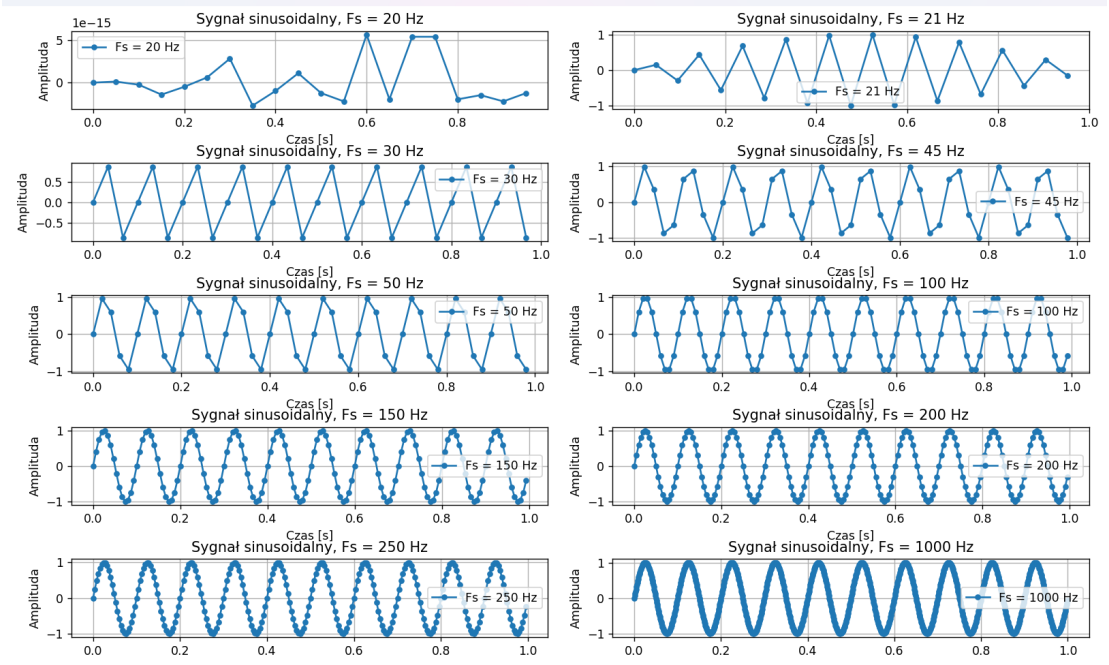
Dmytro Mahaliuk 55722

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

1. Dyskretyzacja

```
def generate_sine_signal(f, fs):
    dt = 1 / fs
    t = np.arange(0, 1, dt)
    s = np.sin(2 * np.pi * f * t)
    return t, s


f = 10
sampling_rates = [20, 21, 30, 45, 50, 100, 150, 200, 250, 1000]

plt.figure(figsize=(12, 10))

for i, fs in enumerate(sampling_rates):
    t, s = generate_sine_signal(f, fs)

    plt.subplot( *args: 5, 2, i + 1)
    plt.plot( *args: t, s, marker='o', linestyle='-', markersize=4, label=f'Fs = {fs} Hz')
    plt.xlabel("Czas [s]")
    plt.ylabel("Amplituda")
    plt.title(f"Sygnał sinusoidalny, Fs = {fs} Hz")
    plt.legend()
    plt.grid()

plt.tight_layout()
plt.show()
```



Twierdzenie **Nyquista-Shannona** (znane też jako **twierdzenie o próbkowaniu**) - aby wiernie odtworzyć sygnał, częstotliwość próbkowania (Fs) musi być co najmniej dwukrotnie większa niż najwyższa częstotliwość obecna w sygnale (f_max)
fs >= 2*f_max

Zjawisko aliasing - Występuje, gdy częstotliwość próbkowania jest zbyt niska (mniejsza niż 2*f_max) co powoduje błędną interpretację sygnału. Aliasing sprawia, że wysokie częstotliwości „nakładają się" na niższe, tworząc zniekształcony obraz sygnału.

```
methods = [None, 'none', 'nearest', 'bilinear', 'bicubic', 'spline16',
           'spline36', 'hanning', 'hamming', 'hermite', 'kaiser', 'quadric',
           'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc', 'lanczos']

image = mpimg.imread('image2.png')

fig, axs = plt.subplots(nrows=3, ncols=6, figsize=(9, 6),
                        subplot_kw={'xticks': [], 'yticks': []})

for ax, interp_method in zip(axs.flat, methods):
    ax.imshow(image, interpolation=interp_method, cmap='viridis')
    ax.set_title(str(interp_method))

plt.tight_layout()
plt.show()
```

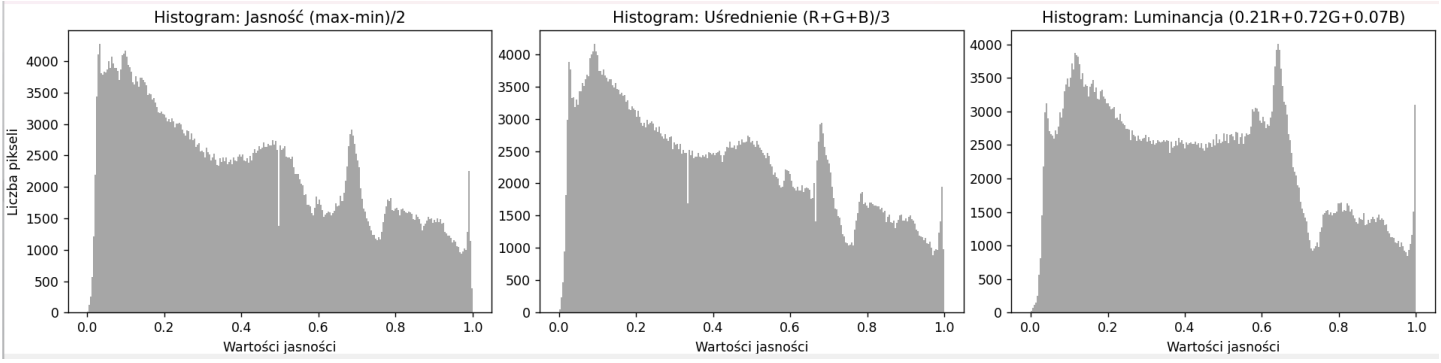
Przejawy aliasingu można wyraźnie zaobserwować na obrazach: ***none, nearest***

2. Kwantyzacja

```
image = mpimg.imread('image2.png')

image_shape = image.shape
print("Wymiary obrazu:", image_shape)

pixel_depth = image.shape[-1]
print("Ilość wartości w pikselu:", pixel_depth)

# Metoda 1: Wyznaczenie jasności piksela: (max(R, G, B) + min(R, G, B)) / 2
gray_max_min = (np.max(image[:, :, :3], axis=2) + np.min(image[:, :, :3], axis=2)) / 2

# Metoda 2: Uśrednienie wartości piksela: (R + G + B) / 3
gray_average = np.mean(image[:, :, :3], axis=2)

# Metoda 3: Wyznaczenie luminancji piksela: 0.21R + 0.72G + 0.07B
gray_luminance = 0.21 * image[:, :, 0] + 0.72 * image[:, :, 1] + 0.07 * image[:, :, 2]

fig2, axs2 = plt.subplots( nrows: 1,  ncols: 3, figsize=(18, 5))

# Obraz z metodą "jasność"
axs2[0].imshow(gray_max_min, cmap='gray')
axs2[0].set_title("Jasność (max-min)/2")
axs2[0].axis('off')

# Obraz z metodą "uśrednienie"
axs2[1].imshow(gray_average, cmap='gray')
axs2[1].set_title("Uśrednienie (R+G+B)/3")
axs2[1].axis('off')

# Obraz z metodą "luminancja"
axs2[2].imshow(gray_luminance, cmap='gray')
axs2[2].set_title("Luminancja (0.21R+0.72G+0.07B)")
axs2[2].axis('off')

plt.tight_layout()
plt.show()
```

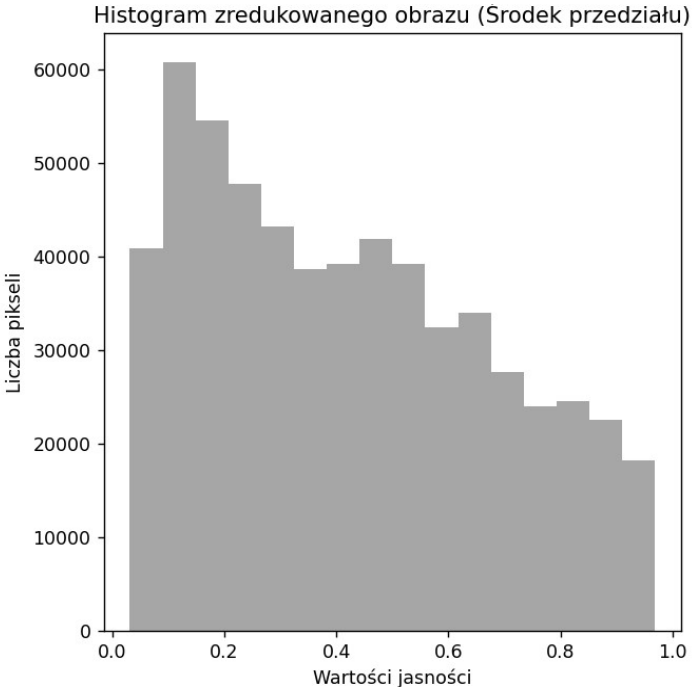
```
Wymiary obrazu: (768, 768, 3)
Ilość wartości w pikselu: 3
```

```python
fig3, axs3 = plt.subplots( nrows: 1,  ncols: 3, figsize=(18, 5))

# Histogram dla obrazu z metodą "jasność"
axs3[0].hist(gray_max_min.flatten(), bins=256, color='gray', alpha=0.7)
axs3[0].set_title("Histogram: Jasność (max-min)/2")
axs3[0].set_xlabel("Wartości jasności")
axs3[0].set_ylabel("Liczba pikseli")

# Histogram dla obrazu z metodą "uśrednienie"
axs3[1].hist(gray_average.flatten(), bins=256, color='gray', alpha=0.7)
axs3[1].set_title("Histogram: Uśrednienie (R+G+B)/3")
axs3[1].set_xlabel("Wartości jasności")

# Histogram dla obrazu z metodą "luminancja"
axs3[2].hist(gray_luminance.flatten(), bins=256, color='gray', alpha=0.7)
axs3[2].set_title("Histogram: Luminancja (0.21R+0.72G+0.07B)")
axs3[2].set_xlabel("Wartości jasności")

plt.tight_layout()
plt.show()
```




```python
hist, bin_edges = np.histogram(gray_average.flatten(), bins=16, range=(0, 1))

bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

gray_average_reduced = np.digitize(gray_average, bin_edges) - 1
gray_average_reduced = np.clip(gray_average_reduced,  a_min: 0,  a_max: 15)
new_image = bin_centers[gray_average_reduced]

fig4, axs4 = plt.subplots( nrows: 1,  ncols: 2, figsize=(10, 6))

axs4[0].imshow(new_image, cmap='gray')
axs4[0].set_title("Obraz zredukowany do 16 kolorów (Środek przedziału)")
axs4[0].axis('off')

axs4[1].hist(new_image.flatten(), bins=16, color='gray', alpha=0.7)
axs4[1].set_title("Histogram zredukowanego obrazu (Środek przedziału)")
axs4[1].set_xlabel("Wartości jasności")
axs4[1].set_ylabel("Liczba pikseli")

plt.tight_layout()
plt.show()
```

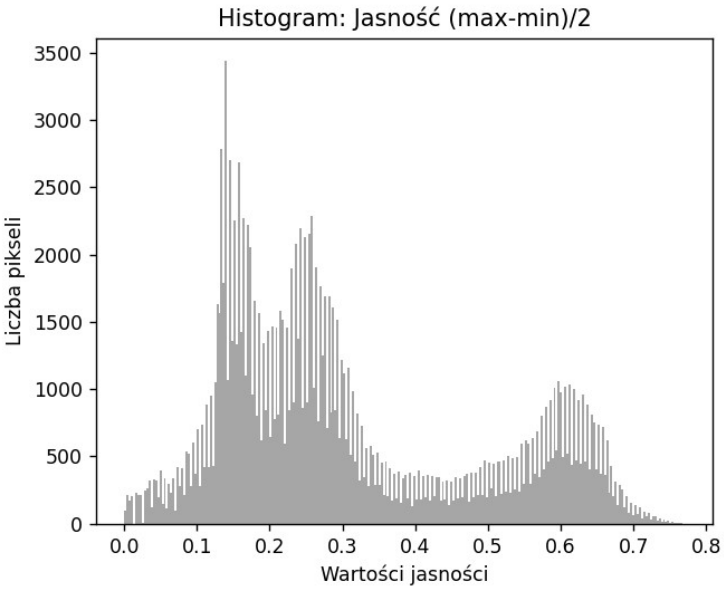
Obraz zredukowany do 16 kolorów (Środek przedziału)



3. Binaryzacja

```
image = mpimg.imread('image3.png')

gray_max_min = (np.max(image[:, :, :3], axis=2) + np.min(image[:, :, :3], axis=2)) / 2

fig2, axs2 = plt.subplots( nrows: 1,  ncols: 2, figsize=(10, 5))
axs2[0].imshow(gray_max_min, cmap='gray')
axs2[0].set_title("Jasność (max-min)/2")
axs2[0].axis('off')
axs2[1].hist(gray_max_min.flatten(), bins=256, color='gray', alpha=0.7)
axs2[1].set_title("Histogram: Jasność (max-min)/2")
axs2[1].set_xlabel("Wartości jasności")
axs2[1].set_ylabel("Liczba pikseli")
plt.tight_layout()
plt.show()
```

Jasność (max-min)/2



```python
def find_threshold_with_min_distance(hist, bin_edges, min_value=0, min_distance=0.0):
    initial_mean = (bin_edges[0] + bin_edges[1]) / 2
    bin_edges_new = np.array([initial_mean + i * 2 * initial_mean for i in range(len(bin_edges) - 1)])

    valid_indices = hist >= min_value
    filtered_hist = hist[valid_indices]
    filtered_edges = bin_edges_new[valid_indices]

    final_peaks = []
    final_peak_values = []

    for i in range(len(filtered_edges)):
        if not final_peaks:
            final_peaks.append(filtered_edges[i])
            final_peak_values.append(filtered_hist[i])
        else:
            if filtered_edges[i] - final_peaks[-1] < min_distance:
                if filtered_hist[i] > final_peak_values[-1]:
                    final_peaks[-1] = filtered_edges[i]
                    final_peak_values[-1] = filtered_hist[i]
            else:
                final_peaks.append(filtered_edges[i])
                final_peak_values.append(filtered_hist[i])

    threshold = np.mean(final_peaks) if final_peaks else None

    return threshold

hist, bin_edges = np.histogram(gray_max_min.flatten(), bins=256, range=(0, 1))

threshold = find_threshold_with_min_distance(hist, bin_edges, min_value: 900, min_distance: 0.3)

print(f"Wyznaczony punkt progowania: {threshold:.4f}")

binary_image = gray_max_min > threshold

plt.figure(figsize=(6, 6))
plt.imshow(binary_image, cmap='gray')
plt.title(f"Binarizacja obrazu z progiem: {threshold:.4f}")
plt.axis('off')
plt.show()
```