components are used to construct circuits. Their properties are summarised in the following table.

Component name	Characteristic quantity	Differential equation	Impedance	Phase shift
Resistor	Resistance R	$V = IR$	R	0
Inductor	Inductance L	$V = L\frac{dI}{dt}$	$iL\omega$	$-\frac{\pi}{2}$
Capacitor	Capacity C	$I = C\frac{dU}{dt}$	$-i\frac{1}{\omega C}$	$\frac{\pi}{2}$

Non ideal components

In reality, the circuit components may not be exactly described by the three ideal components. For example, a real capacitor is not perfectly dielectric. To simulate a real capacitor, we can add a resistor of large resistance to a capacitor in series. Other real components can also be constructed using the three base components. This program does not only allow the user to connect components but also allows the user to connect constructed circuit elements. Therefore the user can create their own non-ideal components and store them for later use. So non-ideal components are not provided straightforward. We encourage users to construct non-ideal components by their own.

Code design and implementation

File structure

This project has *main.cpp* file containing the main function and 3 *header file (.h)* associated with 3 s*ource files (.cpp)* in which the classes and functions are defined. *components.h* declare an abstract class, `component`, as the base class, and its three derived class `resistor`, `capacitor` and `inductor.` The actual definition is contained in *components.cpp*. *circuits.h* along with *circuit.cpp* define a `circuit` class for circuit construction. *interface.h* and *interface.cpp* include the functions related to the user interface.

Class definitions

Component class

A generic component class as defined as an abstract base class. The three ideal AC circuit component classes, `resistor`, `capacitor` and `inductor` are derived from this base class as polymorphism.

```
class componet{
protected:
    complex<double> impedance;
    double frequency;
public:
    member functions;
}
```

The essential member variable of `component` class is the impedance, which is a complex number. And a pure virtual function `setimpedance()` is declared. It is implemented in the derived classes according to the equations mentioned before for different kinds of components. Once the impedance is obtained, the magnitude as well as phase shift can be easily obtained. All this information is printed out by an `info()` function.

Circuit class

Besides the components, a circuit class is needed to represent the topology structure of AC circuits. Since we only discuss parallel-series circuits here, each circuit can be described by a series-parallel graph defined in graph theory.

A circuit class inherits impedance related member variables and functions from component class to avoid redefinition. Additionally, the circuit needs to contain the circuit topology structure. The circuit topology structure is represented via recursive definition. A vector of circuit pointer called `subcircuits` is used to store the sub-circuits of the current circuit. Moreover, a string `conntype` member variable is defined to indicate the connection type of the sub circuits. In this program, a circuit can have at most two sub circuits, which have four connection types: `parallel`, `series`, `single`, `empty` and `component`. The first two types indicate that the two sub-circuits are connected in parallel or series, respectively. The `single` and `empty` types are defined in case of one or two sub-circuits is deleted. For `empty` type, the impedance is set to $0 + i0$, so that it is equivalent to a section of wire. The `component` type is for the base case which has no sub-circuits. Instead, a generic component (resistor, capacitor or inductor) is stored in a component pointer `basecomponent`. Hence the recursively defined circuits ended with `component` or `empty` branches. Corresponding constructors are defined to create any circuits mentioned above.

```
class circuit: public component{
protected:
    vector<circuit*> subcircuits;
    string conntype;
    component* basecomponent;
public:
    member functions;
}
```

A `delsubcircuit()` member function is introduced to allow the user to delete any unwanted sub-circuit. An `info()` member function is used to output all information of the current circuit. Impedance along with the magnitude and the phase shift are printed as required. The circuit topology structure is not print out directly; however, the decompose tree of the circuit is shown. According to graph theory[1], a series-parallel graph can be decomposed into a tree diagram. Here we constrain the branches of each node to no more than two (2 for `series`/`parallel` type, 1 for `single` type and 0 for `empty` or `component` type). Then the topology of a circuit can be illustrated via the tree diagram. A `plot()` member function is provided to draw such a diagram.

User Interface

A text-based user interface is created to guide the user to construct circuits. Two vectors are created to store components and components, respectively. `circuits_container` is a vector of circuit pointer. Every circuit constructed is stored in the container by order of construction. To construct a new circuit, the user can use any circuits stored in the container. `components_container` is a vector of component pointer to store components. The components can be taken from the container to create `component` type circuit.

The program starts with a menu, which provides the actions which can be taken. When any action is ended, the program will return to the menu page to ask for further action. Essential actions are provided such as creating components and circuits, getting information of created circuits and components, modifying existing circuits and components. During the constructing, anything created will be stored in the containers. The container can even be freed through the `Clear all data` option.

Error handling

As the `empty` type circuit has zero impedance, it may cause the 'division by zero' problem when it is added to the circuit in parallel. However, we do not intend to directly forbid this case since short circuit also has its physical meaning. To avoid dividing by zero, the minimum finite value of `double` type is added to the denominator while calculating the impedance of parallel connection.

```
impedance=1/(1/(impedance_1+e)+(impedance_2+e)+e)
if (abs(impedance.real()) < 2*e) impedance.real(0.0);
if (abs(impedance.imag()) < 2*e) impedance.imag(0.0);
```

where `e = std::numeric_limits<double>::min()` is the minimum finite value of `double` type. A threshold value is rounded to zero by setting a threshold `2*e`.

The constraints on user input are carefully considered. Non-negative characteristic values (resistance, inductance and capacity) for components are ensured. Moreover, the input index of the intended visiting components/circuits are required in the range of containers via comparing the input index with the size of container vectors.

Results

The program starts with the menu below shown in the console.

```
==============Menu================
What do you want to do now:
1: Create components
2: Create circuits
3: List all stored components
4: Get information of existing circuits
5: Modify existing circuits
6: Clear all data (circuits and components)
7: Help
8: Quit
==================================
input the number to choose:
```

To start constructing a circuit, the user can enter `1` to create some components or `2` to start the circuit construction. Since the recursively defined circuit must start with `component` or `empty` circuit while there is no circuit at the beginning, if `2: Create circuits` is chosen, the program will guide the user to constructed the first `component circuit`.

```
You are going to construct a circuit
There is not any circuit constructed, at least one "component" circuit is required.
1: choose one from components container; 2: create a new component
```

As shown above, there are two ways to construct a `component` circuit. The user can directly create on or choose one from components container. Note that if the user chooses to create one directly, the created component will only be stored in the `component` circuit, and no component will be added to the `components_container`. Strictly speaking, the `components_container` is not necessary for circuit constructing, while storing components in the container can simplify the

constructing process.

Once a circuit is constructed, the user can visit the circuits by entering `4: Get information of existing circuits` from the menu page. To test the code, an RLC series circuit is constructed with the following parameters:

R	L	C	frequency	expecting impedance of RLC
$500\ \Omega$	$1\ \mathrm{H}$	$0.0001\ \mathrm{F}$	$100\ \mathrm{Hz}$	$500 + i612.403$

```
impedance = (500,612.403), frequency = 100 Hz.
circuit type is " series"
The magnitude is 790.593 Ohm and the Phase shift is: 0.886102 rod
*Tree diagram
S4
|-L2
`-S3
  |-R0
  `-C1
```

It is shown that the impedance obtained is exactly as what is expected. Note the tree diagram do not only show the structure of circuit but also show the construction steps. As shown by this tree diagram, `R0` and `C1` are connected in series as `S3` first. Then `S3` are connected with `L2` in series to get `S4`. To see the information of any sub-structure of a circuit, you can visit them by the index shown in the tree diagram. The index of each sub-structure is just behind each one's name. For example, from the tree, we can know that `R0` is stored at `circuits_container[0]`. As the position is known, the `4: Get information of existing circuits` from the menu page allows you to access it.

Conclusions and discussion

This program can simulate any series-parallel AC circuit. The most powerful part is that it allows constructing circuit by connecting two circuits. Therefore any possible series-parallel circuits can be constructed using this program. A user-friendly interface is provided to guide the user. The tree diagram gives an explicit representation of the topology structures of circuits. It provided more information, such as the constructing process, than a plain circuit diagram

Several further improvements can be made if time allowed an external library be included. Gnuplot with its ASCII plot package can be used to draw a voltage-current diagram and even voltage-time and current-time diagram. The tree diagram can also be simplified via further topology analysis. Due to the constraint on the number of branches, the current tree diagram is not unique to the corresponding circuit. Thus allowing nodes have more than two branches can give a more nature diagram.

Reference

[1] Rawlins, C., 2000. *Basic AC circuits*. Elsevier.

[2] He, X. and Yesha, Y., 1987. Parallel recognition and decomposition of two terminal series parallel graphs. *Information and Computation*, *75*(1), pp.15-38.