



# Krótkoczasowa Transformata Fouriera

PRZYGOTOWAŁ MICHAŁ TRACEWICZ

# Czym jest transformata Fouriera?

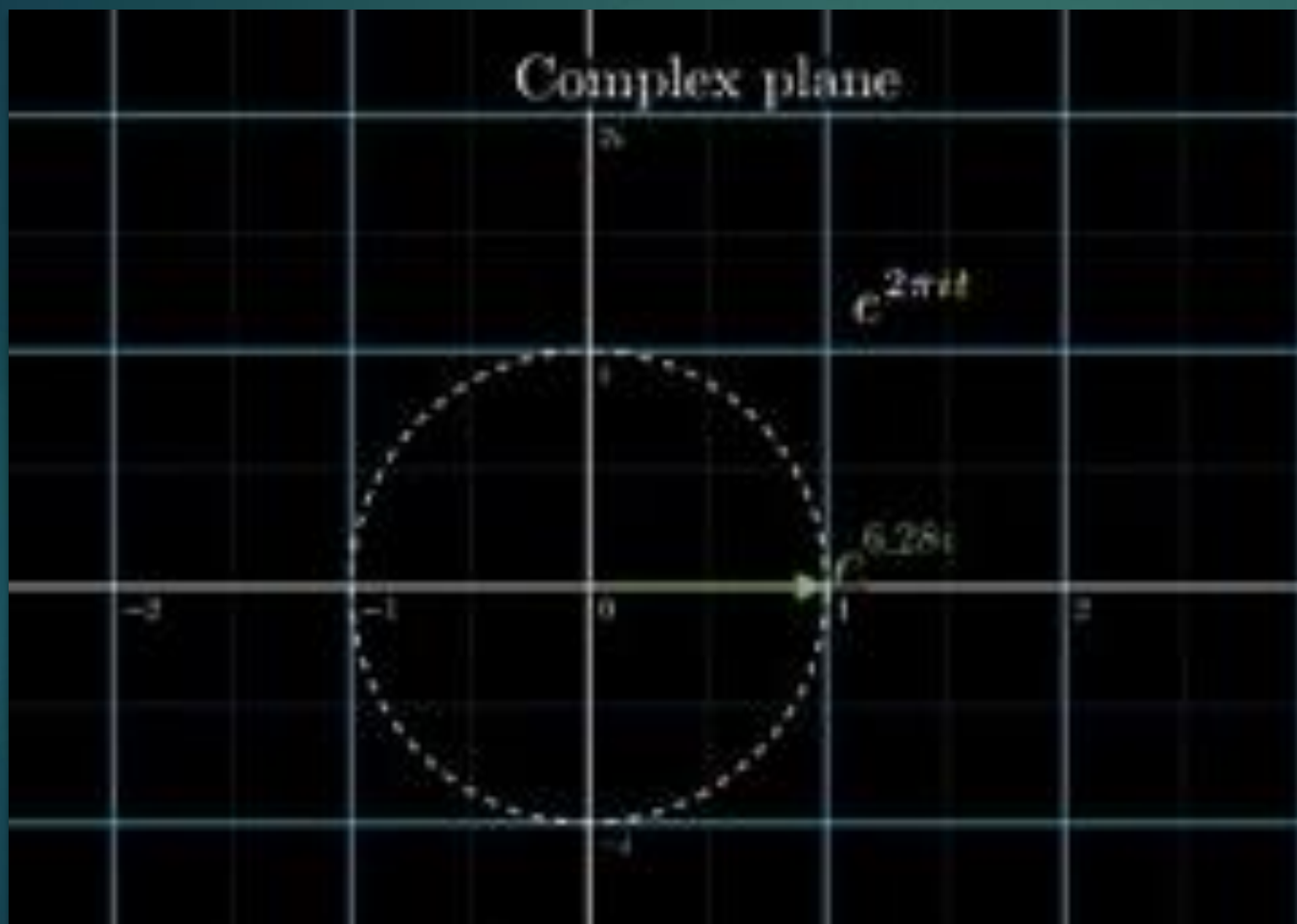
Transformata Fouriera jest to funkcja  $x: \mathbb{R} \rightarrow \mathbb{C}$ ,  
opisana wzorem:

$$X(f) = \int_{\mathbb{R}} x(t) e^{-2\pi i f t} dt, \text{ gdzie}$$

$f$  - częstotliwość,

$t$  - czas,

$x(t)$  - sygnał



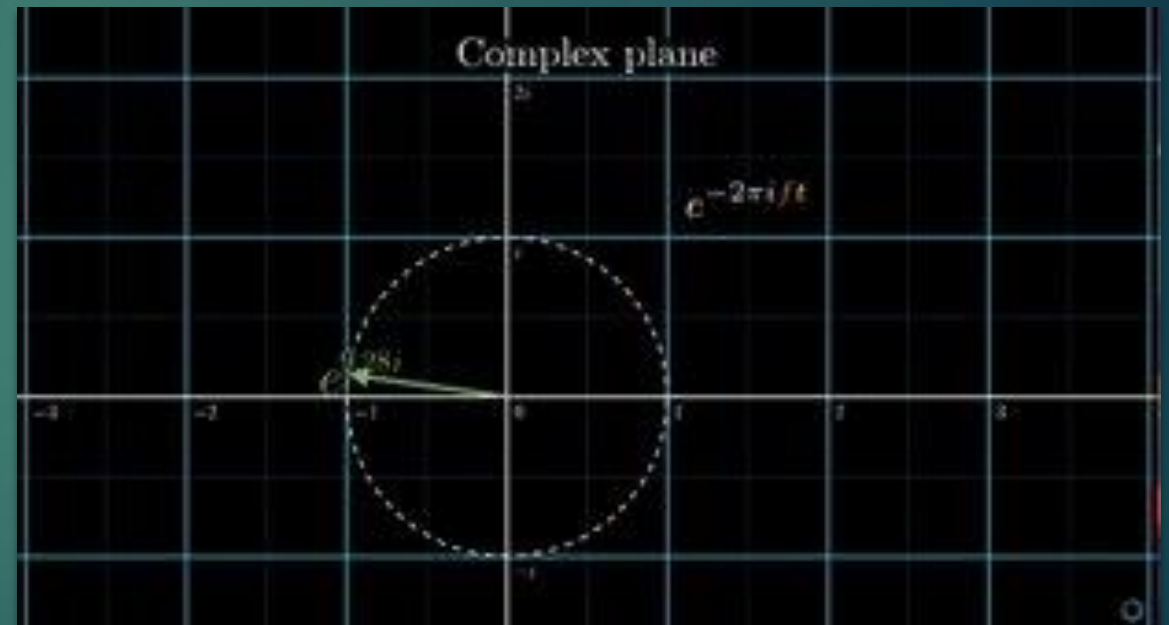
# Wyjaśnienie wzoru

Liczby zespolone posiadają  
własność taką, że

wykres  $e^{2\pi i t}$  tworzy okrąg o  
promieniu , rysowany jest on w  
kierunku przeciwny do ruch  
wskazówek zegara.

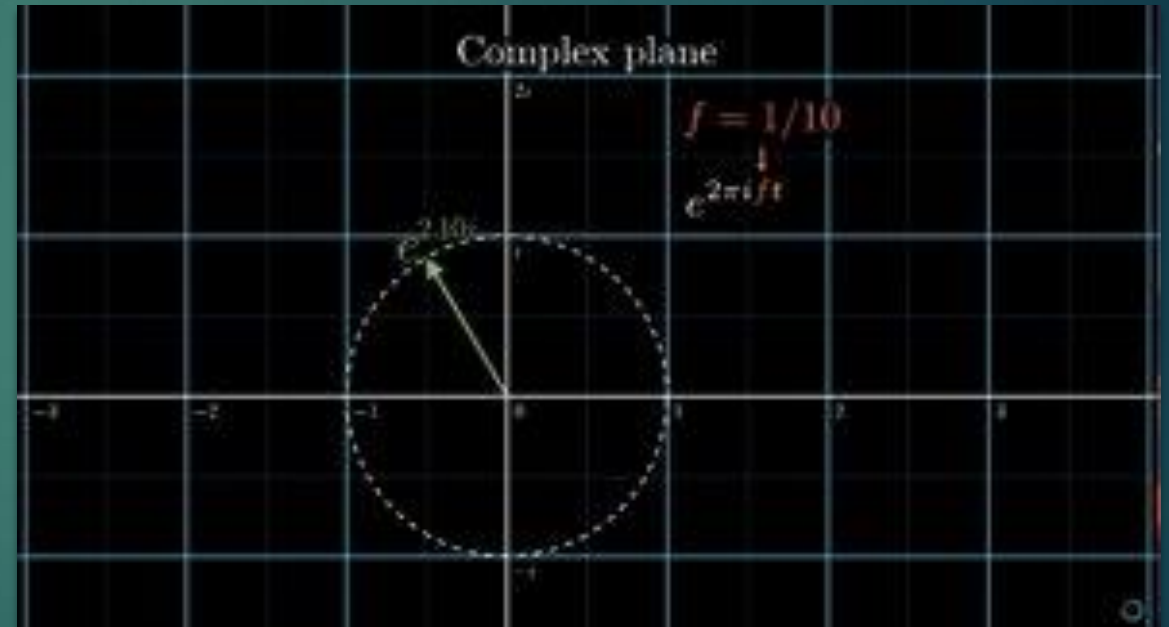
# Wyjaśnienie wzoru c.d.

Jeżeli przed wykładnikiem  
dodamy – to nasz wykres będzie  
rysowany zgodnie z ruchem  
wskazówek zegara



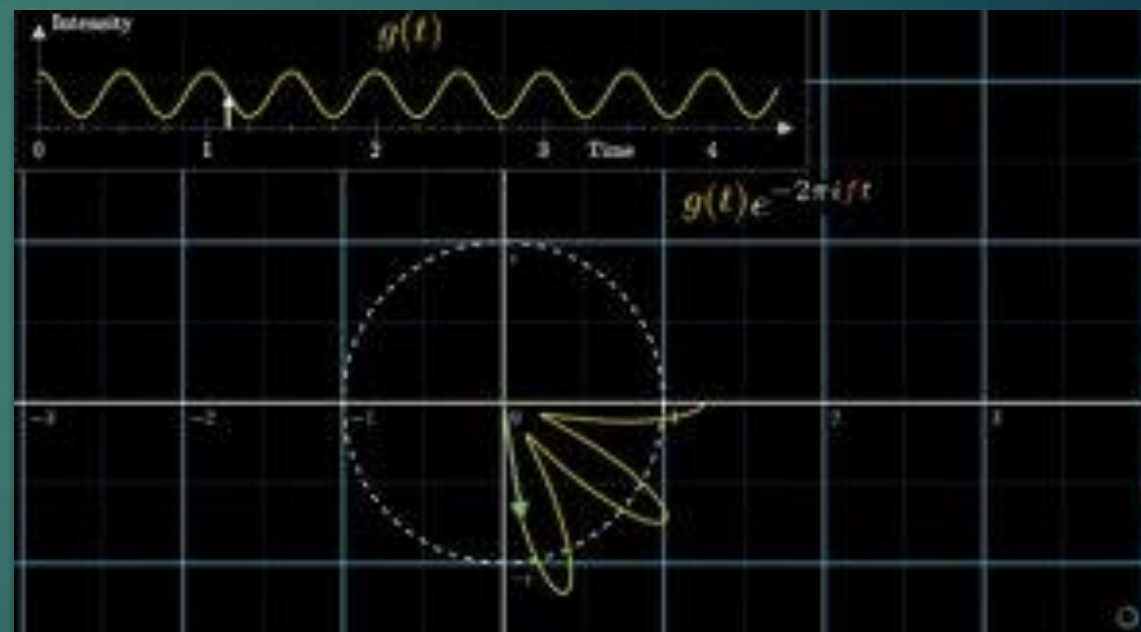
# Wyjaśnienie wzoru c.d.

Dodając częstotliwość czyli  $f$  regulujemy z jaką szybkością po okręgu porusza się nasza strzałka, bez niej jeden obrót zajmuję sekunde



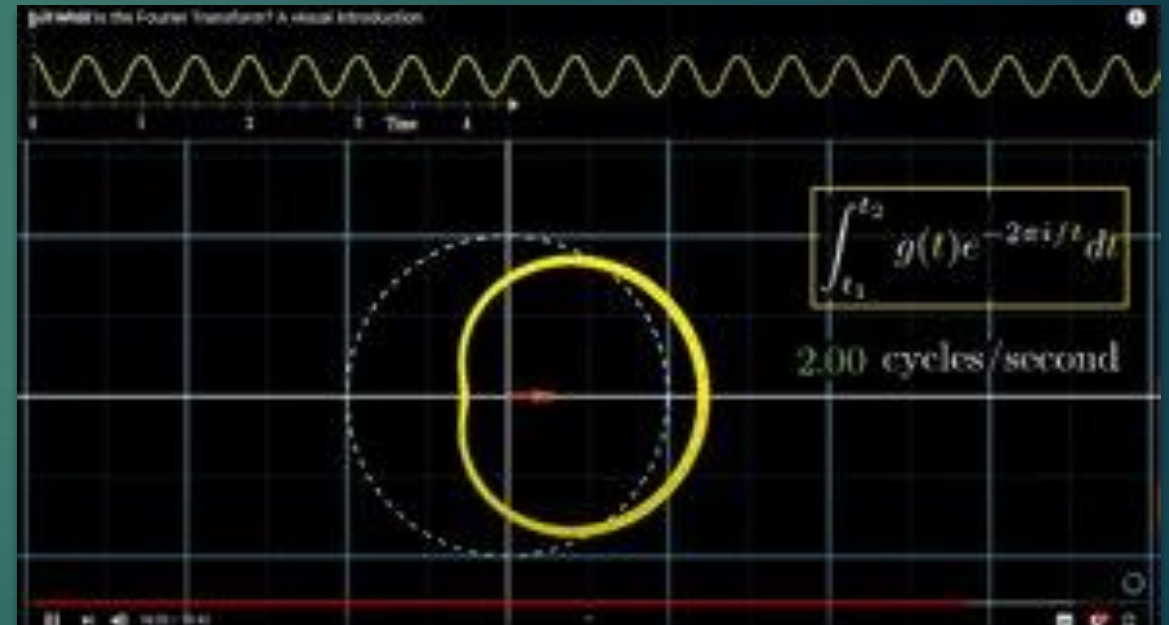
# Wyjaśnienie wzoru c.d.

Gdy weźmiemy sygnał  
(tu oznaczony jako  $g(t)$ )  
i wymnożymy przez niego  
dotychczasowy wzór to  
będziemy skalować odległość w  
jakiej znajduje się środek układu  
współrzędnych.



# Wyjaśnienie wzoru c.d.

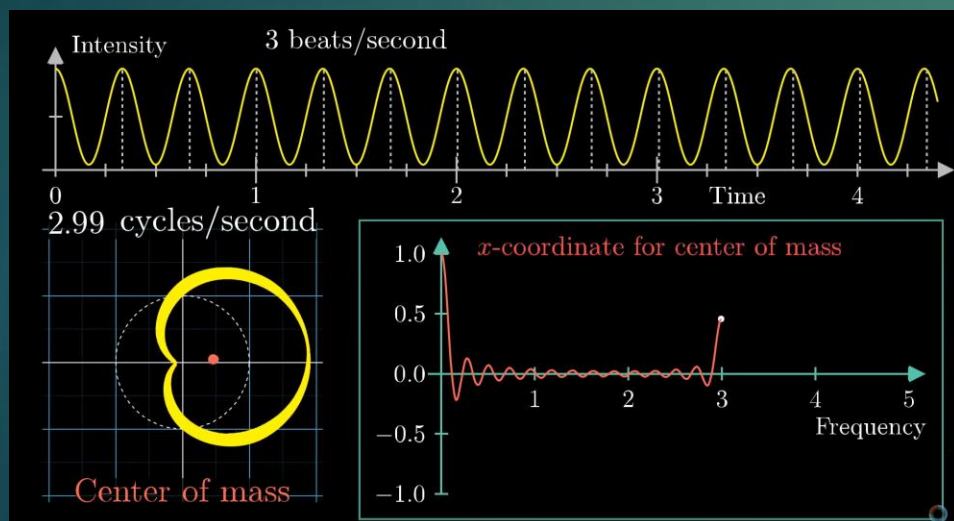
Całkując to wyrażenie otrzymamy współzrędną środka ciężkości figury powstałej z wykresu.



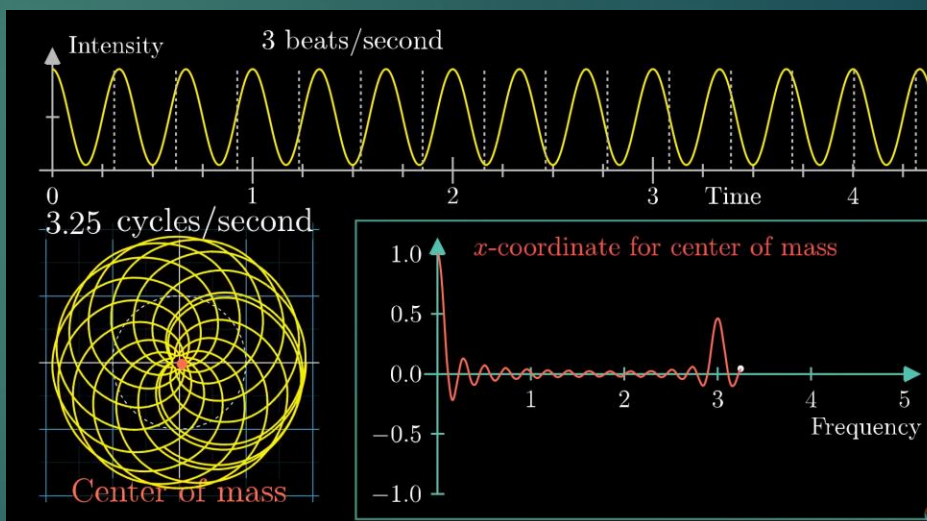


# Wyjaśnienie wzoru c.d.

Gdy dana częstotliwość pokrywa się z jedną z częstotliwości składowych sygnału to na wykresie wskazującym położenie środka ciężkości pokaze się znaczny wzrost



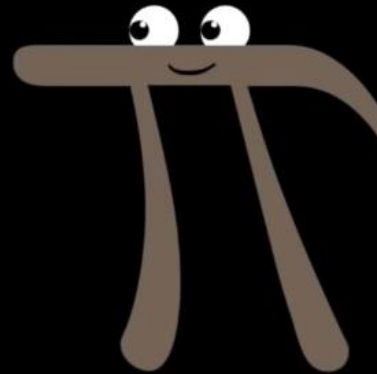
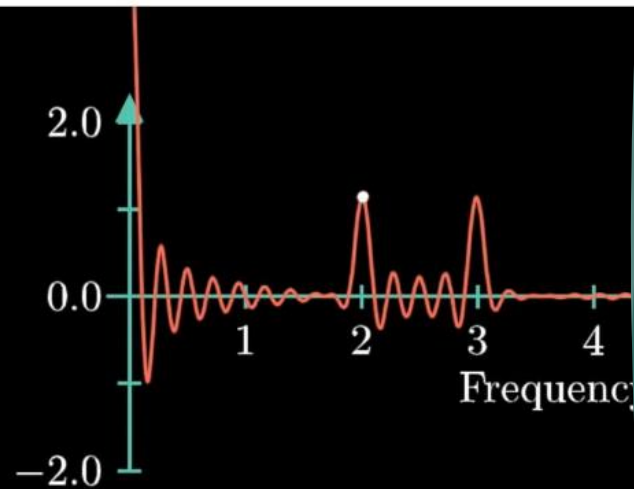
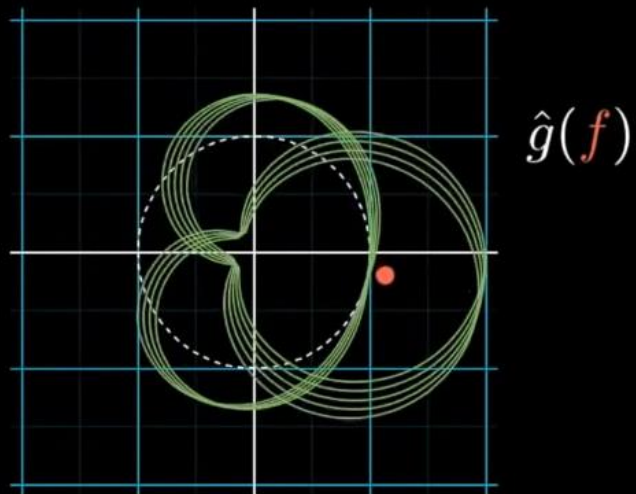
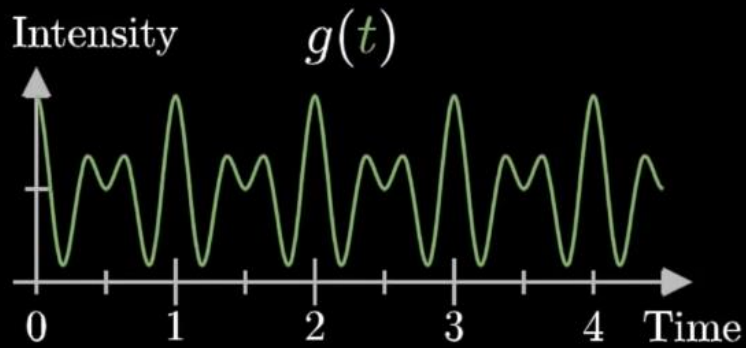
Gdy częstotliwość wynosi 2.99 widzimy, że na wykresie pozycji środka ciężkości pojawia się wzrost



Gdy częstotliwość wynosi 3.25 widzimy, że na wykresie pozycji środka ciężkości zbliżamy się do zera



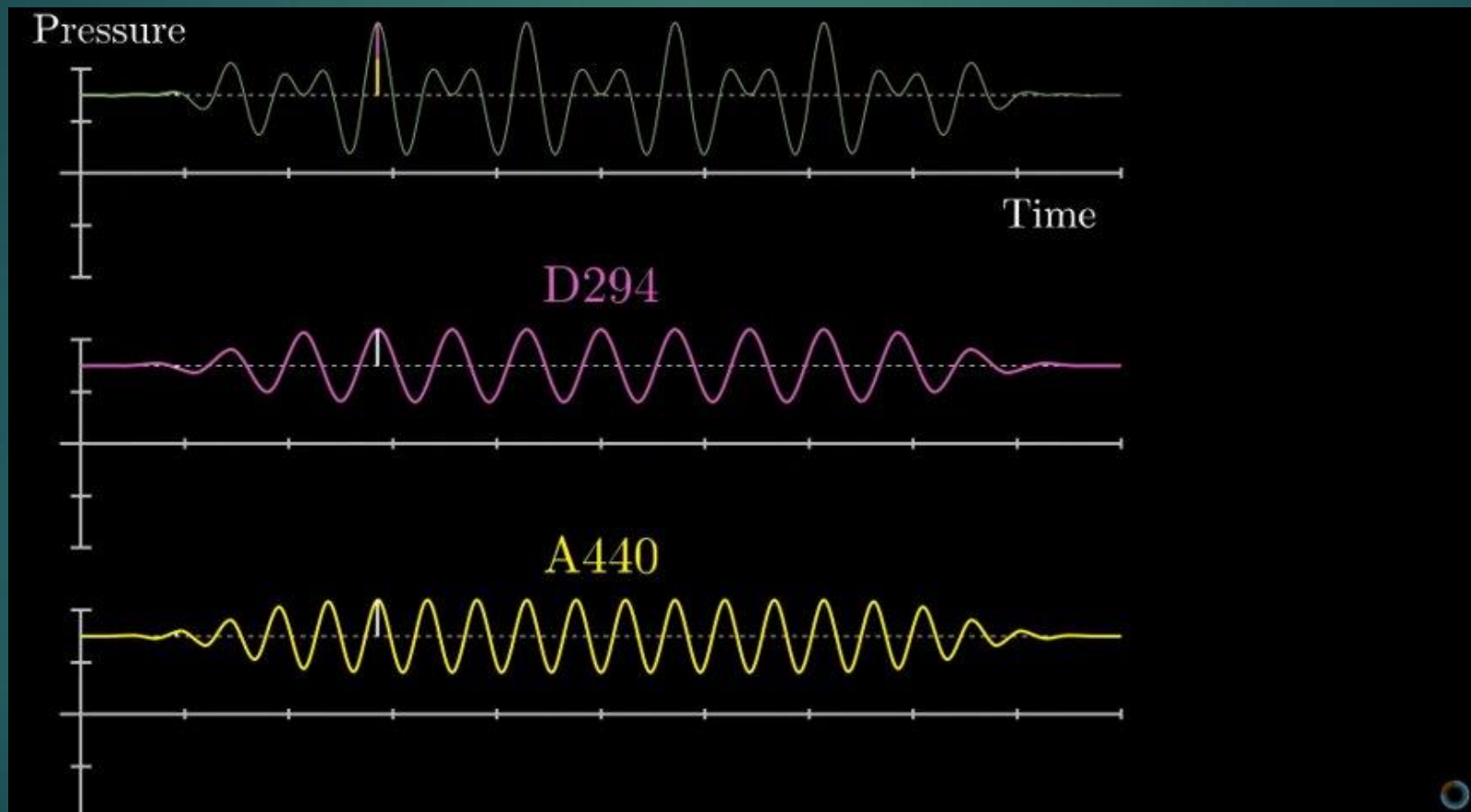
$$\hat{g}(f) = \int_{t_1}^{t_2} g(t) e^{-2\pi i f t} dt$$



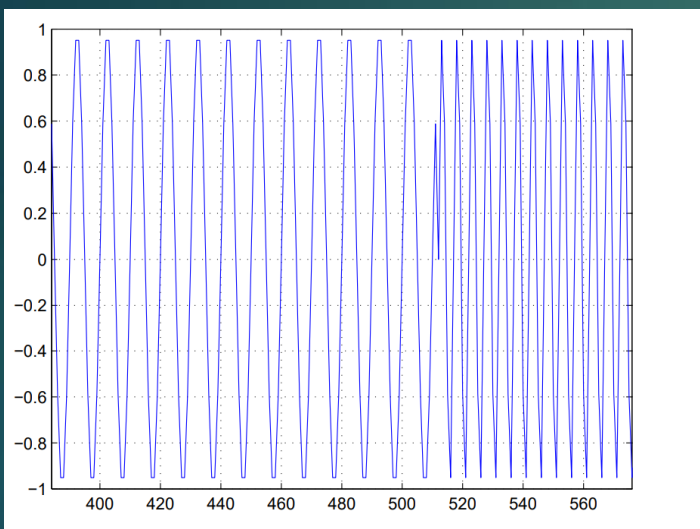
Wyjaśnienie wzoru  
c.d.

<https://www.youtube.com/watch?v=spUNpyF58BY>

# Przykładowe zastosowanie - wyciąganie częstotliwości z sygnału

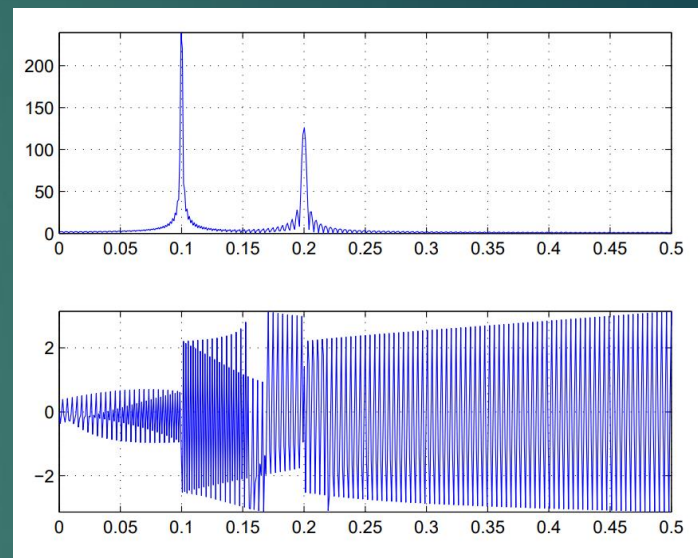


# Przykładowe zastosowanie - wyciąganie częstotliwości z sygnału



Sygnał posiadający dwie  
częstotliwości: 0.1 i 0.2

Wykresy pochodzą ze skryptu  
"Podstawy Przetwarzania Sygnałów"  
dr. Dariusz Borkowski

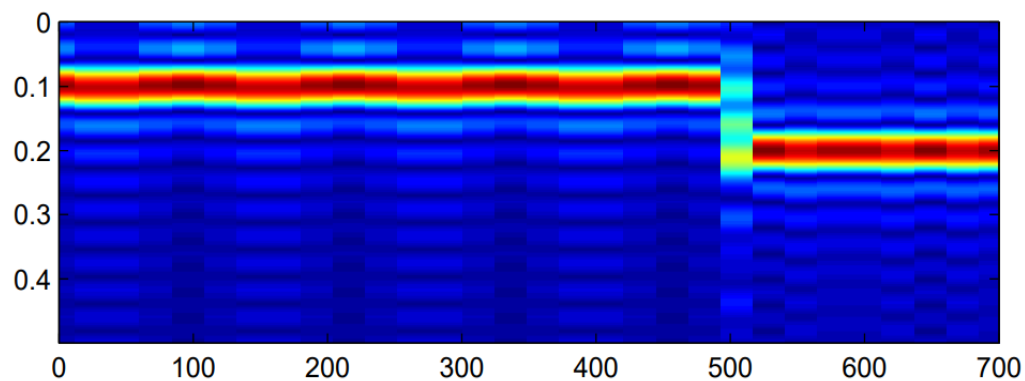
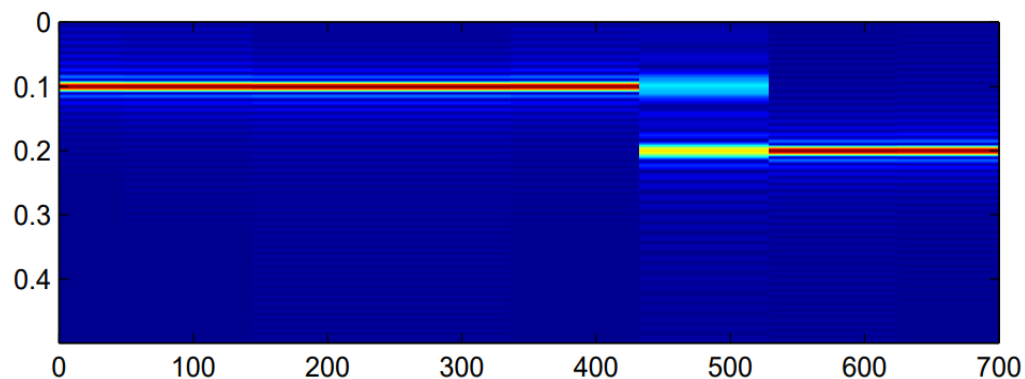


Moduł oraz faza uzyskane  
z pomocą transformaty,  
na pierwszym wykresie  
możemy zaobserwować  
jakie częstotliwości  
występują

# Czego nie może transformata Fourier'a?

Transformata Fouriera pozwala nam wychwycić jakie częstotliwości pojawiają się w naszym sygnale, jednak nie posiadamy czytelej dla człowieka informacji o tym w jakim czasie te częstotliwości się pojawiają

# Rozwiązanie - krótkoczasowa transformata Fourier'a



Wykres powstały z krótkoczasowej transformaty Fourier'a

Na górnym wykresie dokładniej widzimy jaką częstotliwość występowała natomiast na dolnym w jakim czasie.

Oś x – czas

Oś y - częstotliwość

Kolor im cieplejszy tym większa wartość w danym punkcie

# Jak działa krótkoczasowa transformata Fourier'a?

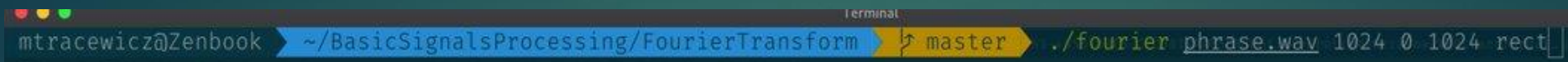
W celu uzyskania czasu wystąpienia częstotliwości dzielimy nasz sygnał na równe fragmenty i liczymy na każdym z nich transformatę Fourier'a.

Większa ilość fragmentów zwiększy nam dokładność czasową kosztem dokładności częstotliwości.

Możemy częściowo temu zaradzić poprzez wybranie fragmentów tak aby na siebie zachodziły.



# Moja implementacja w C++

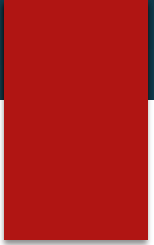


```
mtracewicz@Zenbook ~/BasicSignalsProcessing/FourierTransform master ./fourier phrase.wav 1024 0 1024 rect
```

Argumenty to kolejno:

- Plik z którego wczytamy sygnał
- Ilość próbek, które znajdują się w jednym fragmencie
- Jakie nachodzenie się fragmentów zostanie zastosowane
- Do jakiej wielkości próbki ma być uzupełniony zerami wybrany fragment
- Okno (prostokątne/hanna/hamma) - przez nie przemnażamy nasz fragment





```
//Sprawdzenie poprawności, czy ilość próbek we fragmencie nie jest większa niż długość sygnału
//oraz czy nachodzenie nie jest większe niż 1/4 fragmentu
if (lb > xt.size() || ovlp > lb / 4)
{
    std::cerr << "Invalid parameters";
    return 1;
}
```

```
//Obliczamy ilość próbek w przedziale
int64_t npt = lb - 2 * ovlp;
//Obliczamy ilość przedziałów
int64_t nsi = fix(double(xt.size()) / npt);
//W razie gdy ilość próbek sygnału wejściowego nie jest równa
//ilości przedziałów pomnożonej przez ilość próbek - uzupełniamy zerami
if (xt.size() > (npt * nsi))
{
    appendZeros(xt, (nsi + 1) * npt - xt.size());
}
```

```
//Jeżeli fragmenty mają się nachodzić to modyfikujemy wektor zawierający próbki  
if (ovlp > 0)  
{  
    nsi = fix(double(xt.size()) / npt);  
    for (int64_t i = (nsi - 1); i >= 1; --i)  
    {  
        std::vector<std::vector<double>> tmp;  
        tmp.push_back(sliceVector(xt, 0, i * npt - 1));  
        tmp.push_back(sliceVector(xt, i * npt, i * npt + ovlp - 1));  
        tmp.push_back(sliceVector(xt, i * npt - ovlp, i * npt - 1));  
        tmp.push_back(sliceVector(xt, i * npt, xt.size() - 1));  
        xt = concatenateVectors(tmp);  
    }  
    std::vector<std::vector<double>> tmp;  
    tmp.push_back(zeros(ovlp));  
    tmp.push_back(xt);  
    tmp.push_back(zeros(ovlp));  
    xt = concatenateVectors(tmp);  
}
```

```
//Obliczenie okna
if (window.compare("hamm") == 0)
{
    wn = createHammWindow(npt);
}
else if (window.compare("hann") == 0)
{
    wn = createHannWindow(npt);
}
else
{
    wn = ones(npt);
}
```

```
std::vector<double> createHammWindow(int64_t count)
{
    std::vector<double> result;
    for (int64_t i = 0; i < count; i++)
    {
        result.push_back(0.54 - 0.46 * cos(2 * M_PI * i / count));
    }
    return result;
}
```

```
std::vector<double> createHannWindow(int64_t count)
{
    std::vector<double> result;
    for (int64_t i = 0; i < count; i++)
    {
        result.push_back(0.5 - 0.5 * cos(2 * M_PI * i / count));
    }
    return result;
}
```

```
auto xs = zeros(npt, nsi);  
assign(xs, xt, npt * nsi);  
auto xsf = createValues(xs, wn, lfft);  
min(xsf, 255);  
xsf = cutRows(xsf, lfft / 2);  
auto time = calculateTime(xt.size(), fs);
```



*//Wyświetlamy wykres sygnału wejściowego*

```
Gnuplot plot;
```

```
plot.set_grid()
```

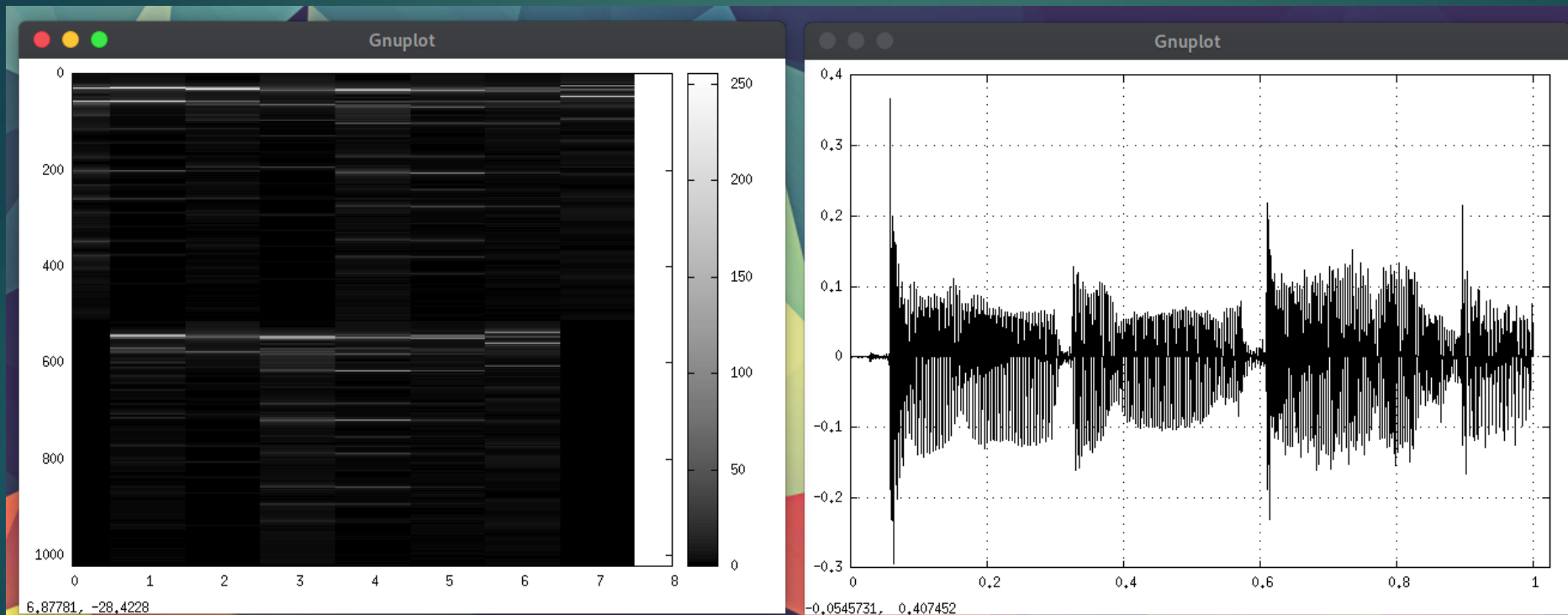
```
    .set_xrange(0.0, double(xt.size()) / fs)
```

```
    .set_style("impulses lc black")
```

```
    .plot_xy(time, xt);
```

```
//Wyświetlamy wykres wynikowy
Gnuplot plot2;
const int iWidth = nsi;
const int iHeight = lfft;
plot2.set_xrange(0, iWidth)
      .set_yrange(iHeight, 0)
      .set_cbrange(0, 255)
      .cmd("set palette gray");
unsigned char ucPicBuf[iWidth * iHeight];
for (int i = iHeight - 1; i >= 0; i--)
{
    for (int j = 0; j < iWidth; j++)
    {
        ucPicBuf[i * iWidth + j] = xsf[j][i];
    }
}
plot2.plot_image(ucPicBuf, iWidth, iHeight);
```

# Moja implementacja – wynik



Po lewej Krótkoczasowa transformata Fourier'a  
Po prawej sygnał wczytany z pliku