RAPORT

Z góry przepraszam za brak zdjęć w raporcie, ale nie umiałem tego ładnie zrobić, więc jest oddzielny folder ze zdjęciami, nazwy są w przykładowym kodzie programu do konkretnego przykładu.

ROZMYCIE GAUSSA

import scikit\_build\_example as sbe

import numpy as np

H, W = 50, 50

f = 1.0 # nieużywane tutaj, bo robimy obraz punktowy

kernel\_size = 7

sigma = 1.5

#Tworzymy sztuczny obraz 2D: jednoklatkowa biała kropka na czarnym tle

image = np.zeros((H, W), dtype=float)

image[H//2, W//2] = 1.0

img\_list = image.tolist()

#Wywołanie funkcji Gaussian Blur z C++

blur\_full = sbe.gaussian\_blur2d(img\_list, kernel\_size, sigma)

blur\_full\_arr = np.array(blur\_full)

half = kernel\_size // 2

blur\_same = blur\_full\_arr[half:half+H, half:half+W].tolist()

import matplotlib.pyplot as plt

plt.imsave("gauss\_orig.png", image, cmap="gray", vmin=0, vmax=1)

plt.imsave("gauss\_blur.png", np.array(blur\_same), cmap="gray", vmin=0, vmax=1)

print("Zapisano: gauss\_orig.png i gauss\_blur.png")

Generowanie sygnałów o zadanej częstotliwości

Sygnał sinusoidalny i kosinusoidalny:

import scikit\_build\_example as sbe

import numpy as np

# 1) Parametry ogólne

f = 5.0 # częstotliwość (w „jednostkach” – dowolnych)

t\_start = 0.0 # początek przedziału czasowego

t\_end = 1.0 # koniec przedziału (np. 1 sekunda)

N = 500 # liczba próbek

# 2) Generujemy sinusoidę:

# sbe.sine(frequency, t\_start, t\_end, num\_samples) -> List[float]

x\_sin = sbe.sine(f, t\_start, t\_end, N)

# 3) Generujemy kosinusoidę:

x\_cos = sbe.cosine(f, t\_start, t\_end, N)

# 4) Tworzymy wektor czasu w Pythonie (do wykresu):

t = np.linspace(t\_start, t\_end, N)

# 5) Rysujemy i zapisujemy sinusoidę:

sbe.plot\_line(t.tolist(), x\_sin, "sinusoidal\_signal.png")

# 6) Rysujemy i zapisujemy kosinusoidę:

sbe.plot\_line(t.tolist(), x\_cos, "cosine\_signal.png")

print("Zapisano: sinusoidal\_signal.png oraz cosine\_signal.png")  
  
Sygnał prostokątny

import scikit\_build\_example as sbe

import numpy as np

# Parametry

f = 5.0

t\_start = 0.0

t\_end = 1.0

N = 500

duty\_cycle = 0.3 # wypełnienie 30% – proporcja czasu, gdy sygnał = +1

# 1) Generujemy sygnał prostokątny: sbe.square(frequency, t\_start, t\_end, num\_samples, duty\_cycle)

x\_square = sbe.square(f, t\_start, t\_end, N, duty\_cycle)

# 2) Wektor czasu do wykresu

t = np.linspace(t\_start, t\_end, N)

# 3) Rysujemy i zapisujemy:

sbe.plot\_line(t.tolist(), x\_square, "square\_signal.png")

print("Zapisano: square\_signal.png")

# Sygnał piłokształtny

import scikit\_build\_example as sbe

import numpy as np

# Parametry

f = 5.0

t\_start = 0.0

t\_end = 1.0

N = 500

# 1) Generujemy sygnał piłokształtny: sbe.sawtooth(frequency, t\_start, t\_end, num\_samples)

x\_saw = sbe.sawtooth(f, t\_start, t\_end, N)

# 2) Wektor czasu

t = np.linspace(t\_start, t\_end, N)

# 3) Rysujemy i zapisujemy:

sbe.plot\_line(t.tolist(), x\_saw, "sawtooth\_signal.png")

print("Zapisano: sawtooth\_signal.png")

DFT i IDFT

## DFT sygnału sinusoidalnego

import scikit\_build\_example as sbe

import numpy as np

# Parametry

f = 10.0

N = 1000

t = np.linspace(0, 2\*np.pi, N)

# 1) Sygnał w dziedzinie czasu (sin f·t)

x = np.sin(f \* t)

x\_list = x.tolist()

# 2) DFT z C++: zwraca listę liczb zespolonych

X\_complex = sbe.dft(x\_list)

# 3) Moduł^2 (widmo mocy)

X\_mag2 = [abs(z)\*\*2 for z in X\_complex]

# 4) Rysujemy sygnał w dziedzinie czasu:

sbe.plot\_line(t.tolist(), x\_list, "sin\_ft.png")

# 5) Rysujemy widmo |DFT|^2 (oś k = 0..len(X)-1):

sbe.plot\_line(list(range(len(X\_mag2))), X\_mag2, "sin\_ft\_fft.png")

print("Zapisano: sin\_ft.png oraz sin\_ft\_fft.png")

# IDFT

import scikit\_build\_example as sbe

import numpy as np

# Parametry

f = 10.0

N = 1000

t = np.linspace(0, 2\*np.pi, N)

# 1) Sygnał i jego DFT:

x = np.sin(f \* t)

X\_complex = sbe.dft(x.tolist())

# 2) IDFT z C++:

x\_reconstructed = sbe.idft(X\_complex) # lista 1000 wartości

# 3) Rysujemy oryginał i zrekonstruowany:

sbe.plot\_line(t.tolist(), x.tolist(), "orig\_sin.png")

sbe.plot\_line(t.tolist(), x\_reconstructed, "recon\_sin.png")

print("Zapisano: orig\_sin.png i recon\_sin.png")

Filtracja 1D

import scikit\_build\_example as sbe

import numpy as np

# 1) Generujemy sygnał złożony z sumy dwóch sinusoid:

f1, f2 = 5.0, 50.0

N = 1000

t = np.linspace(0, 1.0, N)

x = 0.7\*np.sin(2\*np.pi\*f1\*t) + 0.3\*np.sin(2\*np.pi\*f2\*t)

x\_list = x.tolist()

# 2) Definiujemy prosty filtr uśredniający o długości M:

M = 25

kernel = [1.0/M] \* M # wektor o długości M, suma = 1

# 3) Zastosuj conv1d:

y\_filtered = sbe.conv1d(x\_list, kernel)

# 4) Wykresy:

sbe.plot\_line(t.tolist(), x\_list, "signal\_orig.png")

# Po konwolucji długość = N + M - 1, t trzeba przedłużyć:

t\_extended = np.linspace(0, 1.0 + (M-1)\*(1.0/(N-1)), N+M-1)

sbe.plot\_line(t\_extended.tolist(), y\_filtered, "signal\_filtered.png")

print("Zapisano: signal\_orig.png i signal\_filtered.png")

Filtracja 2D

import scikit\_build\_example as sbe

import numpy as np

import matplotlib.pyplot as plt

# 1) Tworzymy sztuczny obraz 2D: biała kropka na czarnym tle

H, W = 50, 50

image = np.zeros((H, W), dtype=float)

image[25, 25] = 1.0

img\_list = image.tolist()

# 2) Definiujemy prosty kernel 2D (np. filtr uśredniający 5×5)

kernel\_size = 5

kernel\_2d = [[1.0/(kernel\_size\*kernel\_size)]\*kernel\_size for \_ in range(kernel\_size)]

# 3) Wywołanie conv2d z C++:

blur\_full = sbe.conv2d(img\_list, kernel\_2d) # wynik ma rozmiar (H+4)×(W+4)

blur\_full\_arr = np.array(blur\_full)

# 4) Aby uzyskać rozmiar „same” (50×50), wycinamy po 2 piksele z każdej krawędzi:

off = kernel\_size // 2 # = 2

blur\_same = blur\_full\_arr[off:off+H, off:off+W]

# 5) Zapisujemy oryginał i wynik blur jako obrazy przy pomocy Matplotlib:

plt.imsave("image\_orig.png", image, cmap='gray', vmin=0, vmax=1)

plt.imsave("image\_blur.png", blur\_same, cmap='gray', vmin=0, vmax=1)

print("Zapisano: image\_orig.png i image\_blur.png")

WIZUALIZACJA WYKRESÓW 1D

import scikit\_build\_example as sbe

import numpy as np

x = np.linspace(0, 10 \* np.pi, 1000)

y = np.sin(x)

x\_list = x.tolist()

y\_list = y.tolist()

# Teraz spróbuj bez nazwy pliku – powinno otworzyć okno Gnuplota

sbe.plot\_line(x\_list, y\_list)

**Dodanie dwóch liczb**

>>> import scikit\_build\_example as sbe

>>> a = 5.0

>>> b = 6.0

>>> result = sbe.add(a, b)

>>> print(f"{a} + {b} = {result}")

1. + 6.0 = 11.0

>>>