ComplexArithmetic

February 11, 2021

1 Complex Arithmetic

This is a tutorial designed to introduce you to complex arithmetic. This topic isn't particularly expansive, but it's important to understand it to be able to work with quantum computing.

This tutorial covers the following topics:

- Imaginary and complex numbers
- Basic complex arithmetic
- Complex plane
- Modulus operator
- Imaginary exponents
- Polar representation

If you need to look up some formulas quickly, you can find them in this cheatsheet.

If you are curious to learn more, you can find more information at Wikipedia.

This notebook has several tasks that require you to write Python code to test your understanding of the concepts. If you are not familiar with Python, here is a good introductory tutorial for it.

Let's start by importing some useful mathematical functions and constants, and setting up a few things necessary for testing the exercises. **Do not skip this step**.

Click the cell with code below this block of text and press `Ctrl+Enter` (`+Enter` on Mac).

[3]:
```
# Run this cell using Ctrl+Enter ( +Enter on Mac).
from testing import exercise
from typing import Tuple

import math

Complex = Tuple[float, float]
Polar = Tuple[float, float]
```

2 Algebraic Perspective

2.1 Imaginary numbers

For some purposes, real numbers aren't enough. Probably the most famous example is this equation:

$$x^2 = -1$$

which has no solution for x among real numbers. If, however, we abandon that constraint, we can do something interesting - we can define our own number. Let's say there exists some number that solves that equation. Let's call that number i.

$$i^2 = -1$$

As we said before, i can't be a real number. In that case, we'll call it an **imaginary unit**. However, there is no reason for us to define it as acting any different from any other number, other than the fact that $i^2 = -1$:

$$i + i = 2i \quad i - i = 0 - 1 \cdot i = -i(-i)^2 = -1$$

We'll call the number i and its real multiples **imaginary numbers**.

A good video introduction on imaginary numbers can be found here.

2.1.1 Exercise 1: Powers of i.

Input: An even integer n.

Goal: Return the nth power of i, or i^n.

Fill in the missing code (denoted by `...`) and run the cell below to test your work.

[4]:
```
@exercise
def imaginary_power(n : int) -> int:
    # If n is divisible by 4
    if n % 4 == 0:
        return 1
    else:
        return -1
```

```
Success!
```

Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

2.2 Complex Numbers

Adding imaginary numbers to each other is quite simple, but what happens when we add a real number to an imaginary number? The result of that addition will be partly real and partly imaginary, otherwise known as a **complex number**. A complex number is simply the real part and the imaginary part being treated as a single number. Complex numbers are generally written as the sum of their two parts: $a + bi$, where both a and b are real numbers. For example, $3 + 4i$, or $-5 - 7i$ are valid complex numbers. Note that purely real or purely imaginary numbers can also be written as complex numbers: 2 is $2 + 0i$, and $-3i$ is $0 - 3i$.

When performing operations on complex numbers, it is often helpful to treat them as polynomials in terms of i.

2.2.1 Exercise 2: Complex addition.

Inputs:

1. A complex number $x = a + bi$, represented as a tuple `(a, b)`.
2. A complex number $y = c + di$, represented as a tuple `(c, d)`.

Goal: Return the sum of these two numbers $x + y = z = g + hi$, represented as a tuple `(g, h)`.

> A tuple is a pair of numbers. You can make a tuple by putting two numbers in parentheses like this: `(3, 4)`. * You can access the nth element of tuple `x` like so: `x[n]` * For this tutorial, complex numbers are represented as tuples where the first element is the real part, and the second element is the real coefficient of the imaginary part * For example, $1+2i$ would be represented by a tuple `(1, 2)`, and $7-5i$ would be represented by `(7, -5)`.
>
> You can find more details about Python's tuple data type in the official documentation.

Need a hint? Click here

Remember, adding complex numbers is just like adding polynomials. Add components of the same type - add the real part to the real part, add the complex part to the complex part. A video explanation can be found here.

[6]:
```
@exercise
def complex_add(x : Complex, y : Complex) -> Complex:
    # You can extract elements from a tuple like this
    a = x[0]
    b = x[1]

    c = y[0]
    d = y[1]

    # This creates a new variable and stores the real component into it
    real = a + c
    # Replace the ... with code to calculate the imaginary component
    imaginary = b + d

    # You can create a tuple like this
    ans = (real, imaginary)

    return ans
```

```
Success!
```

Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

2.2.2 Exercise 3: Complex multiplication.

Inputs:

1. A complex number $x = a + bi$, represented as a tuple `(a, b)`.
2. A complex number $y = c + di$, represented as a tuple `(c, d)`.

Goal: Return the product of these two numbers $x \cdot y = z = g + hi$, represented as a tuple `(g, h)`.

Need a hint? Click here

Remember, multiplying complex numbers is just like multiplying polynomials. Distribute one of the complex numbers:

$$(a + bi)(c + di) = a(c + di) + bi(c + di)$$

Then multiply through, and group the real and imaginary terms together. A video explanation can be found here.

[8]:
```
@exercise
def complex_mult(x : Complex, y : Complex) -> Complex:
    # Fill in your own code
    (a, b) = x
    (c, d) = y
    real = (a*c)-(b*d)
    imaginary = (a*d)+(c*b)

    return (real, imaginary)
```

```
Success!
```

Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

2.3 Complex Conjugate

Before we discuss any other complex operations, we have to cover the **complex conjugate**. The conjugate is a simple operation: given a complex number $x = a + bi$, its complex conjugate is $\overline{x} = a - bi$.

The conjugate allows us to do some interesting things. The first and probably most important is multiplying a complex number by its conjugate:

$$x \cdot \overline{x} = (a + bi)(a - bi)$$

Notice that the second expression is a difference of squares:

$$(a + bi)(a - bi) = a^2 - (bi)^2 = a^2 - b^2 i^2 = a^2 + b^2$$

This means that a complex number multiplied by its conjugate always produces a non-negative real number.

Another property of the conjugate is that it distributes over both complex addition and complex multiplication:

$$\overline{x + y} = \overline{x} + \overline{yx \cdot y} = \overline{x} \cdot \overline{y}$$

2.3.1 Exercise 4: Complex conjugate.

Input: A complex number $x = a + bi$, represented as a tuple `(a, b)`.

Goal: Return $\overline{x} = g + hi$, the complex conjugate of x, represented as a tuple `(g, h)`.

Need a hint? Click here

A video explanation can be found here.

[9]:
```
@exercise
def conjugate(x : Complex) -> Complex:
    real = x[0]

    imaginary = - x[1]

    return (real, imaginary)
```

```
Success!
```

Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

2.4 Complex Division

The next use for the conjugate is complex division. Let's take two complex numbers: $x = a + bi$ and $y = c + di \neq 0$ (not even complex numbers let you divide by 0). What does $\frac{x}{y}$ mean?

Let's expand x and y into their component forms:

$$\frac{x}{y} = \frac{a + bi}{c + di}$$

Unfortunately, it isn't very clear what it means to divide by a complex number. We need some way to move either all real parts or all imaginary parts into the numerator. And thanks to the conjugate, we can do just that. Using the fact that any number (except 0) divided by itself equals 1, and any number multiplied by 1 equals itself, we get:

$$\frac{x}{y} = \frac{x}{y} \cdot 1 = \frac{x}{y} \cdot \frac{\overline{y}}{\overline{y}} = \frac{x\overline{y}}{y\overline{y}} = \frac{(a + bi)(c - di)}{(c + di)(c - di)} = \frac{(a + bi)(c - di)}{c^2 + d^2}$$

By doing this, we re-wrote our division problem to have a complex multiplication expression in the numerator, and a real number in the denominator. We already know how to multiply complex numbers, and dividing a complex number by a real number is as simple as dividing both parts of the complex number separately:

$$\frac{a + bi}{r} = \frac{a}{r} + \frac{b}{r}i$$

2.4.1 Exercise 5: Complex division.

Inputs:

1. A complex number $x = a + bi$, represented as a tuple `(a, b)`.
2. A complex number $y = c + di \neq 0$, represented as a tuple `(c, d)`.

Goal: Return the result of the division $\frac{x}{y} = \frac{a+bi}{c+di} = g + hi$, represented as a tuple `(g, h)`.

Need a hint? Click here

A video explanation can be found here.

[10]:
```
@exercise
def complex_div(x : Complex, y : Complex) -> Complex:
    (a, b) = x
    (c, d) = y

    denominator = (c ** 2) + (d ** 2)

    real = ((a * c) + (b * d)) / denominator
    imaginary = ((a * (-d) ) + (c * b)) / denominator

    return (real, imaginary)
```

```
Success!
```

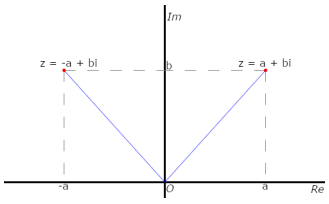
Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

3 Geometric Perspective

3.1 The Complex Plane

You may recall that real numbers can be represented geometrically using the number line - a line on which each point represents a real number. We can extend this representation to include imaginary and complex numbers, which gives rise to an entirely different number line: the imaginary number line, which only intersects with the real number line at 0.

A complex number has two components - a real component and an imaginary component. As you no doubt noticed from the exercises, these can be represented by two real numbers - the real component, and the real coefficient of the imaginary component. This allows us to map complex numbers onto a two-dimensional plane - the **complex plane**. The most common mapping is the obvious one: $a + bi$ can be represented by the point (a, b) in the **Cartesian coordinate system**.



This mapping allows us to apply complex arithmetic to geometry, and, more importantly, apply geometric concepts to complex numbers. Many properties of complex numbers become easier to understand when viewed through a geometric lens.

3.2 Modulus

One such property is the **modulus** operator. This operator generalizes the **absolute value** operator on real numbers to the complex plane. Just like the absolute value of a number is its distance from 0, the modulus of a complex number is its distance from $0 + 0i$. Using the distance formula, if $x = a + bi$, then:

$$|x| = \sqrt{a^2 + b^2}$$

There is also a slightly different, but algebraically equivalent definition:

$$|x| = \sqrt{x \cdot \overline{x}}$$

Like the conjugate, the modulus distributes over multiplication.

$$|x \cdot y| = |x| \cdot |y|$$

Unlike the conjugate, however, the modulus doesn't distribute over addition. Instead, the interaction of the two comes from the triangle inequality:

$$|x + y| \leq |x| + |y|$$

3.2.1 Exercise 6: Modulus.

Input: A complex number $x = a + bi$, represented as a tuple `(a, b)`.

Goal: Return the modulus of this number, $|x|$.

> Python's exponentiation operator is `**`, so 2^3 is `2 ** 3` in Python.
>
> You will probably need some mathematical functions to solve the next few tasks. They are available in Python's math library. You can find the full list and detailed information in the official documentation.

Need a hint? Click here

In particular, you might be interested in Python's square root function. A video explanation can be found here.

[13]:
```
@exercise
def modulus(x : Complex) -> float:
    return math.sqrt(x[0] ** 2 + x[1] ** 2)
```

```
Success!
```

Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

3.3 Imaginary Exponents

The next complex operation we're going to need is exponentiation. Raising an imaginary number to an integer power is a fairly simple task, but raising a number to an imaginary power, or raising an imaginary (or complex) number to a real power isn't quite as simple.

Let's start with raising real numbers to imaginary powers. Specifically, let's start with a rather special real number - Euler's constant, e:

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

(Here and later in this tutorial θ is measured in radians.)

Explaining why that happens is somewhat beyond the scope of this tutorial, as it requires some calculus, so we won't do that here. If you are curious, you can see this video for a beautiful intuitive explanation, or the Wikipedia article for a more mathematically rigorous proof.

Here are some examples of this formula in action:

$$e^{i\pi/4} = \frac{1}{\sqrt{2}} + \frac{i}{\sqrt{2}}e^{i\pi/2} = ie^{i\pi} = -1e^{2i\pi} = 1$$

One interesting consequence of this is Euler's Identity:

$$e^{i\pi} + 1 = 0$$

While this doesn't have any notable uses, it is still an interesting identity to consider, as it combines 5 fundamental constants of algebra into one expression.

We can also calculate complex powers of e as follows:

$$e^{a+bi} = e^a \cdot e^{bi}$$

Finally, using logarithms to express the base of the exponent as $r = e^{\ln r}$, we can use this to find complex powers of any positive real number.

3.3.1 Exercise 7: Complex exponents.

Input: A complex number $x = a + bi$, represented as a tuple `(a, b)`.

Goal: Return the complex number $e^x = e^{a+bi} = g + hi$, represented as a tuple `(g, h)`.

Euler's constant e is available in the math library, as are Python's trigonometric functions.

[14]:
```
@exercise
def complex_exp(x : Complex) -> Complex:
    (a, b) = x

    expa = math.e ** a
```

```
    real = expa * math.cos(b)
    imaginary = expa * math.sin(b)

    return (real, imaginary)
```

```
Success!
```

Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

3.3.2 Exercise 8*: Complex powers of real numbers.

Inputs:

1. A non-negative real number r.
2. A complex number $x = a + bi$, represented as a tuple `(a, b)`.

Goal: Return the complex number $r^x = r^{a+bi} = g + hi$, represented as a tuple `(g, h)`.

Remember, you can use functions you have defined previously

Need a hint? Click here

You can use the fact that $r = e^{\ln r}$ to convert exponent bases. Remember though, $\ln r$ is only defined for positive numbers - make sure to check for $r = 0$ separately!

[15]:
```
@exercise
def complex_exp_real(r : float, x : Complex) -> Complex:
    if (r == 0):
        return (0,0)

    (a, b) = x

    # Raise r to the power of a
    ra = r ** a

    # Natural logarithm of r
    lnr = math.log(r)

    real = ra * math.cos(b * lnr)
    imaginary = ra * math.sin(b * lnr)

    return (real, imaginary)
```

```
Success!
```

Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

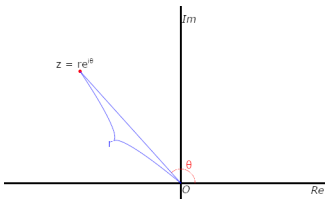
3.4 Polar coordinates

Consider the expression $e^{i\theta} = \cos\theta + i\sin\theta$. Notice that if we map this number onto the complex plane, it will land on a **unit circle** around $0 + 0i$. This means that its modulus is always 1. You can also verify this algebraically: $\cos^2\theta + \sin^2\theta = 1$.

Using this fact we can represent complex numbers using **polar coordinates**. In a polar coordinate system, a point is represented by two numbers: its direction from origin, represented by an angle from the x axis, and how far away it is in that direction.

Another way to think about this is that we're taking a point that is 1 unit away (which is on the unit circle) in the specified direction, and multiplying it by the desired distance. And to get the point on the unit circle, we can use $e^{i\theta}$.

A complex number of the format $r \cdot e^{i\theta}$ will be represented by a point which is r units away from the origin, in the direction specified by the angle θ.



Sometimes θ will be referred to as the number's **phase**.

3.4.1 Exercise 9: Cartesian to polar conversion.

Input: A complex number $x = a + bi$, represented as a tuple `(a, b)`.

Goal: Return the polar representation of $x = re^{i\theta}$, i.e., the distance from origin r and phase θ as a tuple `(r, )`.

- r should be non-negative: $r \geq 0$
- θ should be between $-\pi$ and π: $-\pi < \theta \leq \pi$

Need a hint? Click here

Python has a separate function for calculating θ for this purpose. A video explanation can be found here.

[16]:
```
@exercise
def polar_convert(x : Complex) -> Polar:
    (a, b) = x

    r = math.sqrt(a**2 + b **2)
    theta = math.atan2(b, a)

    return (r, theta)
```

```
Success!
```

Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

3.4.2 Exercise 10: Polar to Cartesian conversion.

Input: A complex number $x = re^{i\theta}$, represented in polar form as a tuple `(r, )`.

Goal: Return the Cartesian representation of $x = a + bi$, represented as a tuple `(a, b)`.

[17]:
```
@exercise
def cartesian_convert(x : Polar) -> Complex:
    (r, theta) = x

    real = r * math.cos(theta)
    imaginary = r * math.sin(theta)

    return (real, imaginary)
```

```
Success!
```

Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

3.4.3 Exercise 11: Polar multiplication.

Inputs:

1. A complex number $x = r_1 e^{i\theta_1}$ represented in polar form as a tuple `(r1, 1)`.
2. A complex number $y = r_2 e^{i\theta_2}$ represented in polar form as a tuple `(r2, 2)`.

Goal: Return the result of the multiplication $x \cdot y = z = r_3 e^{i\theta_3}$, represented in polar form as a tuple `(r3, 3)`.

- r_3 should be non-negative: $r_3 \geq 0$
- θ_3 should be between $-\pi$ and π: $-\pi < \theta_3 \leq \pi$
- Try to avoid converting the numbers into Cartesian form.

Need a hint? Click here

Remember, a number written in polar form already involves multiplication. What is $r_1 e^{i\theta_1} \cdot r_2 e^{i\theta_2}$?

Need another hint? Click here

Is your not coming out correctly? Remember you might have to check your boundaries and adjust it to be in the range requested.

[18]:
```
@exercise
def polar_mult(x : Polar, y : Polar) -> Polar:
    (r1, theta1) = x
    (r2, theta2) = y

    radius = r1 * r2

    angle = theta1 + theta2

    if (angle > math.pi):
        angle = angle - 2.0 * math.pi

    elif (angle <= -math.pi):
        angle = angle + 2.0 * math.pi

    return (radius, angle)
```

```
Success!
```

Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

3.4.4 Exercise 12**: Arbitrary complex exponents.

You now know enough about complex numbers to figure out how to raise a complex number to a complex power.

Inputs:

1. A complex number $x = a + bi$, represented as a tuple `(a, b)`.
2. A complex number $y = c + di$, represented as a tuple `(c, d)`.

Goal: Return the result of raising x to the power of y: $x^y = (a+bi)^{c+di} = z = g + hi$, represented as a tuple `(g, h)`.

Need a hint? Click here

Convert x to polar form, and raise the result to the power of y.

[19]:
```
@exercise
def complex_exp_arbitrary(x : Complex, y : Complex) -> Complex:
    (a, b) = x
    (c, d) = y

    r = math.sqrt(a ** 2 + b ** 2)
    theta = math.atan2(b, a)

    if (r == 0):
        return (0, 0)

    lnr = math.log(r)

    exponent = math.exp(lnr * c - d * theta)

    real = exponent * (math.cos(lnr * d + theta * c ))
    imaginary = exponent * (math.sin(lnr * d + theta * c))

    return (real, imaginary)
```

```
Success!
```

Can't come up with a solution? See the explained solution in the Complex Arithmetic Workbook.

3.5 Conclusion

Congratulations! You should now know enough complex arithmetic to get started with quantum computing. When you are ready, you can move on to the next tutorial in this series, covering linear algebra.