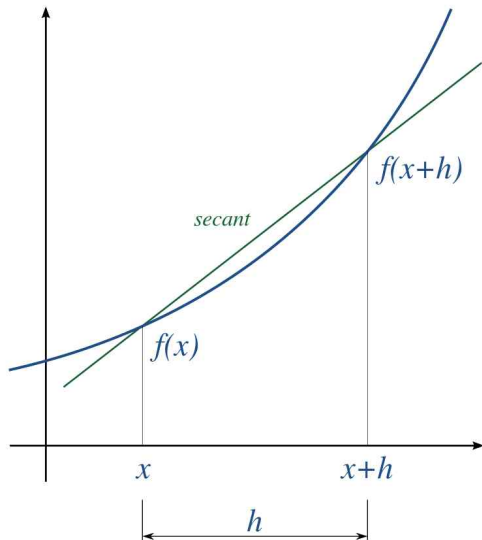


---

## Numerical Differentiation



**Newton's difference quotient(= first-order divided difference)**

$$\frac{f(x+h) - f(x)}{h}$$

**derivative of  $f$  at  $x$**

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

## Symmetric difference quotient

$$\frac{f(x+h) - f(x-h)}{2h}$$

## Order of Accuracy

The numerical solution  $u_h$  is said to be **nth-order accurate** if the error,

$E(h) := \|u - u_h\|$  is proportional to the step-size **h** to the **n-th** power

$$E(h) = \|u - u_h\| \leq Ch^n$$

## Implementation

```
double NewtonsDifference( FUNCTION f, double x, double dx = 0.0001 )
{
    const double y0 = f( x );
    const double y1 = f( x + dx );
    return ( y1 - y0 ) / dx;
}
```

```
double SymmetricDifference( FUNCTION f, double x, double dx = 0.0001 )
{
    const double y0 = f( x - dx );
    const double y1 = f( x + dx );
    return ( y1 - y0 ) / (2.0*dx);
}
```

# Drawing

## Drawing a Single Variable Function

**using** FUNCTION = **double**(\*)( **double** x );

```
void DrawFunction(HDC hdc, FUNCTION Callback, double beginX, double endX,
double xstep, COLORREF color)
{
    double oldX;
    double oldY;
    double x = beginX;
    double y = Callback( x );
    oldX = x;
    oldY = y;
    while (x < endX)
    {
        x += xstep;
        y = Callback( x );
        KVectorUtil::DrawLine(hdc, KVector2(oldX, oldY), KVector2(x, y), 2, PS_SOLID,
color);
        oldX = x;
        oldY = y;
    } //while
}
```

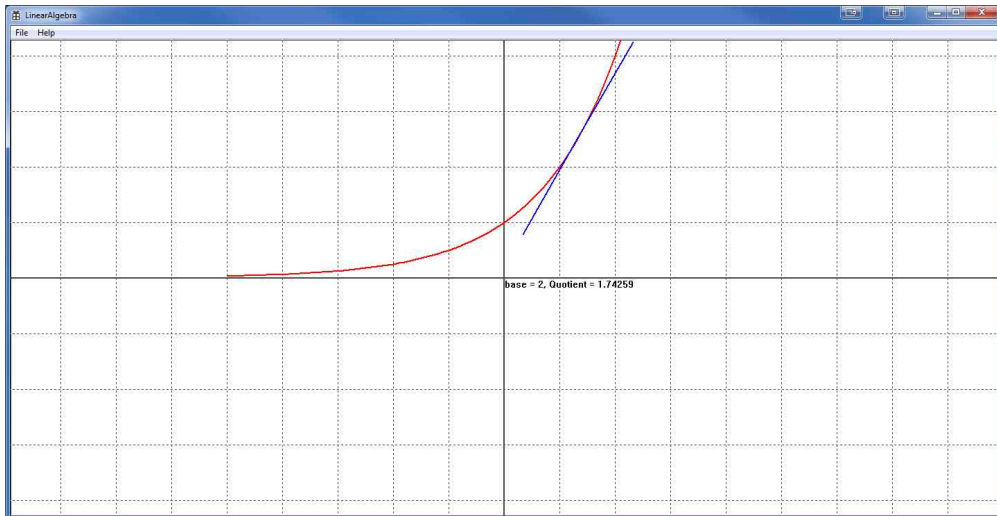
## Drawing a Tangent Line Segment

```
const double dx = 0.001;
double y = Function( x );
double diff = SymmetricDifference( &Function, x, dx );
KVector2 v0 = KVector2( x, y );
KVector2 vdir = KVector2( dx, diff*dx );
vdir.Normalize( );

KVectorUtil::DrawLine( hdc, v0, v0 + vdir * 2.0, 2, PS_SOLID, RGB( 0, 0, 255 )
);
KVectorUtil::DrawLine( hdc, v0, v0 + vdir * -2.0, 2, PS_SOLID, RGB( 0, 0, 255 )
```

);

## Result



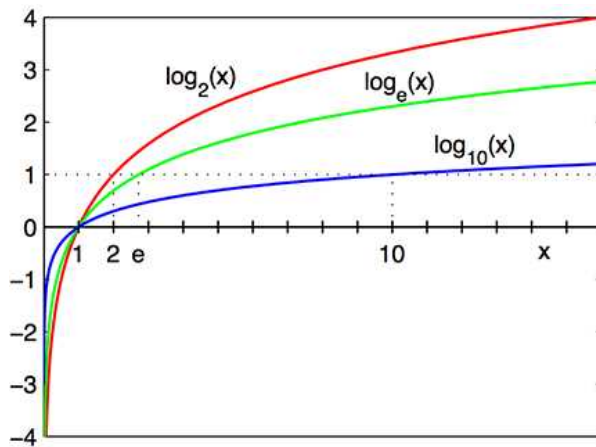
[Fig] Tangent line segment at x

---

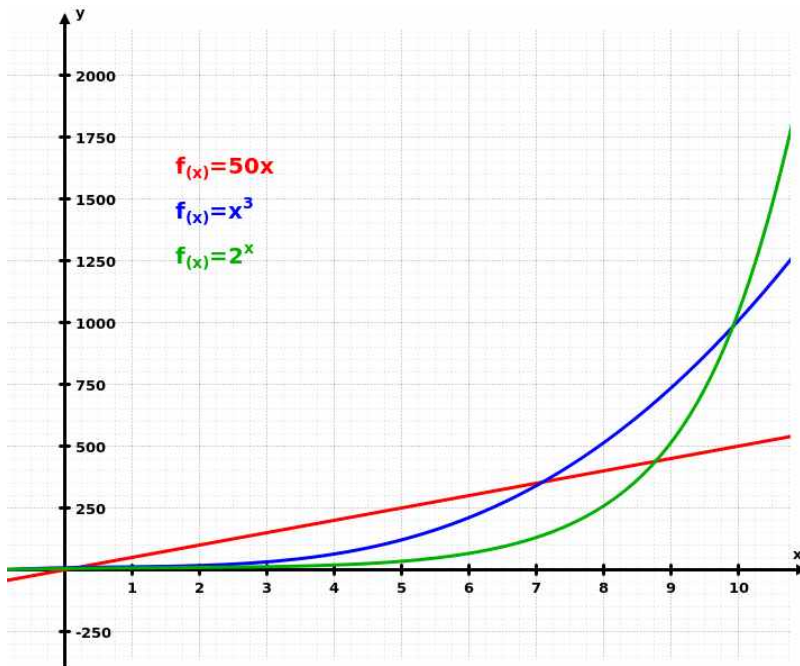
# Euler Constant

## Logarithm

$$\log_b(x) = y, \quad b^y = x$$



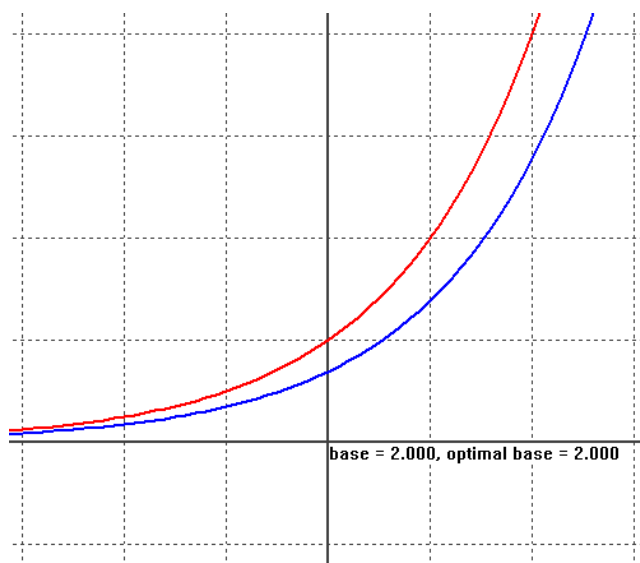
## Why Exponential



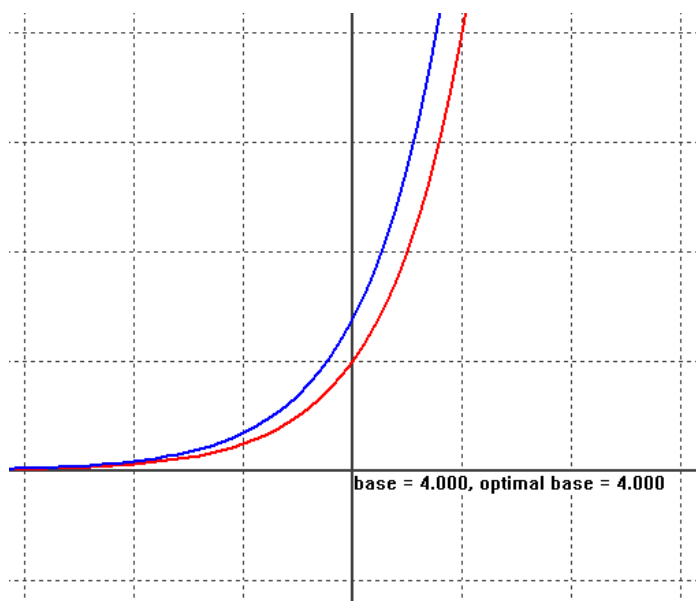
Exponential growth (green) describes many physical phenomena.

## Differentiation of exponential function

$$\begin{aligned}
 \frac{d}{dx}a^x &= \lim_{h \rightarrow 0} \frac{a^{x+h} - a^x}{h} \\
 &= \lim_{h \rightarrow 0} \frac{a^x a^h - a^x}{h} \\
 &= a^x \left( \lim_{h \rightarrow 0} \frac{a^h - 1}{h} \right)
 \end{aligned}$$



[Fig]  $2^x, \frac{d}{dx}2^x$



[Fig]  $4^x, \frac{d}{dx}4^x$

$2 < \textit{MysteriousIrrationalConstant } e < 4$

## Numerical solution

### Implementation

$$s = \sqrt{\frac{\sum_{i=1}^N (x_i - \bar{x})^2}{N-1}}$$

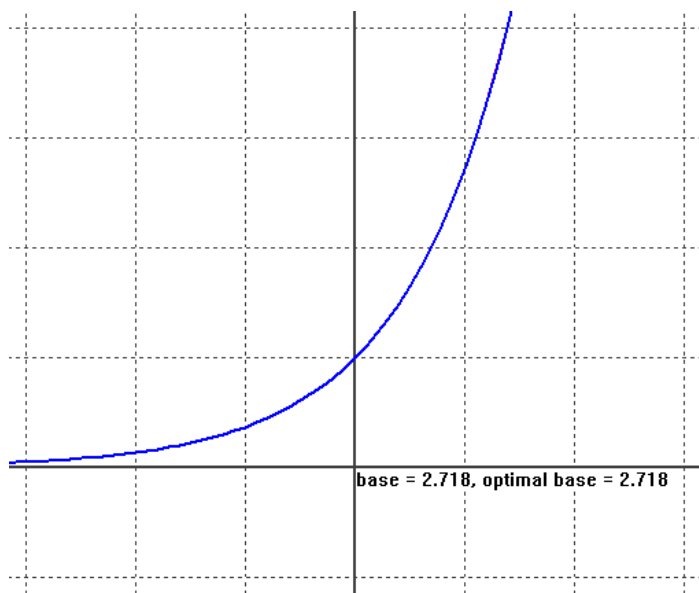
```
double GetStdDeviation( double base_, double beginX, double endX, double xstep
)
{
    std::vector<double>    vecDiff;

    double x = beginX;
    double ydiff;
    double N = 0;
    while(x < endX)
    {
        ydiff = ExpFunction( base_, x ) - SymmetricDifference( &ExpFunction,
base_, x );
        x += xstep;
        N += 1;
        vecDiff.push_back( ydiff );
    }//while

    double sum = 0;
    for(const double diff : vecDiff)
    {
        sum += ( diff * diff );
    }

    return sqrt( sum / ( N - 1 ) );
}
```





$$e \approx 2.718$$

## Why e

$$\frac{d}{dx} e^x = e^x$$

$$a^x = e^{cx}$$

$$2^x = e^{x \ln(2)}$$

$$3^x = e^{x \ln(3)}$$

...

$$\frac{d}{dx} e^{cx} = ce^{cx}$$

$$\frac{d}{dx} a^x = a^x \ln a$$

Complex Number

Quaternion

Fourier Transform

$$\frac{d}{dx} \log_e x = \frac{1}{x}$$

## Meaning of e

### Example

$$(1+1) = 2$$

$$(1+\frac{1}{2})(1+\frac{1}{2}) = 2.25$$

$$(1+\frac{1}{3})(1+\frac{1}{3})(1+\frac{1}{3}) = 2.370$$

$$(1+\frac{1}{4})(1+\frac{1}{4})(1+\frac{1}{4})(1+\frac{1}{4}) = 2.441$$

...

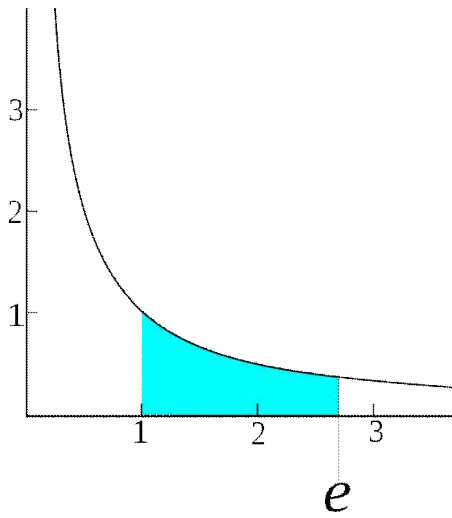
### Definition

$$e = \lim_{n \rightarrow \infty} (1 + \frac{1}{n})^n$$

$$e = \lim_{t \rightarrow \infty} (1 + t)^{\frac{1}{t}}$$

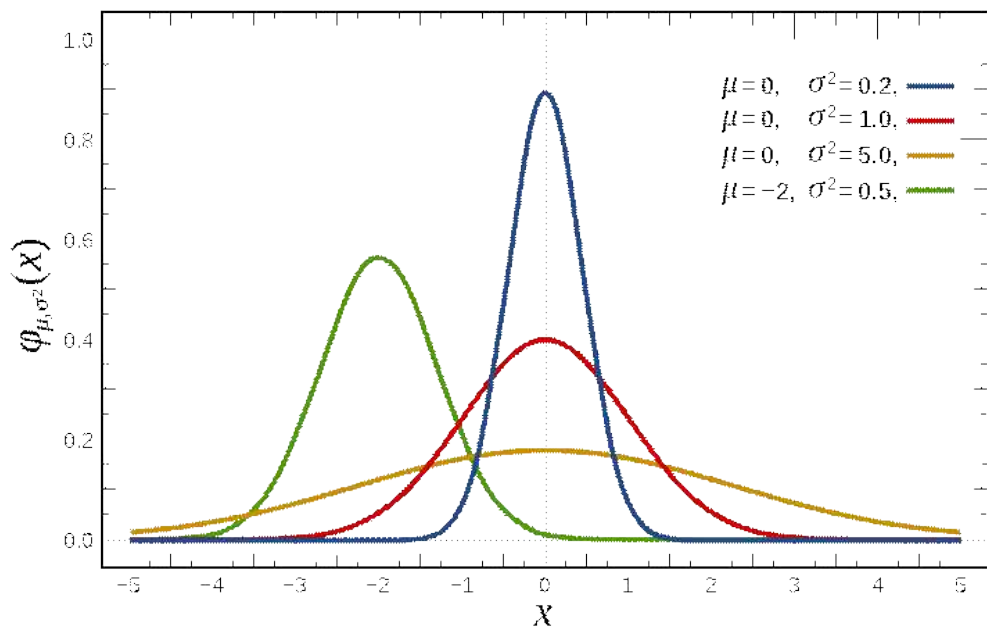
$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{1} + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \dots$$

$$\int_1^e \frac{1}{t} dt = 1$$



## Gaussian function

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}}$$



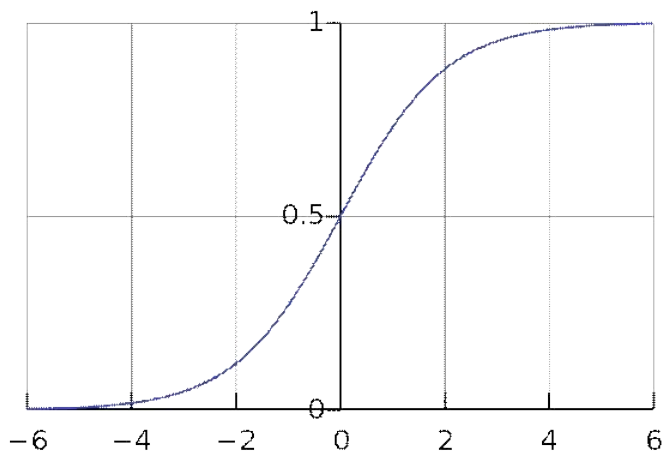
## Standard normal distribution

$$S(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

## Logistic Function

Sigmoid curve

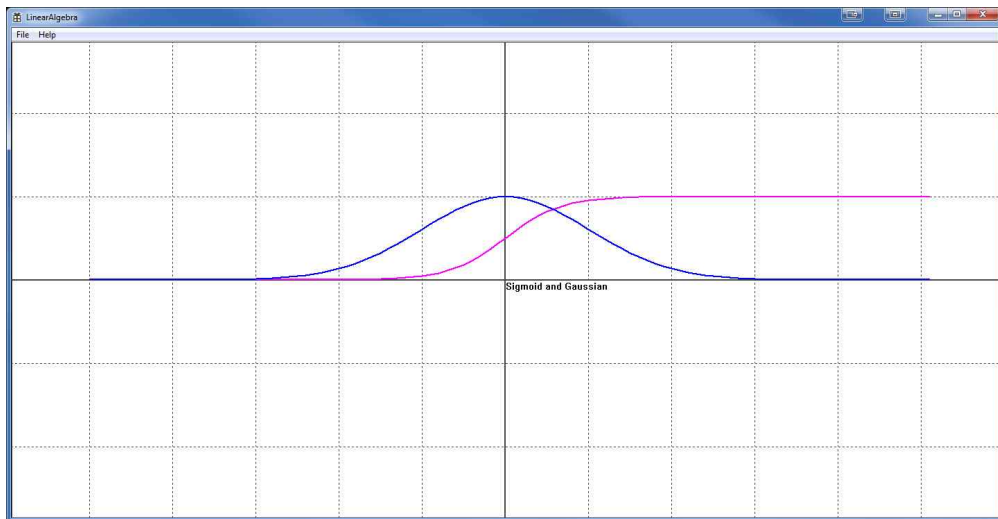
$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$



## Implementation

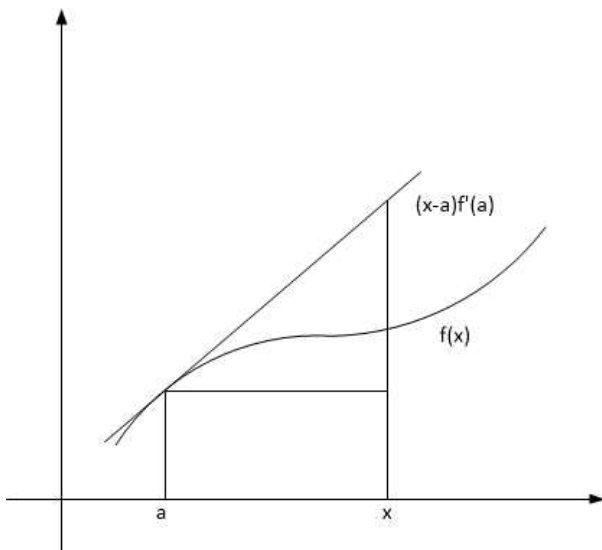
```
double Logistic( double x )
{
    // Logistic Function
    const double L = 1.0;
    const double k = 3.0;
    const double x0 = 0.0;
    return L / (1 + std::exp(-k*(x - x0)));
}

double Gaussian( double x )
{
    // Gaussian Function
    const double a = 1.0;
    const double b = 0.0;
    const double c = 1.0;
    return a*std::exp(-((x-b)*(x-b))/(2*c*c));
}
```



# Taylor's Formula

## Meaning



$$f(x) = f(a) + (x-a)f'(a)$$

$$f(x) = c_0 + c_1x + c_2x^2$$

$$f(x) = c_0 + c_1x + c_2x^2 + c_3x^3 + \dots$$

## Definition

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \dots$$

$$+ \frac{f^k(a)}{k!}(x-a)^k + h_k(x)(x-a)^k$$

$$\lim_{x \rightarrow a} h_k(x) = 0$$

Peano form of the remainder

$$\frac{f'''(a)}{3!}(x-a)^3$$

$$\frac{3 \cdot f'''(a)}{3!}(x-a)^2$$

$$\frac{3 \cdot 2 \cdot f'''(a)}{3!}(x-a)^1$$

$$\frac{3 \cdot 2 \cdot 1 \cdot f'''(a)}{3!}(x-a)$$

## Examples

$$\sin(0) = 0$$

$$\sin'(0) = \cos(0) = 1$$

$$\cos'(0) = -\sin(0) = 0$$

$$-\sin'(0) = -\cos(0) = -1$$

$$-\cos'(0) = -(-\sin(0)) = 0$$

$$\sin'(0) = \cos(0) = 1$$

...

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots$$

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

## Implementation



```
double Cosine(double x)
```

```
{
```

```
    return cos(x);
```

```
}
```

```
long long factorial(long long x, long long result = 1)
```

```
{
```

```
    if (x == 0)
```

```
        return result;
```

```
    else
```

```
        return factorial(x - 1, x * result);
```

```
}
```



```

long long numberOfTaylorSeriesTerms = 3;
double TaylorCosine(double x)
{
    double result = 0;
    for (int n = 0; n < numberOfTaylorSeriesTerms; ++n)
    {
        result += (std::pow(-1, n) / factorial(2 * n)) * std::pow(x, 2 * n);
    }
    return result;
    //return 1 - (x*x) / (2*1) + (x*x*x*x) / (4 * 3 * 2 * 1);
}

```

## Proof

$$f'(a) = \lim_{x \rightarrow a} \frac{f(x) - f(a)}{x - a}$$

$$f'(a) = \frac{f(x) - f(a)}{x - a}$$

$$f(x) = f(a) + (x - a)f'(a)$$

$$f(x) - f(a) = \int_a^x f'(\xi) d\xi$$

$$f(x) = f(a) + \int_a^x f'(\xi) d\xi$$

$$\begin{aligned}
 (f(x))' &= (f(a) + (x - a)f'(a))' \\
 f'(x) &= 0 + f'(a) + (x - a)f''(a) \\
 &= f'(a) + (x - a)f''(a)
 \end{aligned}$$

$$\begin{aligned}
 f(x) &= f(a) + \int_a^x \{f'(a) + (x-a)f''(a)\}d\xi \\
 &= f(a) + f'(a)[\xi]_a^x + f''(a)[\frac{1}{2}(\xi-a)^2]_a^x \\
 &= f(a) + (x-a)f'(a) + \frac{1}{2}(x-a)^2f''(a)
 \end{aligned}$$

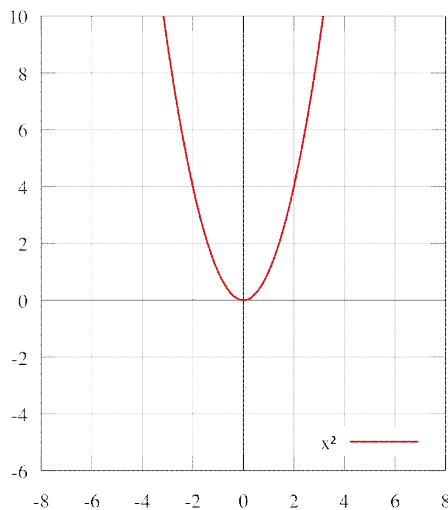
...

---

## Euler's Formula

### Even function

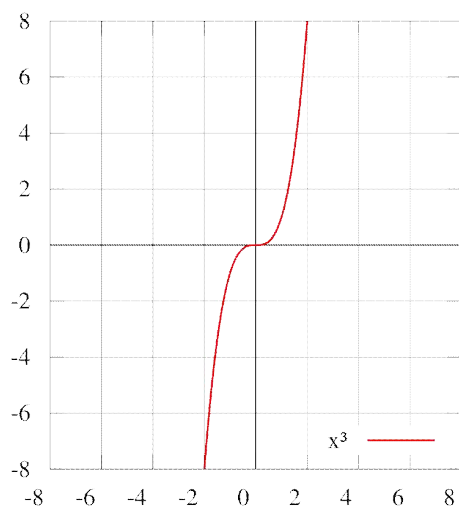
$$f(x) = f(-x)$$



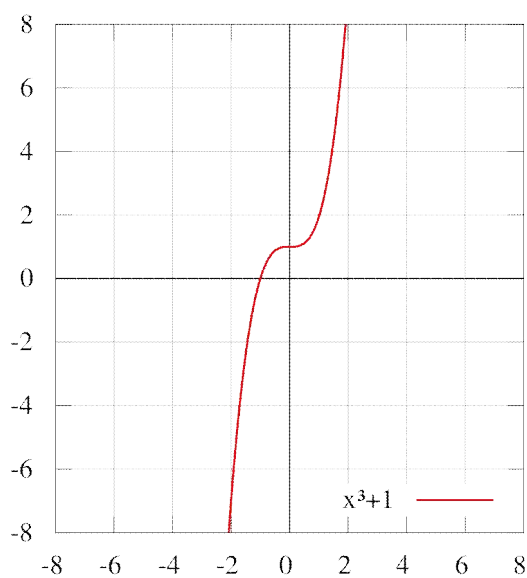
$$f(x) = x^2$$

### Odd function

$$-f(x) = f(-x)$$



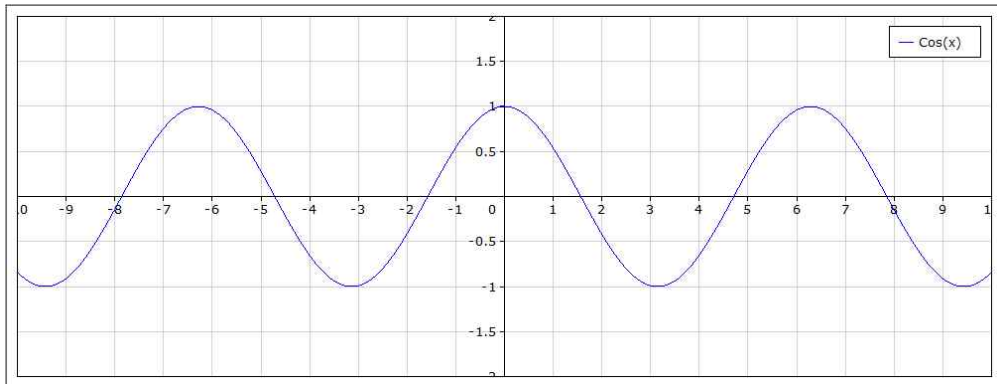
$$f(x) = x^3$$



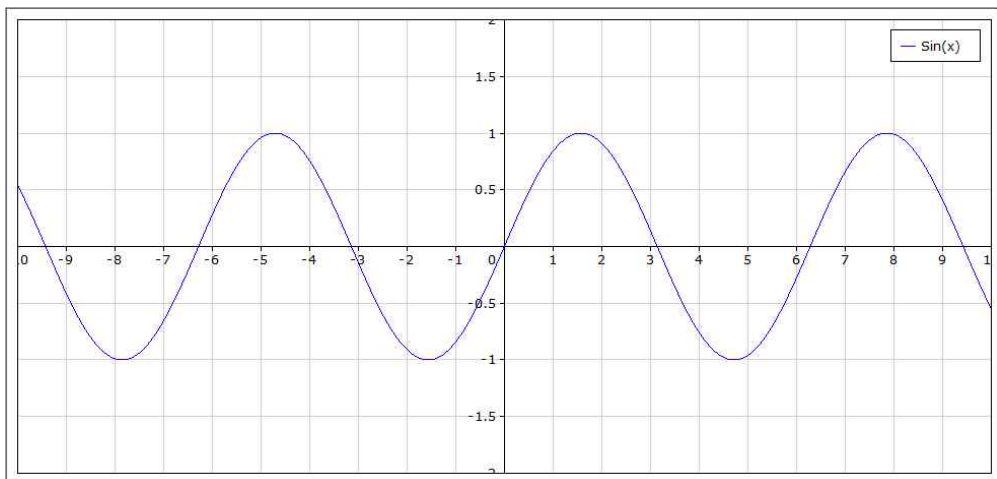
$$f(x) = x^3 + 1$$

not even or odd

## Cosine and Sine



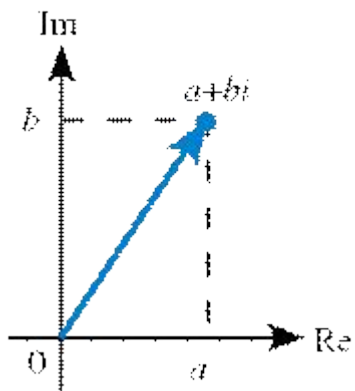
$\cos(x)$



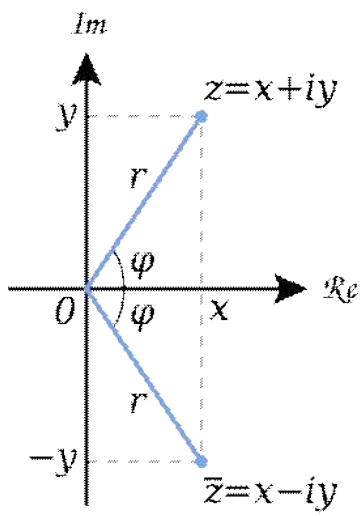
$\sin(x)$

## Complex number

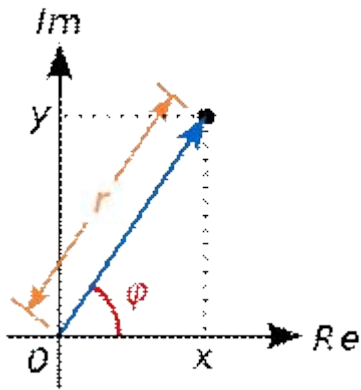
A complex number is a number that can be expressed in the form  $a + bi$ , where  $a$  and  $b$  are real numbers, and  $i$  is a solution of the equation  $x^2 = -1$ .



## Conjugate

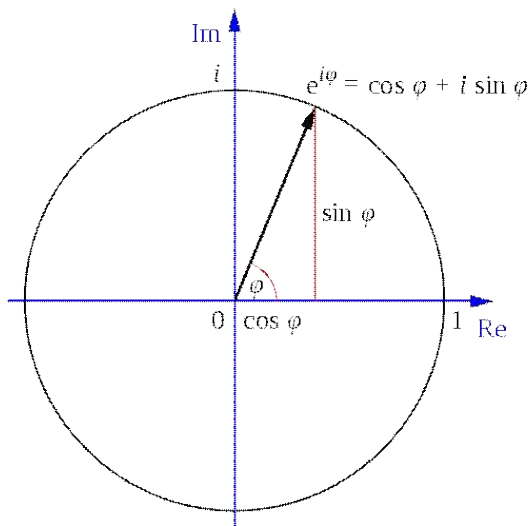


## Polar Coordinate



$$r(\cos(\theta) + i \sin(\theta))$$

## Euler's formula

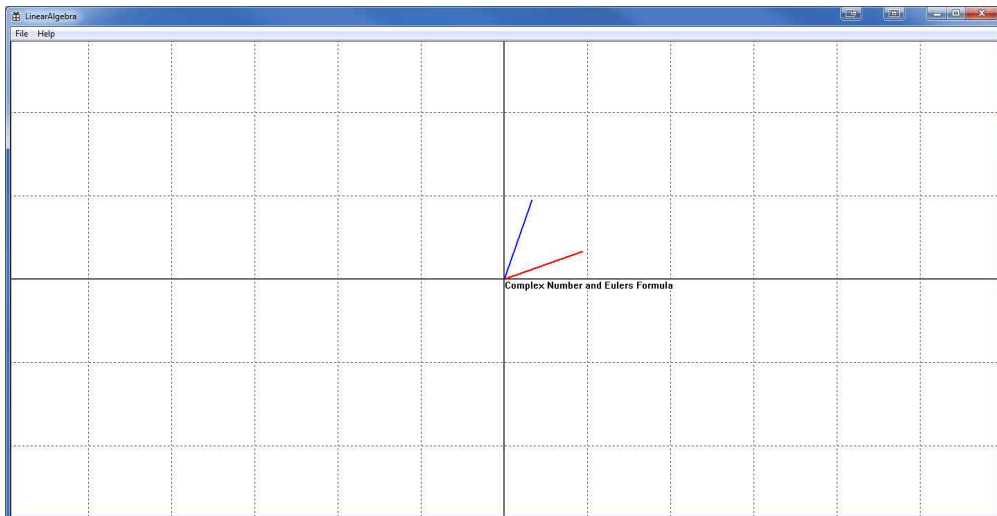


$$e^{ix} = \cos(x) + i \sin(x)$$

$$(\cos(x) + i \sin(x))(\cos(y) + i \sin(y)) = ?$$

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

## 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 ) );
}

{
    std::complex<double>    c2;
    c2 = std::exp( i * -timer );
    double theta = std::arg( c0 * c2 );

    KMatrix2 m;
```

```

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

```

## Conjugate

$$e^{-ix} = \cos(-x) + i \sin(-x) = \cos(x) - i \sin(x)$$

$$e^{ix} e^{-ix} = e^{i(x-x)} = e^{0i} = e^0 = 1$$

## Pythagorean theorem

$$\begin{aligned}
 e^{ix} e^{-ix} &= (\cos(x) + i \sin(x))(\cos(x) - i \sin(x)) \\
 &= \cos^2(x) - \cos(x)i \sin(x) + i \sin(x)\cos(x) - i^2 \sin^2(x) \\
 &= \cos^2(x) + \sin^2(x) = 1
 \end{aligned}$$

## Proof

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

$$\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots$$

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \dots$$

$$e^{ix} = \sum_{n=0}^{\infty} \frac{(ix)^n}{n!} = 1 + ix + \frac{(ix)^2}{2!} + \frac{(ix)^3}{3!} + \dots$$

$$e^{ix} = \cos(x) + i \sin(x)$$



## Quaternion

$$\begin{aligned}e^{ix} &= \cos(x) + i \sin(x) \\ &= \cos(x) + \frac{\sin(x)}{x}(ix)\end{aligned}$$

## Rotating Axis

$$\begin{aligned}\vec{v} &= (x, y, z) \\ (w, \vec{v}) &= (w, x, y, z) \\ \vec{v} &= xi + yj + zk = (x, y, z) \\ ijk &= -1 \\ ij &= k, ji = -k \\ jk &= i, kj = -i \\ ki &= j, ik = -j\end{aligned}$$

## Quaternion

$$\begin{aligned}e^{w+xi+yj+zk} &= e^w (\cos(|v|) + \frac{\sin(|v|)}{|v|}(xi + yj + zk)) \\ |v| &= \sqrt{x^2 + y^2 + z^2}\end{aligned}$$

$$e^{w+xi+0j+0k} = e^w (\cos(x) + \sin(x)(i))$$

## Rotating Idea

$$\begin{aligned}e^{(w+xi+yj+zk)} e^{(ai+bj+ck)} &= e^w e^{(x+a)i+(y+b)j+(z+c)k} \\ e^0 e^{(x+a)i+(y+b)j+(z+c)k} &= e^{(x+a)i+(y+b)j+(z+c)k}\end{aligned}$$

## Conjugate of Complex Number

$$c = a + bi$$

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

$$c^* = a - bi$$

$$c \cdot c^* = a^2 + b^2 = |c|^2$$

## Conjugate of Quaternion

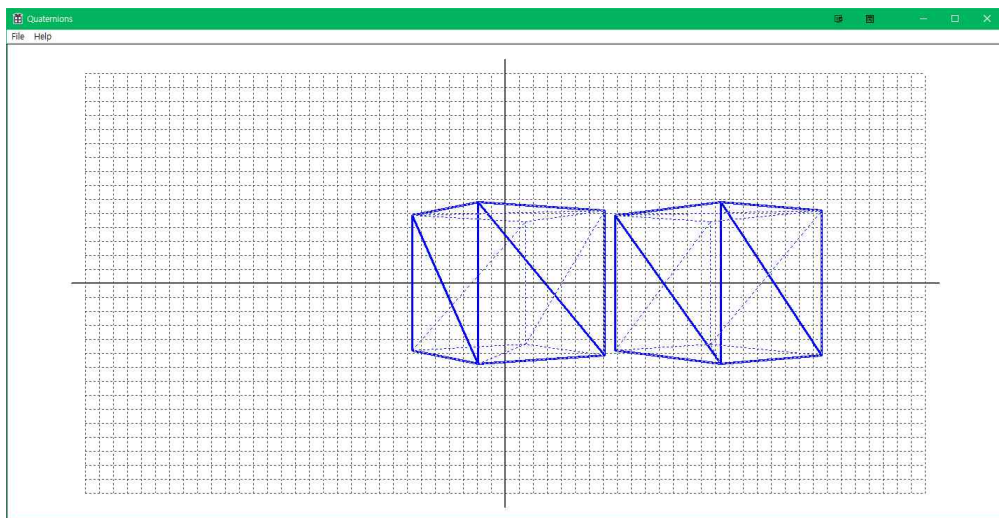
$$q = (w, x, y, z)$$

$$q^* = (w, -x, -y, -z)$$

$$|q| = \sqrt{w^2 + x^2 + y^2 + z^2}$$

$$q \cdot q^* = |q|^2$$

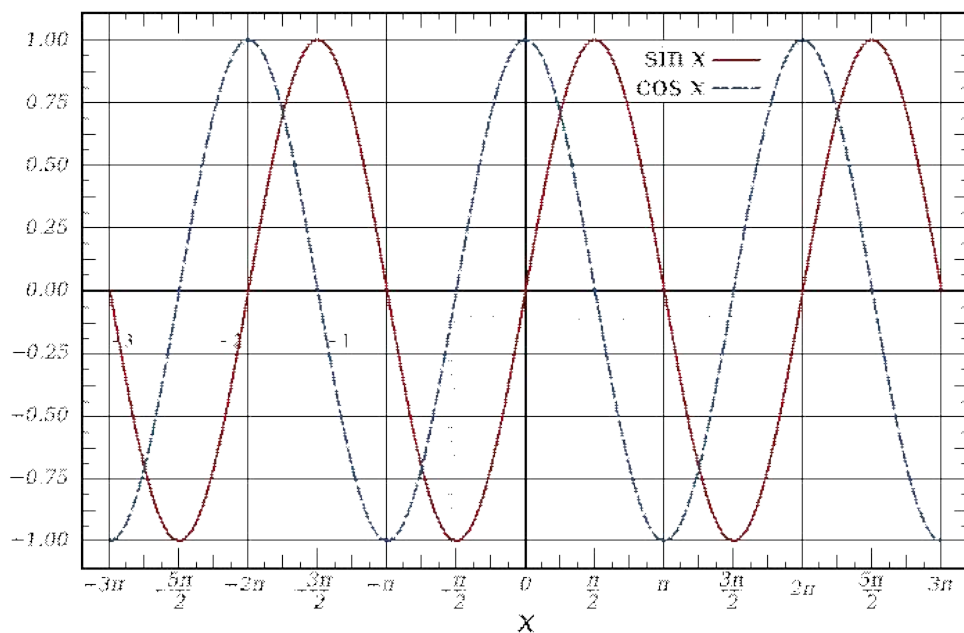
$$v' = qvq^*$$



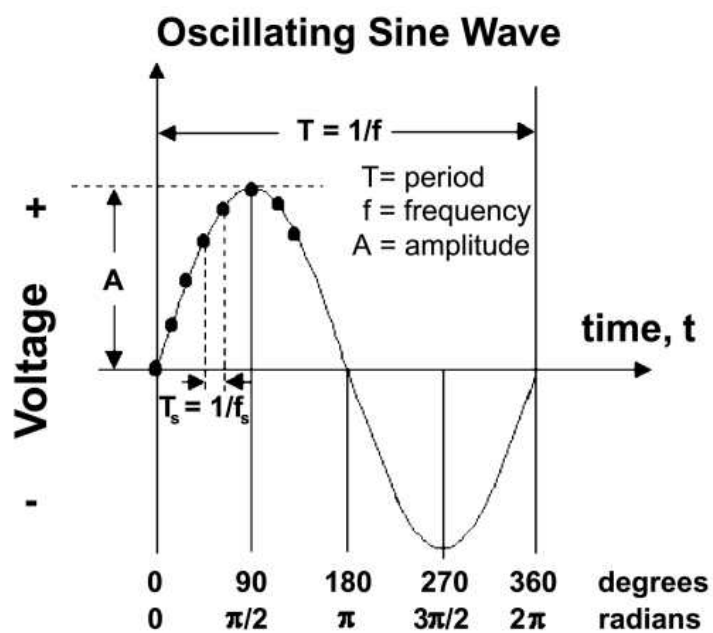

---

## Fourier Transform

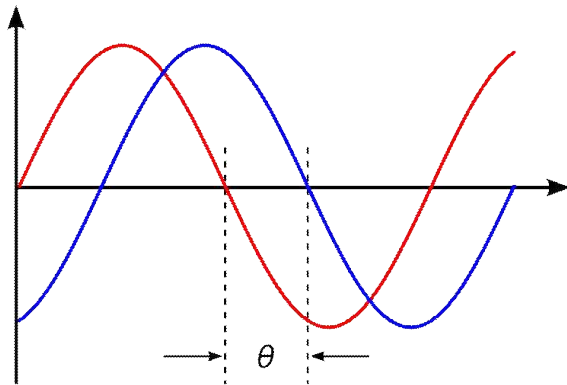
### Sinusoid Curve



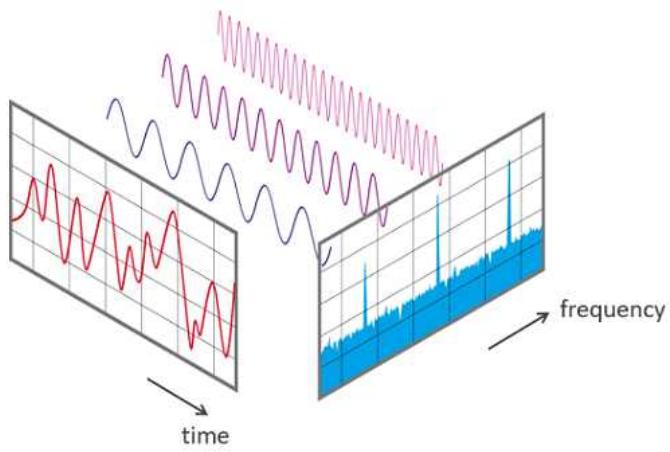
## Period, Frequency and Amplitude

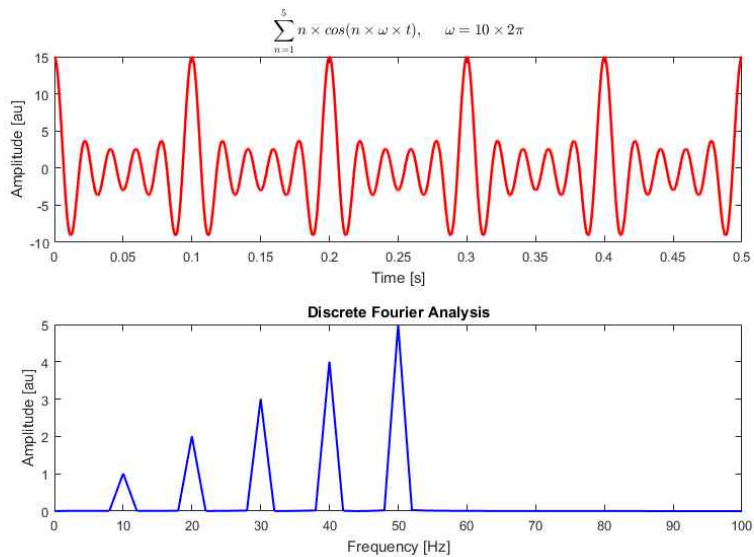


## Phase



## Observation

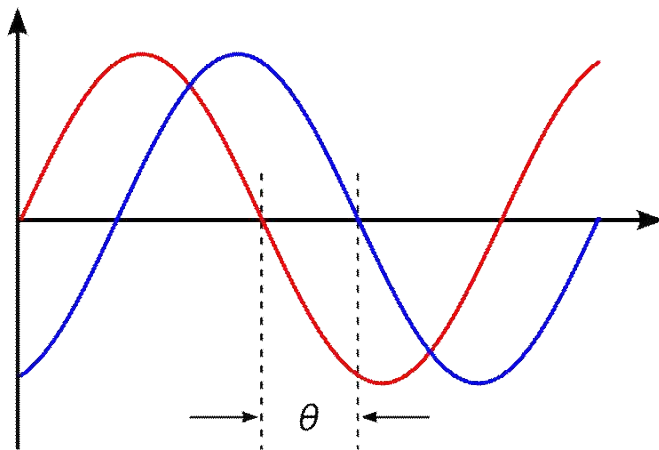




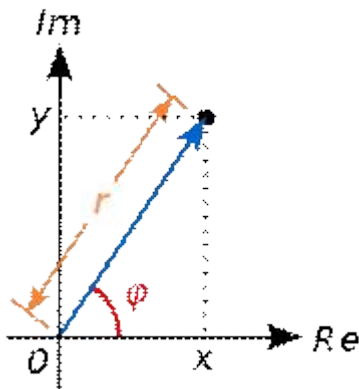
$$f(t) = \cos(2\pi t) + 0.5\cos(2\pi 4t) + \dots$$

$$f(t) = \sum_{v=-\infty}^{\infty} A(v)\cos(2\pi vt)$$

## Phase shift



$$f(v) = \cos(2\pi t) + 0.5\cos(2\pi 4t + \pi/4) + \dots$$



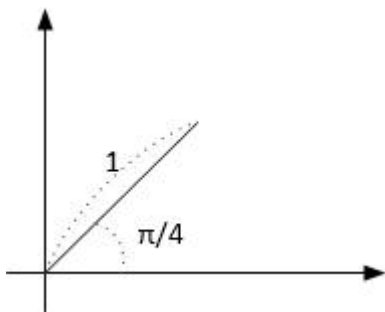
$$F(v) = r(\cos(\theta) + i \sin(\theta))$$

$$e^{ix} = \cos(x) + i \sin(x)$$

$$f(t) = \sum_{v=-\infty}^{\infty} F(v)e^{(2\pi i vt)}$$

$$f(t) = \int_{v=-\infty}^{\infty} F(v)e^{(2\pi i vt)} dv$$

### Example

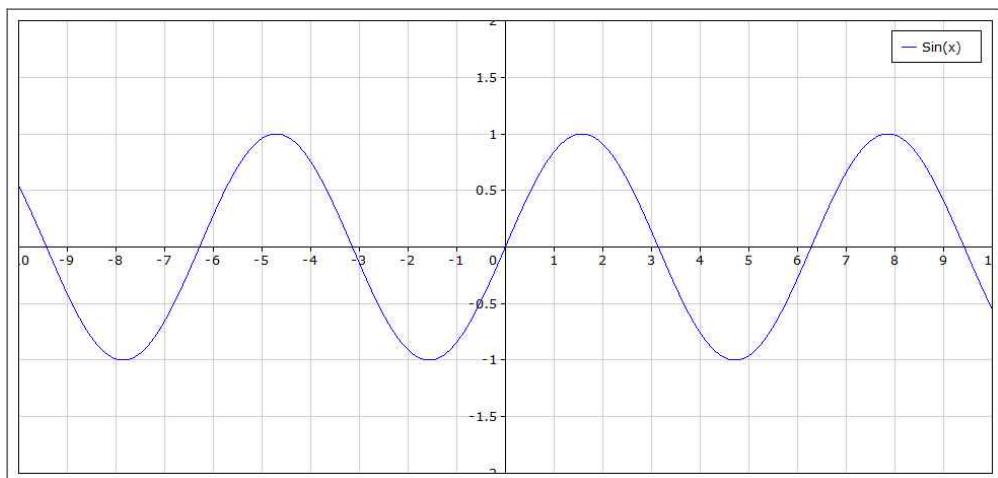
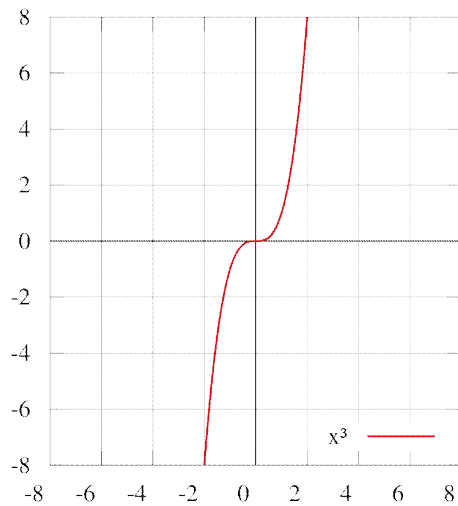


$$\begin{aligned} & (\cos(\pi/4) + i \sin(\pi/4))e^{2\pi i vt} \\ &= (\cos(\pi/4) + i \sin(\pi/4))(\cos(2\pi vt) + i \sin(2\pi vt)) \\ &= \cos(\pi/4)\cos(2\pi vt) + \cos(\pi/4)\sin(2\pi vt)i \\ &+ \sin(\pi/4)\cos(2\pi vt)i - \sin(\pi/4)\sin(2\pi vt) \end{aligned}$$

$$= \cos(2\pi vt + \frac{\pi}{4}) + \sin(\frac{\pi}{4} + 2\pi vt)i$$

## Odd function

$$-f(x) = f(-x)$$



$$\sin(x)$$

## Inverse Fourier Transform

$$f(t) = \sum_{v=-\infty}^{\infty} F(v) e^{2\pi i v t}$$

$$f(t) = \int_{v=-\infty}^{\infty} F(v) e^{2\pi i v t} dv$$

## Forward Fourier Transform

$$F(v) = \sum_{t=-\infty}^{\infty} f(t) e^{-2\pi i v t}$$

$$F(v) = \int_{t=-\infty}^{\infty} f(t) e^{-2\pi i v t} dt$$

### Hint

$$\frac{f(t)}{e^{2\pi i v t}}$$

## Discrete Fourier Transform

$$F(k) = \sum_{n=0}^{N-1} f(n) e^{-2\pi i k \frac{n}{N}}$$

### Amplitude

$$\frac{|F(k)|}{N} = \frac{\sqrt{\operatorname{Re}(F(k))^2 + \operatorname{Im}(F(k))^2}}{N}$$

### Inverse Transform

$$f(n) = \frac{1}{N} \sum_{k=0}^{N-1} F(k) e^{2\pi i k \frac{n}{N}}$$



## Implementation

```
using ComplexArray = std::valarray<std::complex<double> >;
```

### Preparing Signal

```
std::complex<double> test[BIN_SIZE];  
// fill test signal data  
double x = 0;  
double y;  
for(int i = 0; i < BIN_SIZE; ++i)  
{  
    test[i] = std::complex<double>( SignalFunction( x )  
);  
    x += SAMPLING_STEP;  
}
```

### Forward Fourier Transform

```
void dft(ComplexArray& x)  
{  
    const size_t N = x.size();  
  
    ComplexArray xresult(N);  
  
    for (size_t k = 0; k < N; ++k)  
    {  
        xresult[k] = 0;  
        for (size_t n = 0; n < N; ++n)  
        {
```

```

        std::complex<double> e = std::polar(1.0, -2 * M_PI
* k * (double)n / (double)N);
        xresult[k] += x[n] * e;
    }
}
x = xresult;
}

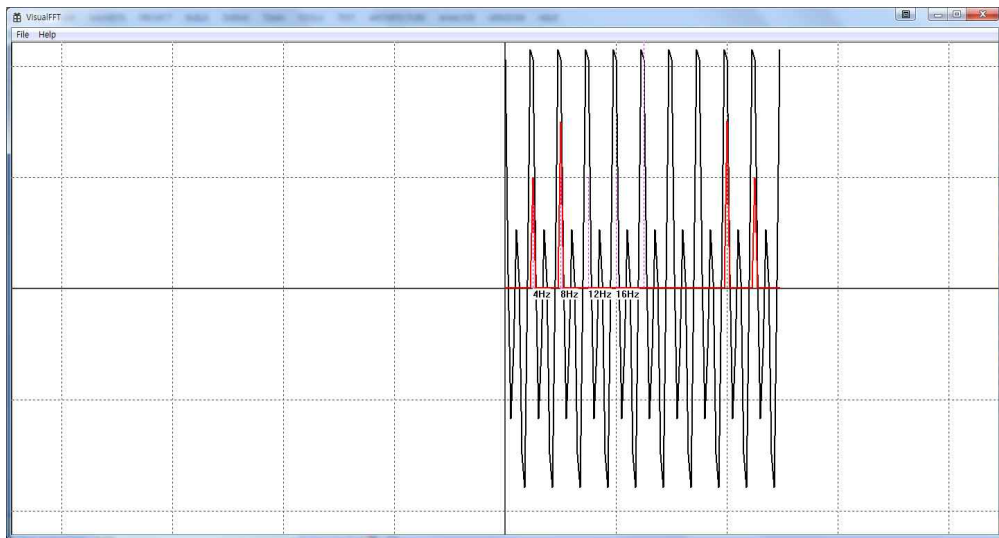
```

## Interpretation of the result

```

// display Hz for every 10 steps
if(i % 10 == 0)
{
    KVectorUtil::DrawLine( g_hdc, KVector2(
(double)x, (double)1 ), KVector2( (double)x, (double)0 ), 1,
PS_DOT, RGB( 255, 0, 255 ) );
    KVector2 screenPos =
KVectorUtil::GetScreenPoint( KVector2( x, 0.0 ) );
    char buffer[80];
    double ratio = (double)i / double(BIN_SIZE);
    sprintf_s( buffer, "%gHz", ratio /
SAMPLING_STEP );
    ::TextOutA( hdc, (int)screenPos.x,
(int)screenPos.y, buffer, strlen( buffer ) );
}

```



## Fast Fourier Transform

@