Discrete Fourier Transformation (DFT)

wiki: https://en.wikipedia.org/wiki/Discrete_Fourier_transform

Definition: transforms a sequence of N complex numbers $x_0, x_1, ..., x_N$ into another sequence of complex numbers $y_0, y_1, ..., y_N$

$$y_k = \sum_{n=0}^{N-1} e^{-2\pi i n k / N} x_n$$

we could proove this reverse property (orthogonality)

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} e^{2\pi i n k / N} y_k$$

generate data

```
data_x = rand(1,8);
```

built-in

```
y_builtin = fft(data_x);
disp(y_builtin)
```

```
Columns 1 through 7
 4.3162 + 0.0000i   0.4947 - 0.6792i   1.0416 + 0.4569i  -0.1300 - 0.9822i  -0.6110 + 0.0000i  -0.1300 + 0.9822i
Column 8
 0.4947 + 0.6792i
```

my naive DFT

```
y_naive_DFT = my_naive_dft(data_x);
disp(y_naive_DFT)
```

```
Columns 1 through 7
 4.3162 + 0.0000i   0.4947 - 0.6792i   1.0416 + 0.4569i  -0.1300 - 0.9822i  -0.6110 - 0.0000i  -0.1300 + 0.9822i
Column 8
 0.4947 + 0.6792i
```

Fast Fourier Transformation (FFT)

wiki: https://en.wikipedia.org/wiki/Fast_Fourier_transform

algorithm (detail see wiki)

1. Cooley–Tukey algorithm
2. Prime-factor FFT algorithm
3. Bruun's FFT algorithm

4. Rader's FFT algorithm
5. Bluestein's FFT algorithm
6. Hexagonal Fast Fourier Transform

Cooley–Tukey algorithm

wiki: https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm

$$y_k = E_k + e^{-2\pi ik/N} O_k$$

$$y_{k+N/2} = E_k - e^{-2\pi ik/N} O_k$$

$$E_k = \sum_{m=0}^{N/2-1} e^{-\frac{2\pi i}{N/2}mk} x_{2m}$$

$$O_k = \sum_{m=0}^{N/2-1} e^{-\frac{2\pi i}{N/2}mk} x_{2m+1}$$

```
y_my_fft = my_fft(data_x);
disp(y_my_fft)
```

test reverse property

```
disp(data_x)
% fft-ifft will go back to orgin data
disp(my_ifft(my_fft(data_x)))
```

function

```
function ret = my_naive_dft(x)
% x(float,(1,N1))
% ret(float,(1,N1))
N1 = size(x,2);
ret = x*exp(-2i*pi*(0:N1-1).'*(0:N1-1)/N1);
end

function ret = my_fft(x)
% reference
%   DFT-wiki: https://en.wikipedia.org/wiki/Discrete_Fourier_transform
%   Cooley-Tukey FFT-wiki: https://en.wikipedia.org/wiki/Fast_Fourier_transform
% x(float,(1,N1))
N1 = size(x,2);
if mod(N1,2)~=0, error('radix-2 required'); end
if N1==2
    ret = [x(1)+x(2),x(1)-x(2)];
else
    tmp1 = my_fft(x(1:2:end));
    tmp2 = my_fft(x(2:2:end));
```

```
    tmp3 = tmp2.*exp(-2i*pi*(0:(N1/2-1))/N1);
    ret = [tmp1+tmp3, tmp1-tmp3];
end
end

function ret = my_ifft(x)
% x(1*N1)
% ret(1*N1)
% not available until 20181025
ret = x;
end
```