FOURIER

Téo Kaltrachian et Sergey Platonov

SOMMAIRE

01 CONTEXTE

02 DFT

03 NOISE REMOVAL

04 COMPRESSION

01

CONTEXTE

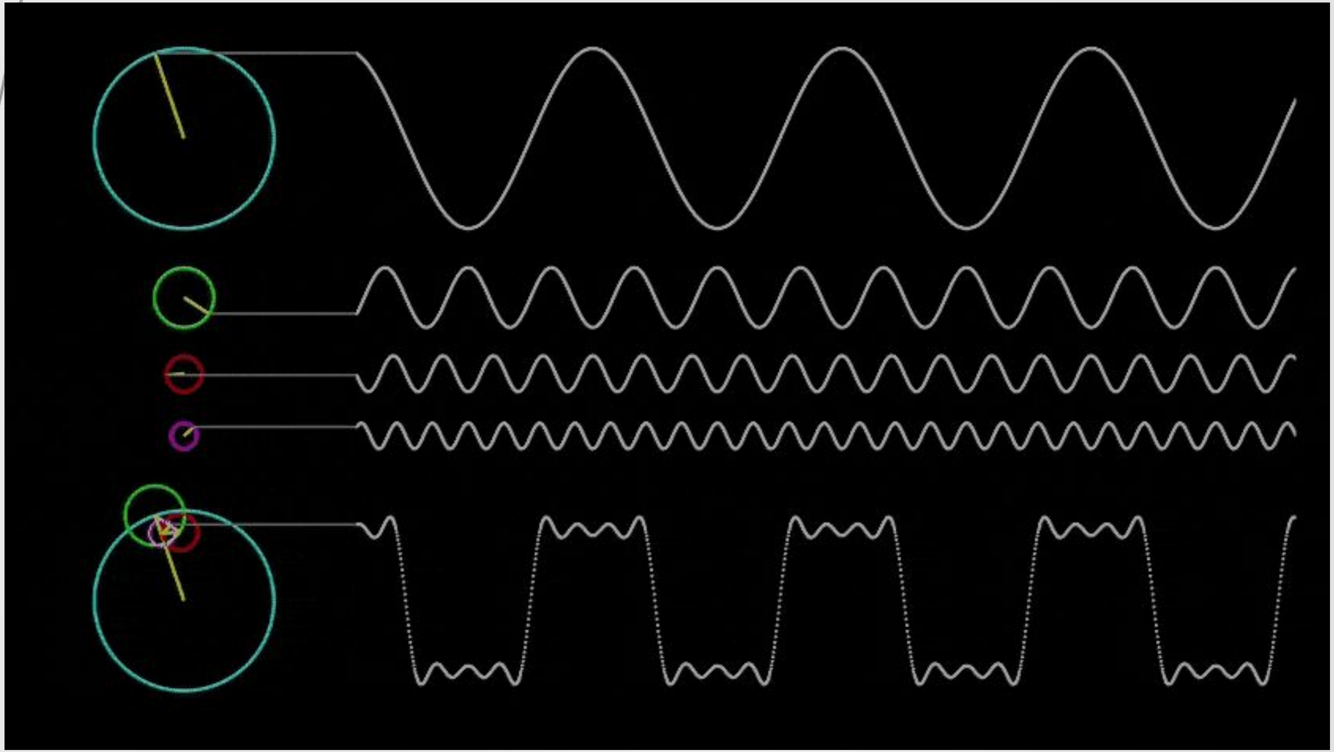
TRANSFORMEE FOURIER ?

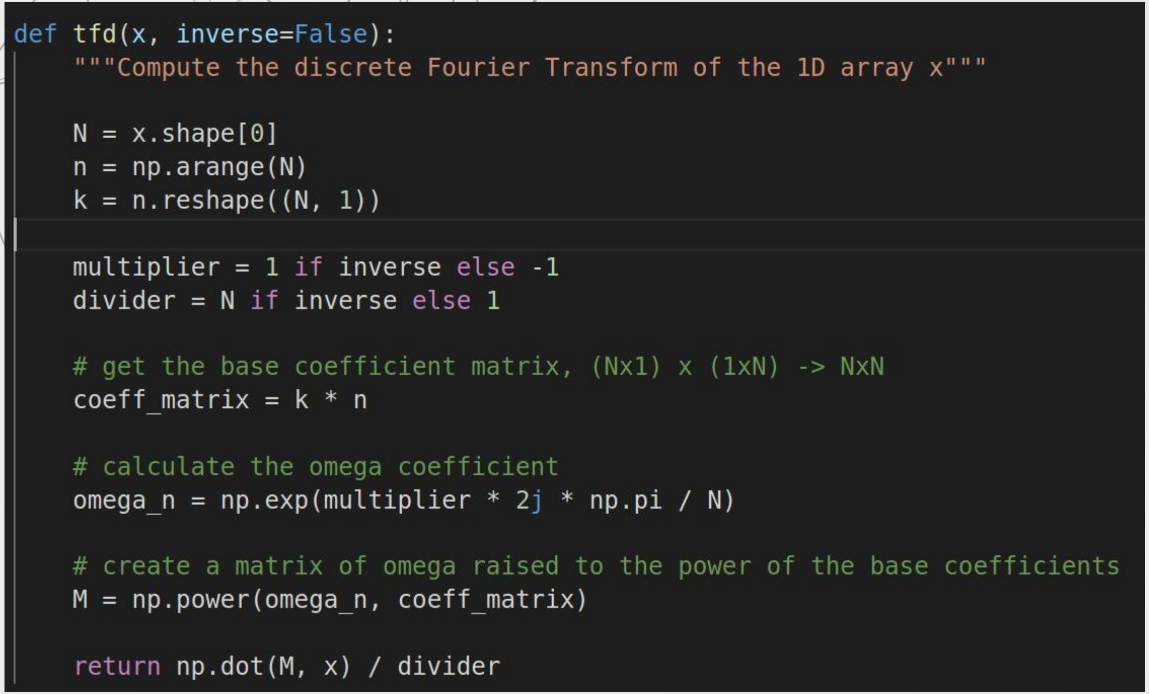
- Compréhension intuitive
- Applications
- Vectorisation du code

02

DFT – DISCRETE FOURIER TRANSFORM

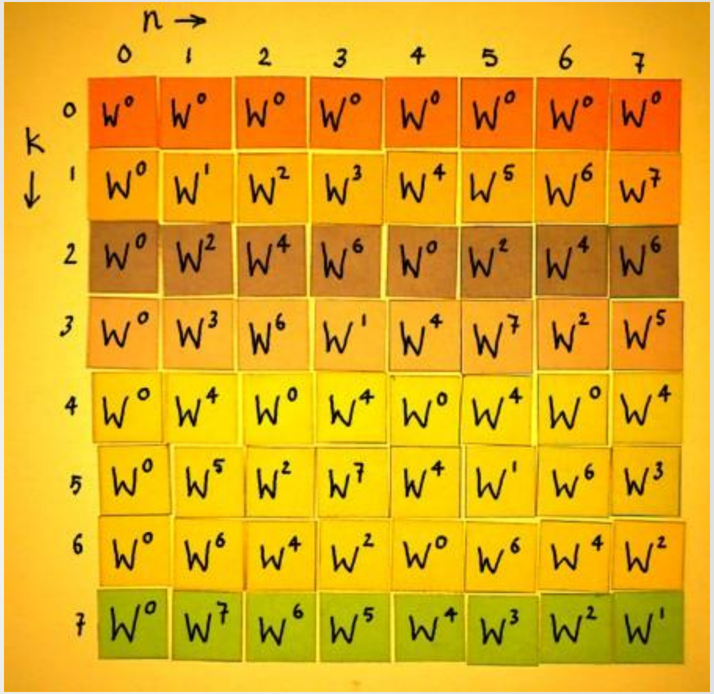
DFT



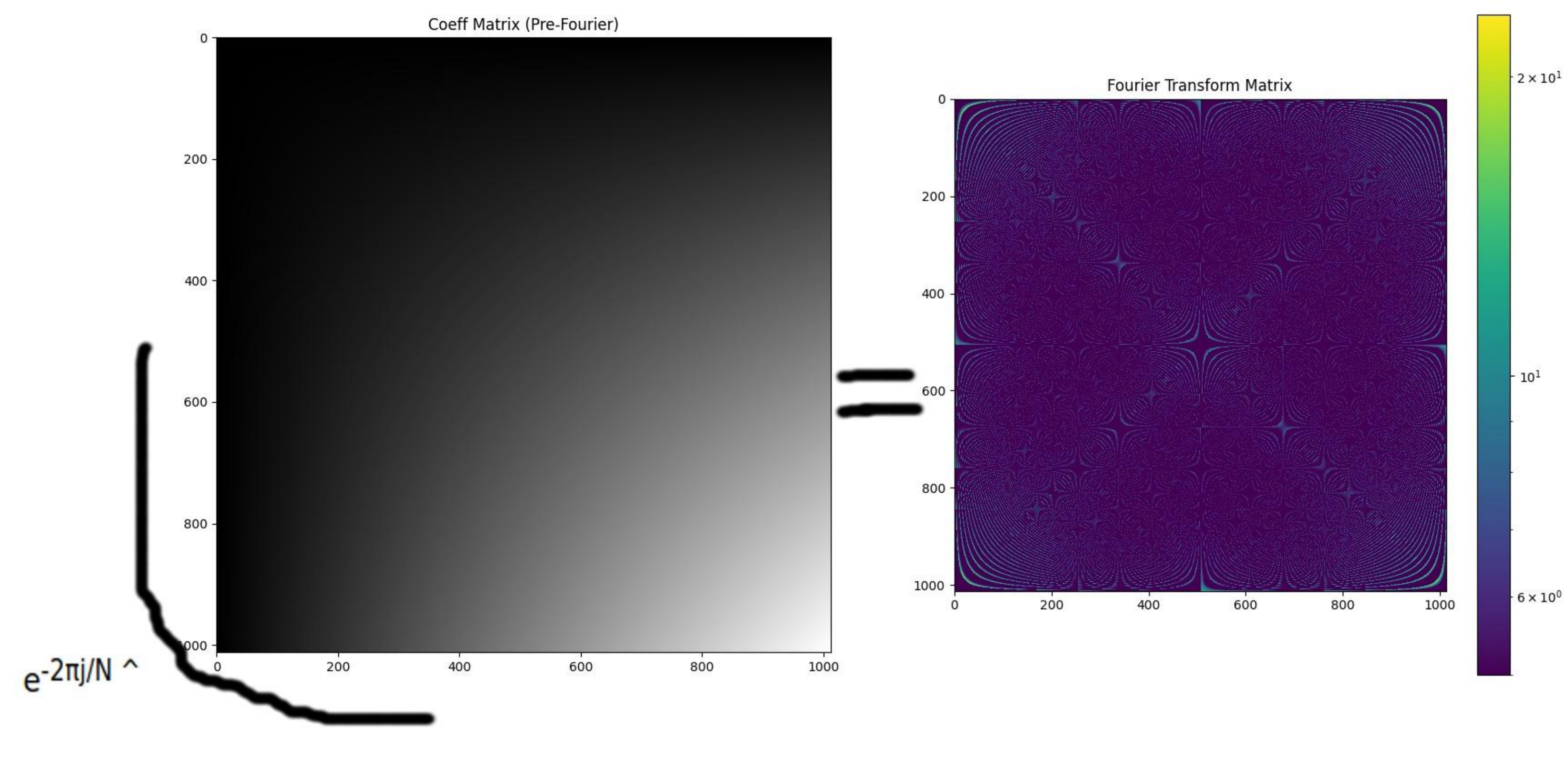


```python
def tfd(x, inverse=False):
    """Compute the discrete Fourier Transform of the 1D array x"""

    N = x.shape[0]
    n = np.arange(N)
    k = n.reshape((N, 1))

    multiplier = 1 if inverse else -1
    divider = N if inverse else 1

    # get the base coefficient matrix, (Nx1) x (1xN) -> NxN
    coeff_matrix = k * n

    # calculate the omega coefficient
    omega_n = np.exp(multiplier * 2j * np.pi / N)

    # create a matrix of omega raised to the power of the base coefficients
    M = np.power(omega_n, coeff_matrix)

    return np.dot(M, x) / divider
```

M

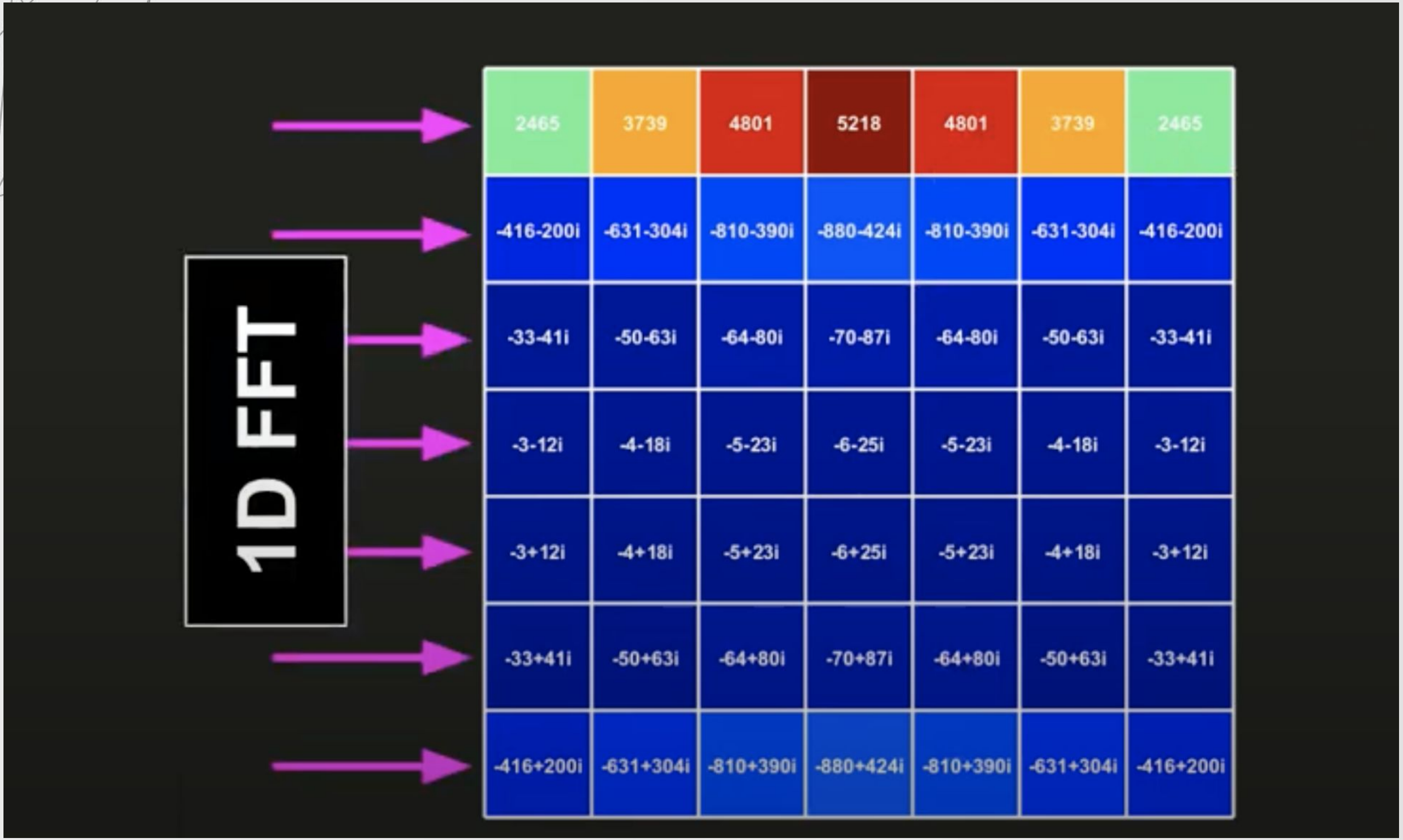


k ↓ n →	0	1	2	3	4	5	6	7
0	W^0	W^0	W^0	W^0	W^0	W^0	W^0	W^0
1	W^0	W^1	W^2	W^3	W^4	W^5	W^6	W^7
2	W^0	W^2	W^4	W^6	W^0	W^2	W^4	W^6
3	W^0	W^3	W^6	W^1	W^4	W^7	W^2	W^5
4	W^0	W^4	W^0	W^4	W^0	W^4	W^0	W^4
5	W^0	W^5	W^2	W^7	W^4	W^1	W^6	W^3
6	W^0	W^6	W^4	W^2	W^0	W^6	W^4	W^2
7	W^0	W^7	W^6	W^5	W^4	W^3	W^2	W^1


Coeff Matrix (Pre-Fourier)
0
200
400
600
800
1000
0
200
400
600
800
1000
Fourier Transform Matrix
0
200
400
600
800
1000
0
200
400
600
800
1000
2×10^1
10^1
6×10^0

$e^{-2\pi j/N}$ ^

DFT2



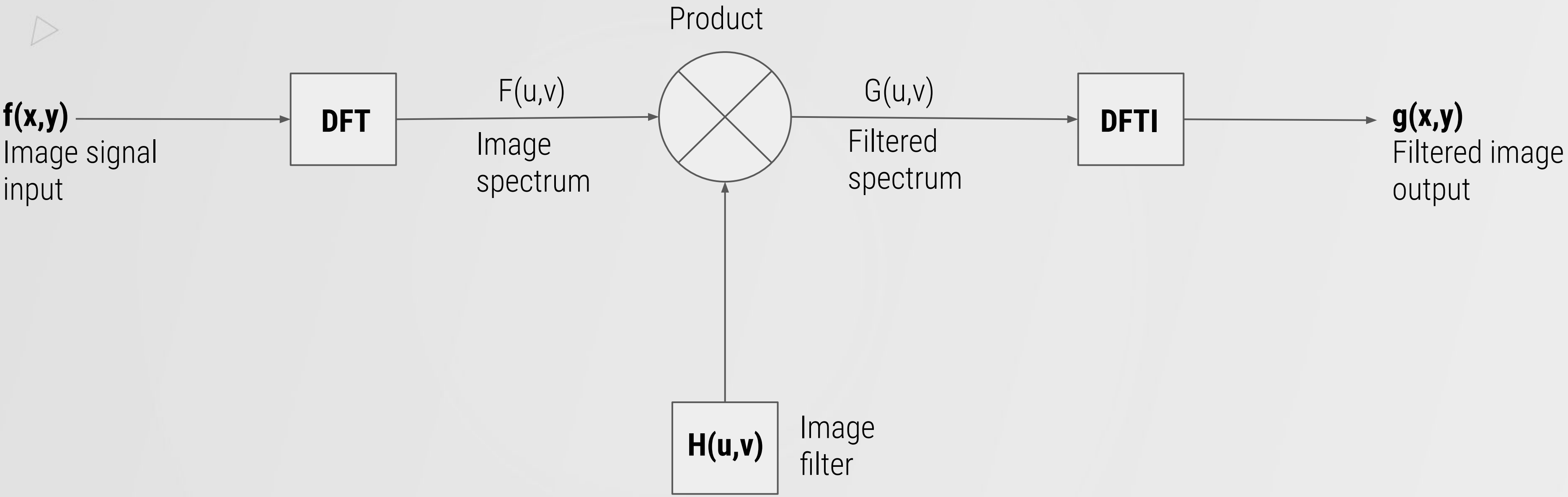
1D FFT

2465	3739	4801	5218	4801	3739	2465
-416-200i	-631-304i	-810-390i	-880-424i	-810-390i	-631-304i	-416-200i
-33-41i	-50-63i	-64-80i	-70-87i	-64-80i	-50-63i	-33-41i
-3-12i	-4-18i	-5-23i	-6-25i	-5-23i	-4-18i	-3-12i
-3+12i	-4+18i	-5+23i	-6+25i	-5+23i	-4+18i	-3+12i
-33+41i	-50+63i	-64+80i	-70+87i	-64+80i	-50+63i	-33+41i
-416+200i	-631+304i	-810+390i	-880+424i	-810+390i	-631+304i	-416+200i

DEMO

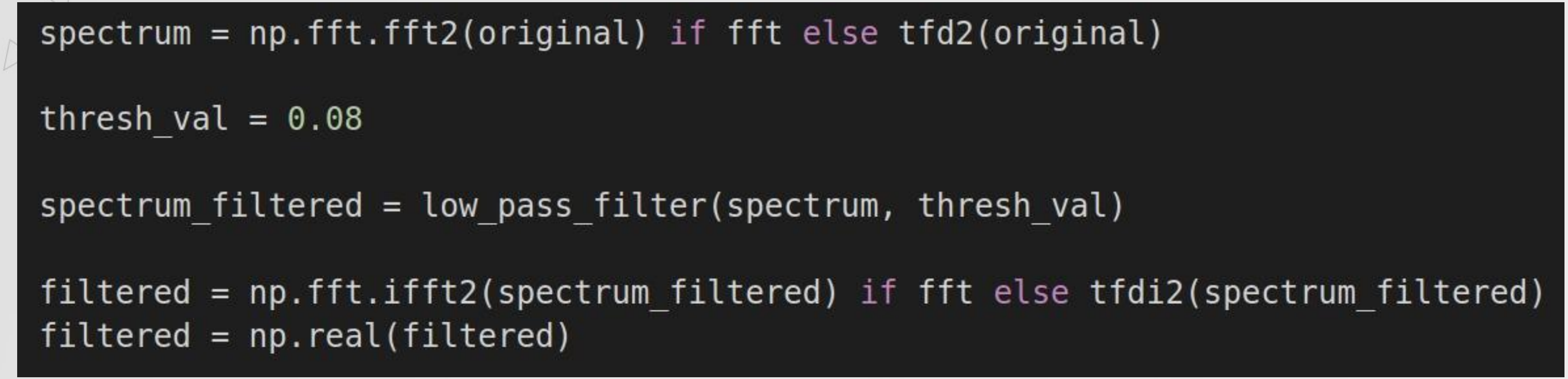
03

NOISE REMOVAL

Product
f(x,y)
Image signal input
DFT
F(u,v)
Image spectrum
G(u,v)
Filtered spectrum
DFTI
g(x,y)
Filtered image output
H(u,v)
Image filter

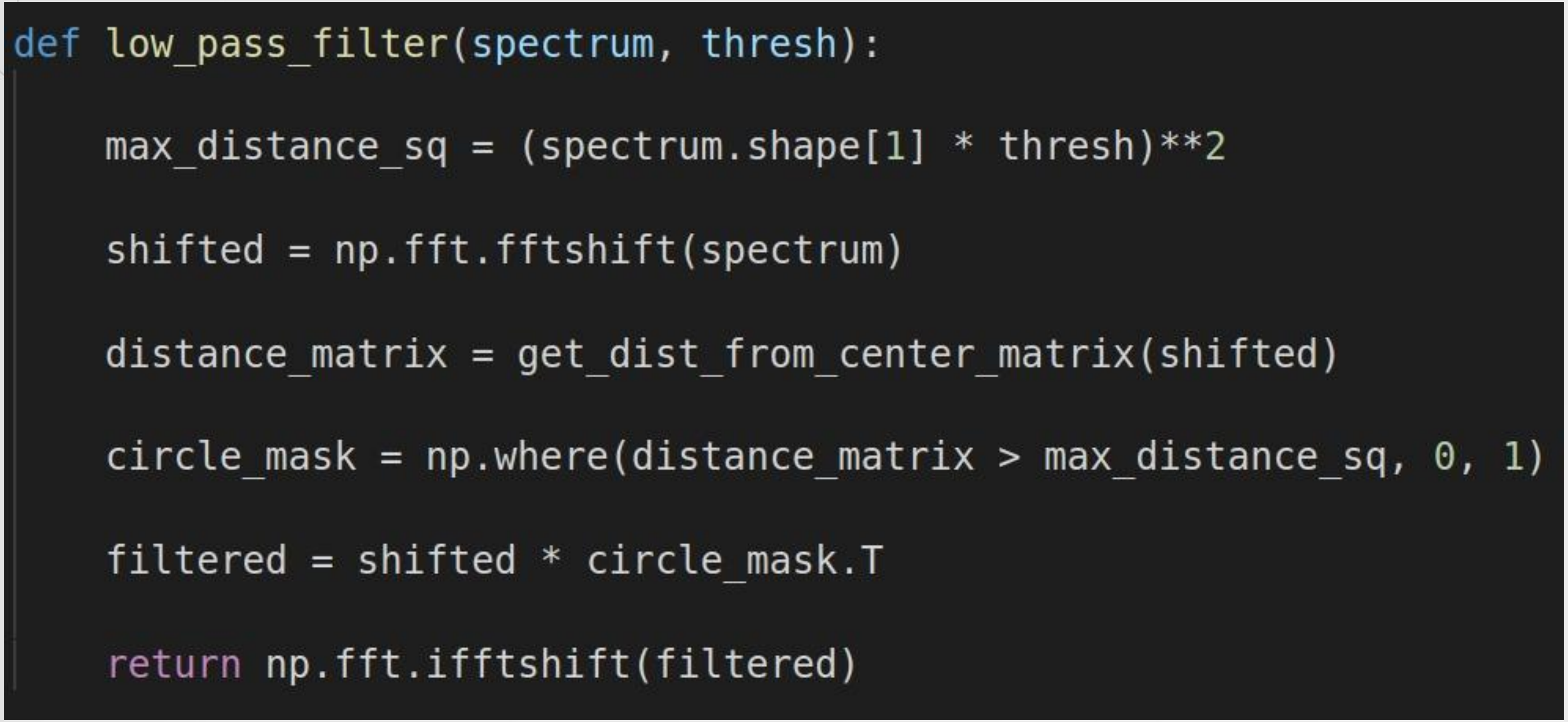
DEMO

Noise



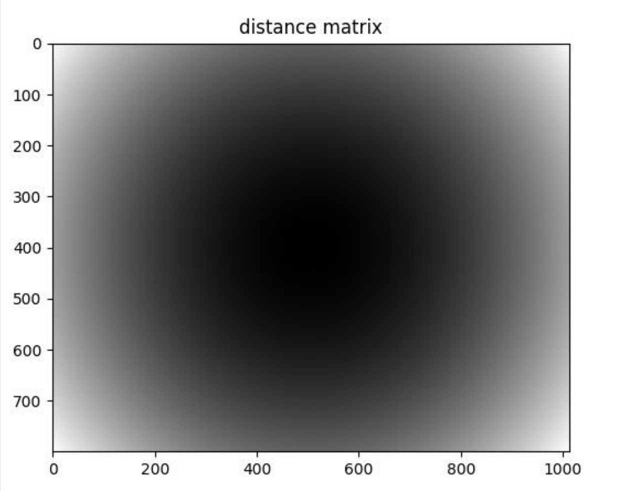
```
spectrum = np.fft.fft2(original) if fft else tfd2(original)

thresh_val = 0.08

spectrum_filtered = low_pass_filter(spectrum, thresh_val)

filtered = np.fft.ifft2(spectrum_filtered) if fft else tfdi2(spectrum_filtered)
filtered = np.real(filtered)
```

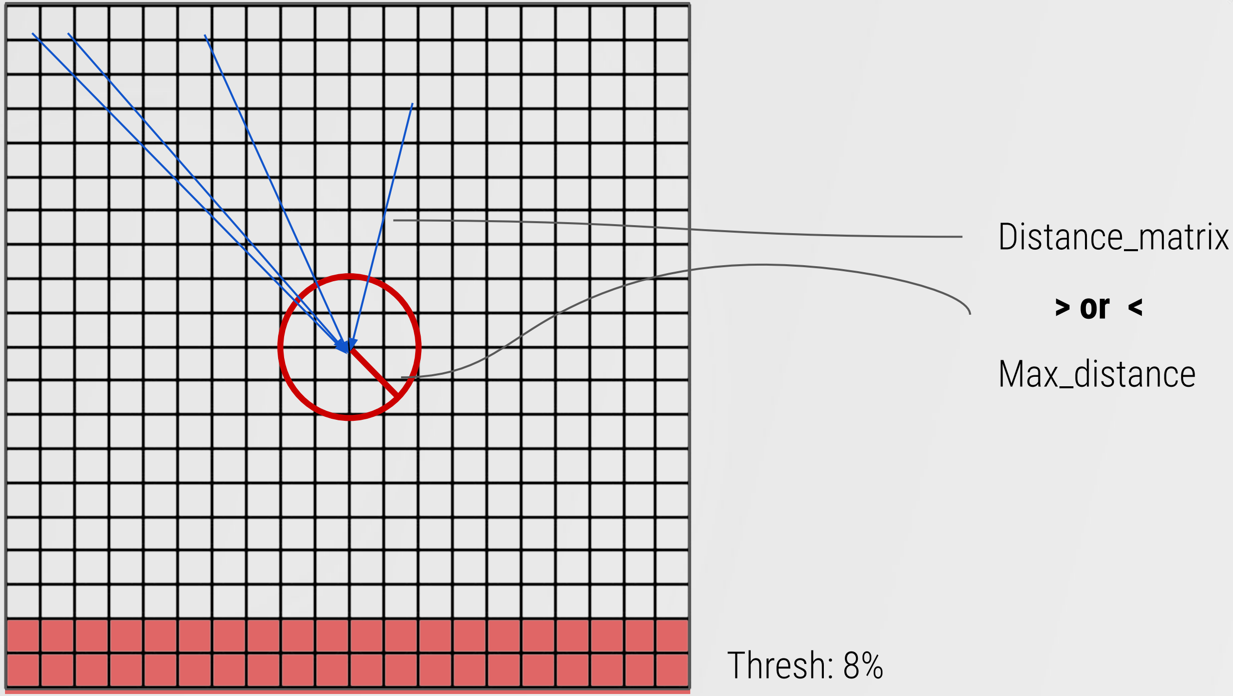
Low_pass_filter



```python
def low_pass_filter(spectrum, thresh):

    max_distance_sq = (spectrum.shape[1] * thresh)**2

    shifted = np.fft.fftshift(spectrum)

    distance_matrix = get_dist_from_center_matrix(shifted)

    circle_mask = np.where(distance_matrix > max_distance_sq, 0, 1)

    filtered = shifted * circle_mask.T

    return np.fft.ifftshift(filtered)
```

Low_pass_filter





Distance_matrix

> or <

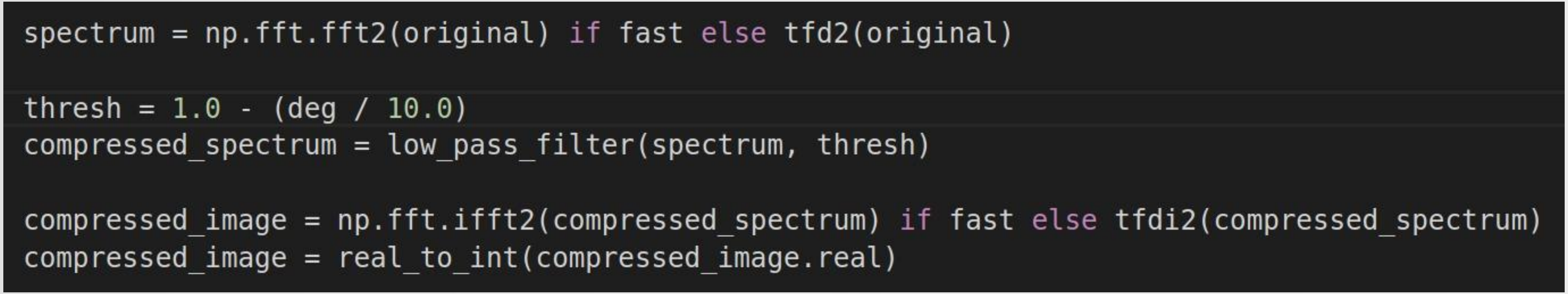
Max_distance

Thresh: 8%

04

COMPRESSION

Compression



```
spectrum = np.fft.fft2(original) if fast else tfd2(original)

thresh = 1.0 - (deg / 10.0)
compressed_spectrum = low_pass_filter(spectrum, thresh)

compressed_image = np.fft.ifft2(compressed_spectrum) if fast else tfdi2(compressed_spectrum)
compressed_image = real_to_int(compressed_image.real)
```

Conclusion