Euler's formula

From Wikipedia, the free encyclopedia

* Elementary Number

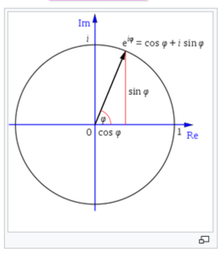
  $0, 1, i, e, \pi$

* Elementary Function

Function Type	Differentiation	Inverse	Differentiation
Power: $x^2,\ x^n$	$2x^1, nx^{n-1}$	Root: $\sqrt{x}$	$\frac{1}{2\sqrt{x}}$
Exponential: e^x	e^x	Log: $\ln x$	$\frac{1}{x}$
Trigonometric: $\cos(x), \sin(x)$	$-\sin(x), \cos(x)$	Inverse Trigonometric: $acos(x), asin(x)$	...

* Euler's Formula

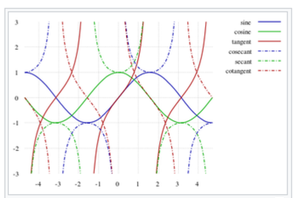
$$e^{i\theta} = \cos(\theta) + i\sin(\theta)$$
$$e^{i\pi} = \cos(\pi) + i\sin(\pi)$$
$$e^{i\pi} + 1 = 0$$



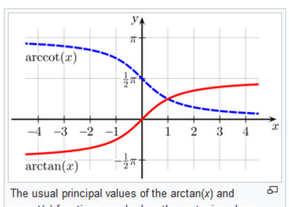
Complex number for rotation

$$e^{i\theta} e^{iy} = e^{i(\theta+y)}$$
$$\cos(\theta+y) + i\sin(\theta+y)$$

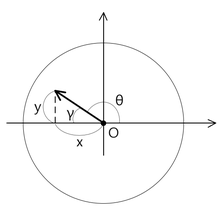
tan, atan, tan2



Trigonometric functions: Sine, Cosine, Tangent, Cosecant (dotted), Secant (dotted), Cotangent (dotted) – animation



The usual principal values of the arctan(x) and arccot(x) functions graphed on the cartesian plane.



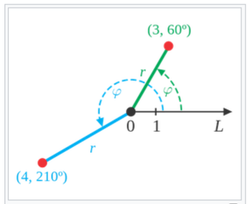
where atan2 is a common variation on the arctangent function defined as

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

Polar coordinate system

From Wikipedia, the free encyclopedia
(Redirected from Polar coordinate)

In mathematics, the **polar coordinate system** is a two-dimensional coordinate system in which each point on a plane is determined by a distance from a reference point and an angle from a reference direction. The reference point (analogous to the origin of a Cartesian coordinate system) is called the *pole*, and the ray from the pole in the reference direction is the *polar axis*. The distance from the pole is called the *radial coordinate*, *radial distance* or simply *radius*, and the angle is called the *angular coordinate*, *polar angle*, or *azimuth*.[1] The radial coordinate is often denoted by *r* or *ρ*, and the angular coordinate by *φ*, *θ*, or *t*. Angles in polar notation are generally expressed in either degrees or radians (2π rad being equal to 360°).



Points in the polar coordinate system with pole *O* and polar axis *L*. In green, the point with radial coordinate 3 and angular coordinate 60 degrees or (3, 60°). In blue, the point (4, 210°).

Converting between polar and Cartesian coordinates [edit]

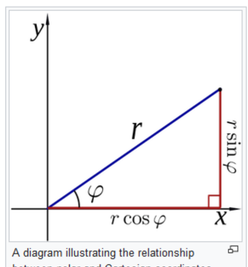
The polar coordinates *r* and *φ* can be converted to the Cartesian coordinates *x* and *y* by using the trigonometric functions sine and cosine:

$$x = r\cos\varphi,$$
$$y = r\sin\varphi.$$

The Cartesian coordinates *x* and *y* can be converted to polar coordinates *r* and *φ* with $r \geq 0$ and *φ* in the interval $(-\pi, \pi]$ by:[13]

$r = \sqrt{x^2 + y^2}$ (as in the Pythagorean theorem or the Euclidean norm), and

$\varphi = \operatorname{atan2}(y, x)$,



A diagram illustrating the relationship between polar and Cartesian coordinates.

Complex numbers [edit]

Every complex number can be represented as a point in the complex plane, and can therefore be expressed by specifying either the point's Cartesian coordinates (called rectangular or Cartesian form) or the point's polar coordinates (called polar form). The complex number *z* can be represented in rectangular form as

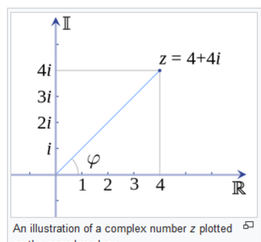
$$z = x + iy$$

where *i* is the imaginary unit, or can alternatively be written in polar form (via the conversion formulae given above) as

$$z = r(\cos\varphi + i\sin\varphi)$$

and from there as

$$z = re^{i\varphi}$$



An illustration of a complex number *z* plotted on the complex plane

Complex number

From Wikipedia, the free encyclopedia

Addition and subtraction [edit]

Two complex numbers a and b are most easily added by separately adding their real and imaginary parts of the summands. That is to say:

$$a + b = (x + yi) + (u + vi) = (x + u) + (y + v)i.$$

Similarly, subtraction can be performed as

$$a - b = (x + yi) - (u + vi) = (x - u) + (y - v)i.$$

Multiplication [edit]

Since the real part, the imaginary part, and the indeterminate i in a complex number are all considered as numbers in themselves, two complex numbers, given as $z = x + yi$ and $w = u + vi$ are multiplied under the rules of the distributive property, the commutative properties and the defining property $i^2 = -1$ in the following way

$$\begin{aligned} z \cdot w &= (x + yi) \cdot (u + vi) \\ &= x(u + vi) + yi(u + vi) && \text{by the (right) distributive law} \\ &= xu + xvi + yiu + yivi && \text{by the (left) distributive law} \\ &= xu + yivi + xvi + yiu && \text{by the commutativity of addition} \\ &= xu + yvi^2 + xvi + yui && \text{by the commutativity of multiplication} \\ &= (xu + yvi^2) + (xvi + yui) && \text{by the associativity of addition} \\ &= (xu - yv) + (xvi + yui) && \text{by the defining property of } i \\ &= (xu - yv) + (xv + yu)i && \text{by the distributive law.} \end{aligned}$$

std::complex

Defined in header <complex>

```
template< class T >
class complex;                          (1)
template<> class complex<float>;        (2)
template<> class complex<double>;       (3)
template<> class complex<long double>;  (4)
```

Member functions

(constructor)	constructs a complex number (public member function)
operator=	assigns the contents (public member function)
real	accesses the real part of the complex number (public member function)
imag	accesses the imaginary part of the complex number (public member function)
operator+= operator-=	compound assignment of two complex numbers or a complex and a scalar
operator*= operator/=	(public member function)

Non-member functions

operator+ operator-	applies unary operators to complex numbers (function template)
operator+ operator- operator* operator/	performs complex number arithmetics on two complex values or a complex and a scalar (function template)

std::arg(std::complex)

Defined in header <complex>

```
template< class T >
T arg( const complex<T>& z );
```

```
std::complex<double> z1(1, 0);
std::cout << "phase angle of " << z1 << " is " << std::arg(z1) << '\n';
```

std::polar(std::complex)

Defined in header <complex>

```
template< class T >
complex<T> polar( const T& r, const T& theta = T());
```

Returns a complex number with magnitude r and phase angle theta.
The behavior is undefined if r is negative or NaN, or if theta is infinite.

std::exp(std::complex)

Defined in header <complex>

```
template< class T >
complex<T> exp( const complex<T>& z );
```

Compute base-e exponential of z, that is *e* (Euler's number, 2.7182818) raised to the z power.

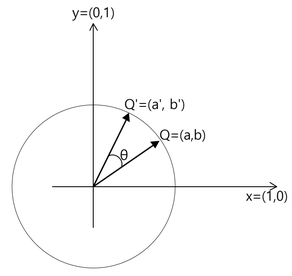
```
#include <complex>
#include <iostream>

int main()
{
   const double pi = std::acos(-1);
   const std::complex<double> i(0, 1);

   std::cout << std::fixed << " exp(i*pi) = " << std::exp(i * pi) << '\n';
}
```

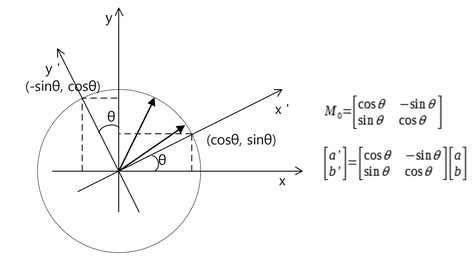
Output:

```
exp(i*pi) = (-1.000000,0.000000)
```

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

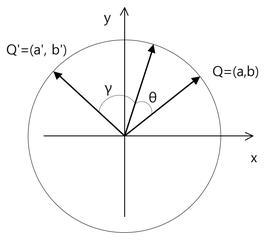
Rotation Matrix





$$M_0 = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\begin{bmatrix} a' \\ b' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$



$$M_0 = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$M_1 = \begin{bmatrix} \cos y & -\sin y \\ \sin y & \cos y \end{bmatrix}$$

$$M_1 M_0 \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \cos y & -\sin y \\ \sin y & \cos y \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

Complex Number for Rotation

$$e^{i\theta} = \cos\theta + i\sin\theta$$
$$e^{iy} = \cos y + i\sin y$$
$$e^{i\theta} e^{iy} = \cos(\theta+y) + i\sin(\theta+y)$$

Implementation

```
static double timer = 0;
timer += (double)fElapsedTime_;
const std::complex<double> i( 0, 1 );
std::complex<double>    c0;
c0 = std::polar<double>( 1.0, M_PI / 4.0 );

{
    std::complex<double>    c1;
    c1 = std::polar<double>( 1.0, timer );
    double theta = std::arg( c0 * c1 );

    KMatrix2 m;
    m.SetRotation( theta );
    KVector2 v( 1, 0 );
    v = m * v;
    KVectorUtil::DrawLine( hdc, KVector2::zero, v, 2, PS_SOLID, RGB( 255, 0, 0 ) );
}
```