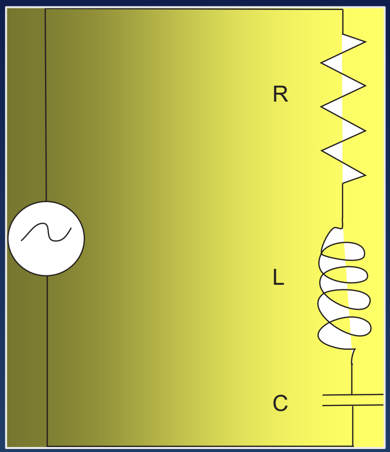
Object-Oriented Programming
OOP (CS)



1st simple OOP, 2nd advanced (text)

Text ≠ advanced objects

≈ procedural (F90, C)

OOP = important philosophy

Rubin H Landau

With

Sally Haerer and Scott Clark

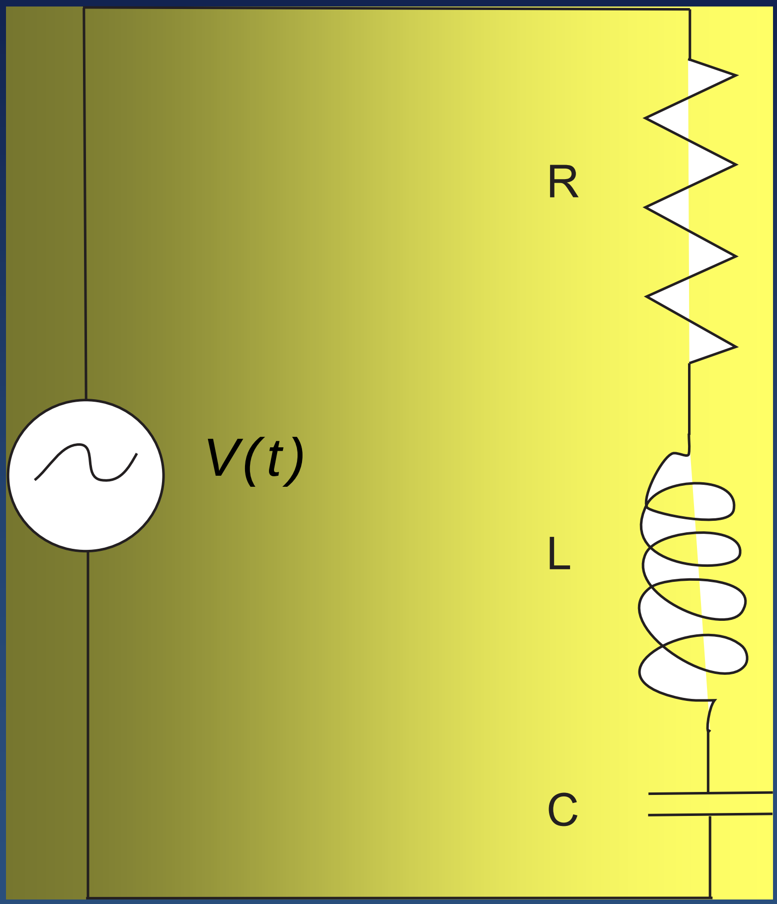
Computational Physics for Undergraduates

BS Degree Program: Oregon State University

"Engaging People in Cyber Infrastructure"

Support by EPICS/NSF & OSU

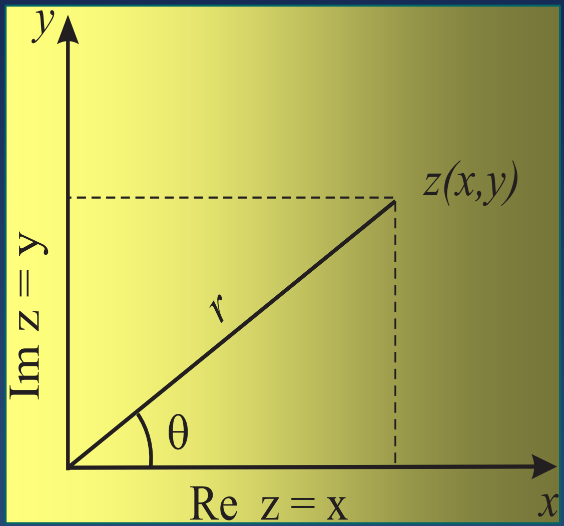
Problem: Complex Impedance of RLC Circuit



- **Resistor *R* = 1000/1.5 Ω**
- **Inductor *L* =1000 H**
- **Capacitor *C* = 1/1000 F**
- **AC voltage** $V(t) = V_0 \cos \omega t$
- $I(t) = ?$
- $0 < \omega < 2/s$

Complex Numbers (Math)

- **Useful in math and science**
- **Double work output, slight increase in input**
- **Manipulate *z*, then ⇒ Re *z*, Im *z***
- **Imaginary number $i = \sqrt{-1}$**
- **Complex $z = x + i\,y$,**
  - **Re $z = x$, Im $z = y$**



- **Java, C: `doubles, floats, ints,` … ≠ complex**
- **Fortran: built in = "primitive data types"**
- **We: construct `complex` number objects**

Complex Rules (to program)

Addition:

$$z_1 + z_2 = (x_1 + x_2) + i(y_1 + y_2),$$

Subtraction:

$$z_1 - z_2 = (x_1 - x_2) + i(y_1 - y_2),$$

Multiplication:

$$\begin{aligned} z_1 \times z_2 &= x_1 + iy_1) \times (x_2 + iy_2), \\ &= (x_1x_2 - y_1y_2) + i(x_1y_2 + x_2y_1), \end{aligned}$$

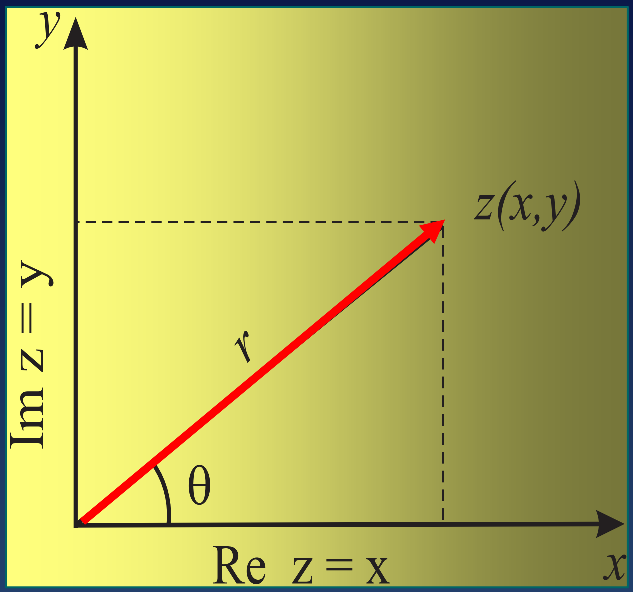
Division:

$$\begin{aligned} \frac{z_1}{z_2} &= \frac{x_1 + iy_1}{x_2 + iy_2} \times \frac{x_2 - iy_2}{x_2 - iy_2}, \\ &= \frac{(x_1x_2 + y_1y_2) + i(y_1x_2 - x_1y_2)}{x_2^2 + y_2^2}, \end{aligned}$$

Complex Conjugate:

$$\begin{aligned} z^* &= x - iy \\ \Rightarrow \quad z \times z^* &= (x + iy)(x - iy) = x^2 + y^2 \end{aligned}$$

Polar Form of *z*



Vector in "imaginary" space

$$r = \sqrt{x^2 + y^2}, \qquad \theta = \tan^{-1}(y/x)$$

$$x = r\cos\theta, \qquad y = r\sin\theta$$

Euler's Theorem

$$e^{i\theta} = \cos\theta + i\sin\theta$$

Polar Representation

$$z \equiv x + iy = re^{i\theta} = r\cos\theta + i\,r\sin\theta$$

Get Some Exercise!

Given:

$b = 1 + 2\,i, \qquad c = 4 + i$

What are the values of:

$b + c$

$b - c$

$b \times c$

$|c|$

b / c ?

Resistance ⇒ Impedance (PH)

- **Kirchoff voltage law**
- **Series ⇒ $\sum V_i = V_0 \cos \omega t$** (1)

$$\frac{dV(t)}{dt} = R\frac{dI}{dt} + L\frac{d^2I}{dt^2} + \frac{I}{C} \quad (2)$$

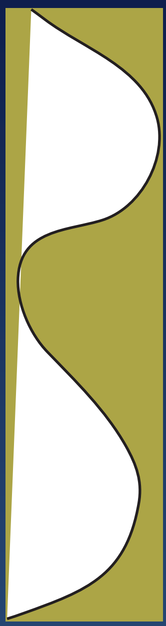
Complex input $$V(t) = V_0 \cos \omega t = V_0 \, \mathbf{Re}\, e^{-i\omega t} \quad (3)$$

Complex output $$I(t) = I_0 e^{-i\omega t}? \quad (4)$$

New Ohm's law ⇒ $$I(t) = \frac{1}{Z} V_0 e^{-i\omega t} \qquad (V = IZ) \quad (5)$$

Complex impedances
Add in series $$Z = R + i\left(\frac{1}{\omega C} - \omega L\right) \quad (6)$$

Abstract Data Types, Objects (CS)



- **What do you see?**
- **Abstract: suggests more than meets the eye**
- **Abstract & formal concepts**
  - **pervade math & science: easier**
  - ***e.g.* velocity *v(t)* no observe**
- **CS *abstract object* = symbol + multiple parts**
- **"Primitive data types": `integers, floats`, …**
- **"Abstract data types" = ∑ primitive**
- **"*Class*" = objects, their data + methods**
  - ***e.g.* complex numbers, plots, matrices**
- **"*Instance*" = object + specific values for parts**
  - ***e.g.* `(Re z, Im z)` ⇒ `2 + 3 i`**

Formal CS: 3 Properties of Abstract Data Types

1. ***Typename:*** **elementary pieces $\Rightarrow$ new data types**
2. ***Set values:*** **mechanism values to data type**
3. ***Set operations:*** **rules of operations (methods)**

e.g. $z = x + i\,y$

a. declare ```complex z`**'' = real + i imaginary**

b. `Re` z `=  double` x, `Im z = double y`

c. rules of complex arithmetic

Implement: Program Structure

- **Start: declaration, e.g.** `double x`
  - **tells compiler kind of variable**
  - **rule: declare every variable**
- **Primitive types (built-in):** `double, float, int, …`
- **User-defined, abstract data types too**
  - `Complex z`
  - **= create object = dynamic = nonstatic (no "**`static`**")**
- **Object Methods: either static or dynamic**
  - **static methods ↔ arguments:** `mod(z)`
  - **dynamic methods ↔ modify objects:** `z.mod`
  - **dynamic: power of OOP**

Java Implementation: `Complex.java`

```
public class Complex
{ public double re, im;                                   // Nonstatic class
  public static void main(String[] argv)                  // Main method
        {  Complex a,  b;                                 // Declare Complex objects
           a = new Complex();                             // Create objects
           b = new Complex(4.7, 3.2);
          Complex c = new Complex(3.1, 2.4);              //Declare & create
          System.out.println("a = (“ +  a.re + ",“ + a.im + "), “  );
  public Complex ()                                       // Default constructor
        { re = 0;     im = 0; }
  public Complex(double x, double y)                      // Full constructor
        { re = x;     im = y; }
  public static Complex add(Complex a, Complex b)         // Static add method
        { Complex temp = new Complex();                   // Create Complex temp
          temp.re = a.re + b.re;                          // Dot operations
          temp.im = a.im + b.im;
          return temp; }
```

Nonstatic Methods: `ComplexDyn.java`

```
public class ComplexDyn
{public double re, im;                                  // Nonstatic class variables
  public static void main (String[] argv)
   { ComplexDyn a, b;                                   // Declare 2 Complex objects
      a = new ComplexDyn();                             // Create objects
      b = new ComplexDyn(4.7, 3.2);
      ComplexDyn c = new ComplexDyn(3.1, 2.4);          // Declare, create
      System.out.println("a = ("+ a.re + ",“ + a.im + "),“ );
      c.add(b);                                         // Non-static addition c = c+b
      c.mult(b); }                                      // Non-static multiplication
  public ComplexDyn(double x, double y)                 // Constructor
    { re = x;     im = y; }
  public void mult(ComplexDyn other)                    // Dynamic other*this
   { ComplexDyn ans = new ComplexDyn();                 // Intermediate
      ans.re =  this.re * other.re  - this.im * other.im;
      ans.im = this.re * other.im + this.im * other.re;
      this.re = ans.re;                                 // Copy value into returned object
      this.im = ans.im;  }
```

Problem Solution: Complex Currents

1. **Extend class `Complex.java` or `ComplexDyn.java`**
2. **Add complex methods:**

   subtract, conjugate, modulus, phase

3. **Test your methods:**

   $z + z = 2z, \quad z + z^* = 2\,\mathrm{Re}\,z$

   $z - z = 0, \quad z - z^* = 2\,\mathrm{Im}\,z$

   $z z^* = |z|^2 = \text{real}$

4. **Hint: debug simple, known answers:**

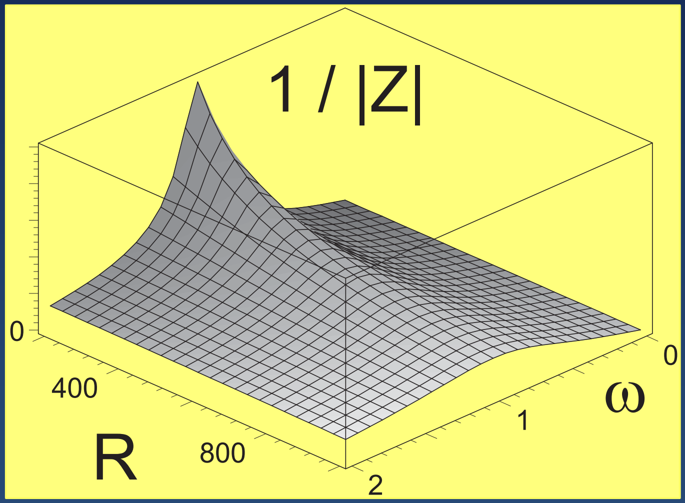
   pure real/imaginary

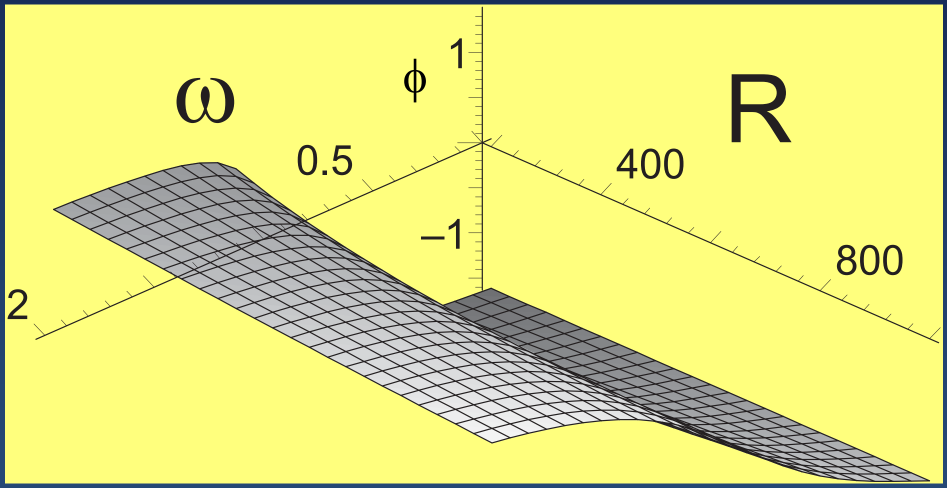
Solution, Assessment: Complex Current

$$\frac{dV(t)}{dt} = R\frac{dI}{dt} + L\frac{d^2I}{dt^2} + \frac{I}{C} \qquad (1)$$

$$I(t) = \frac{1}{Z}V_0 e^{-i\omega t} = \frac{V_0}{|Z|}e^{-i(\omega t+\theta)} \qquad (2)$$

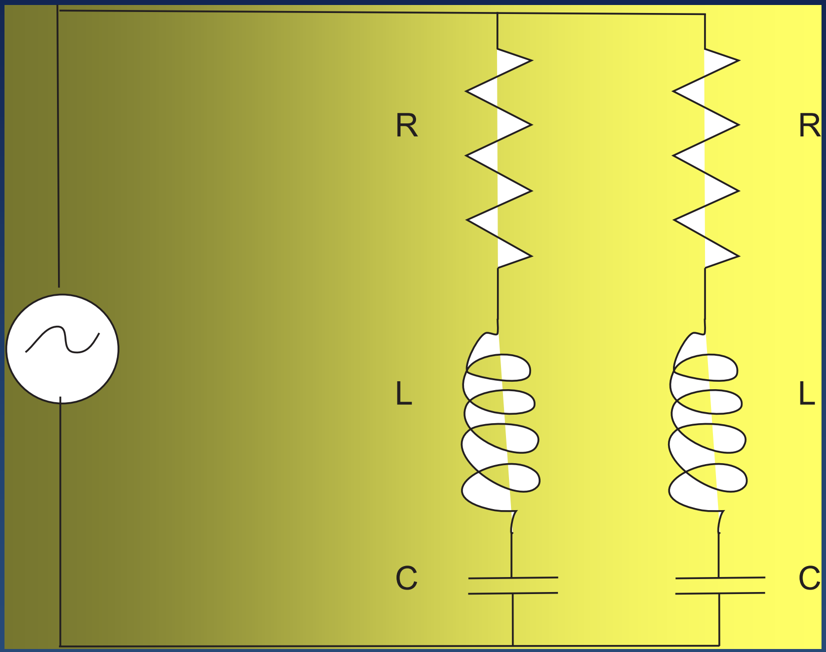
$$Z = R + i\left(\frac{1}{\omega C} - \omega L\right) \qquad (3)$$





1. **Compute, plot magnitude and phase of current *I* (ω)**
2. **Surface plot of the *1/Z*(ω, *R*), ϕ(ω, R)**
3. **Maximum |I| (resonance) at ω = 1/ √LC**
4. **Construct 3-D visualization *Z* (Re ω, Im ω)**

Extension



- **Compute current**
- **Add *Z* in series**
- **Add inverse Z in parallel**

