| Wydział Informatyki Politechniki Białostockiej<br>Przedmiot: Przetwarzanie Sygnałów i Obrazów | Data: 30.11.2021 |
| --- | --- |
| Zajęcia nr 5<br>Temat: Systemy liniowe<br><br>Grupa: PS6<br>Imię i nazwisko: Marek Mierzwiński | Prowadzący:<br>mgr inż Patryk Milewski |

**Wprowadzenie**

System liniowy to system dla którego przepuszczona przez niego suma sygnałów o dowolnych liczbowych współczynnikach będzie taka sama jak suma przepuszczonych przez niego sygnałów o tych samych współczynnikach: $T\{ax_1 + bx_2\} = aT\{x_1\} + bT\{x_2\}$

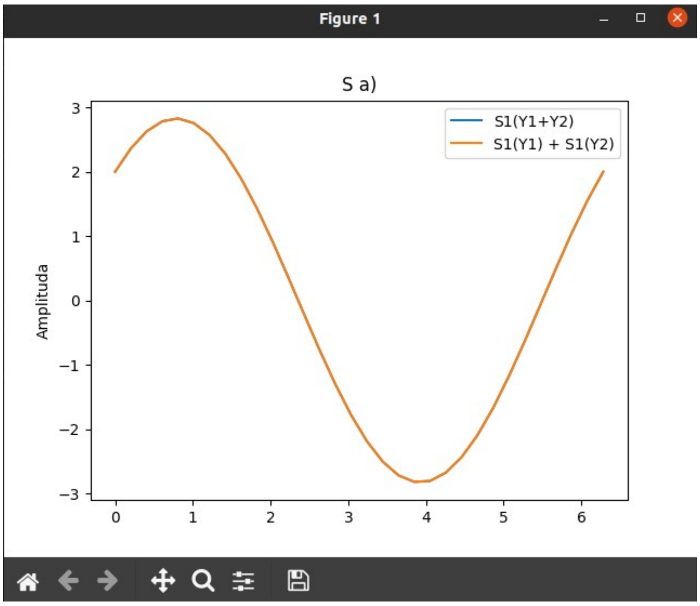
System liniowy, który jest niezmienniczy w czasie to jest daje wciąż taką samą odpowiedź na sekwencję jednostkową nazywamy systemem LTI.

Do realizacji zadań wykorzystano język python, użyto bibliotek:
**-numpy** do wielu zastosowań (manipulacja tablicami, generowanie sygnałów, etc)
**-scipy** do dostępu do funkcji rfft
**-matplotlib** do reprezentacji graficznej sygnałow
**-librosa** do otwarcia plików wav
**-sounddevice** do odtworzenia dźwięku

**Zad. 1**

Zakładając, że x[n] – dowolna sekwencja wejściowa, sprawdzić analitycznie liniowość systemów opisanych następującymi równaniami:
a) S{x[n]} = 2x[n]
b) S{x[n]} = x[n] + 1
c) S{x[n]} = x[n + 1] − x[n]
Wygenerować dwa dowolne sygnały dyskretne $x_1[n]$, $x_2[n]$ (po 32 próbki każdy). Zweryfikować empirycznie liniowość (lub nieliniowość) systemów (a-c), porównując na wykresach odpowiedź sumy sygnałów $S\{x_1[n] + x_2[n]\}$ z sumą odpowiedzi $S\{x_1[n]\} + S\{x_2[n]\}$. Co możesz powiedzieć o przyczynowości analizowanych systemów?

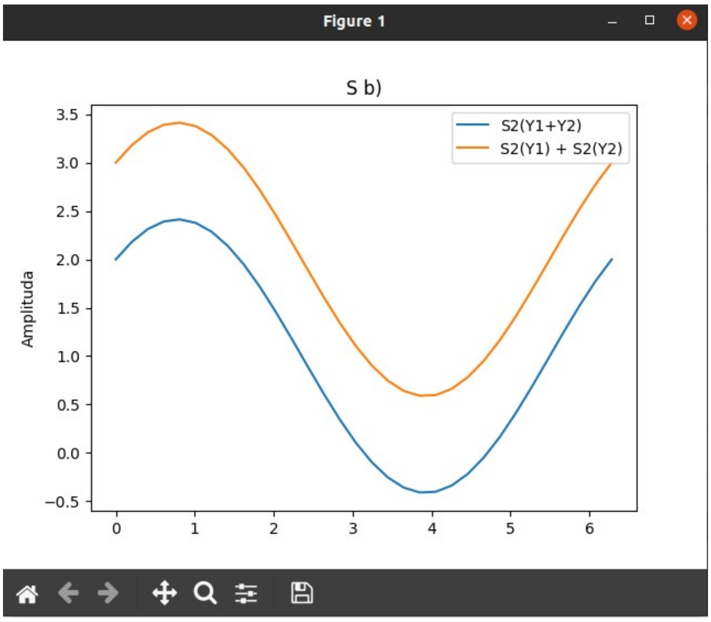
```
import matplotlib.pyplot as plt
import numpy as np

X = np.linspace(0,2*np.pi,32)
Y1 = np.sin(X)
Y2 = np.cos(X)

def S1(x):
   return 2*x
def S2(x):
   return x+1
def S3(x):
   for i in range(len(x)-1):
      x[i]=x[i+1]-x[i]
   return(x)

plt.plot(X,S1(Y1+Y2),label='S1(Y1+Y2)')
plt.plot(X, S1(Y1) + S1(Y2),label='S1(Y1) + S1(Y2)')
plt.legend()
plt.title("S a)")
plt.ylabel("Amplituda")
```

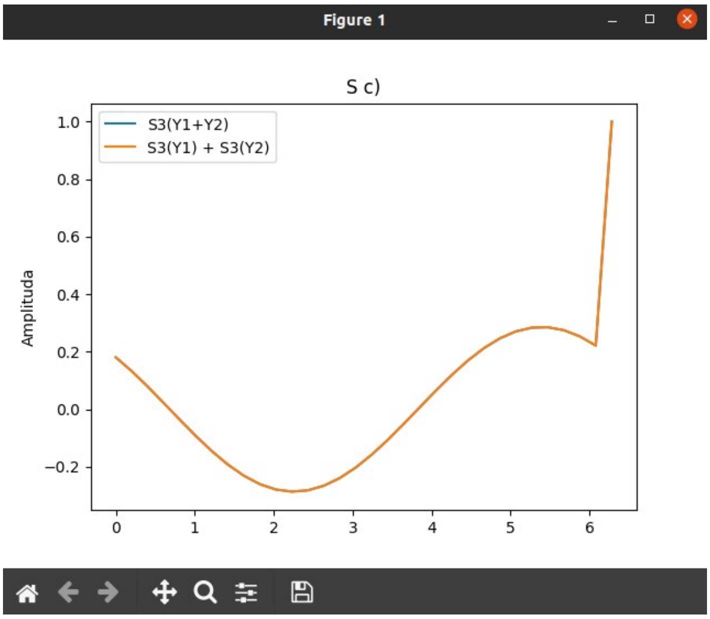


```python
plt.show()
plt.plot(X, S2(Y1 + Y2),label='S2(Y1+Y2)')
plt.plot(X, S2(Y1) + S2(Y2),label='S2(Y1) + S2(Y2)')
plt.legend()
plt.title("S b)")
plt.ylabel("Amplituda")
plt.show()
plt.plot(X, S3(Y1 + Y2),label='S3(Y1+Y2)')
plt.plot(X, S3(Y1) + S3(Y2),label='S3(Y1) + S3(Y2)')
plt.legend()
plt.title("S c)")
plt.ylabel("Amplituda")
plt.show()
```

Wyniki z systemów a i c się pokrywają, są one zatem liniowe, system b nie jest liniowy.

Przyczynowym jest tylko system c, ponieważ jego odpowiedź zależy od wejść przyszłych.

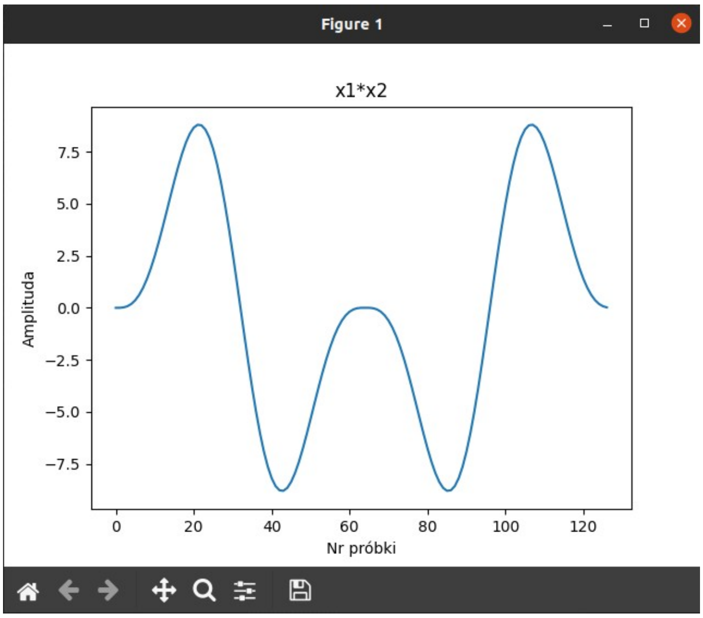


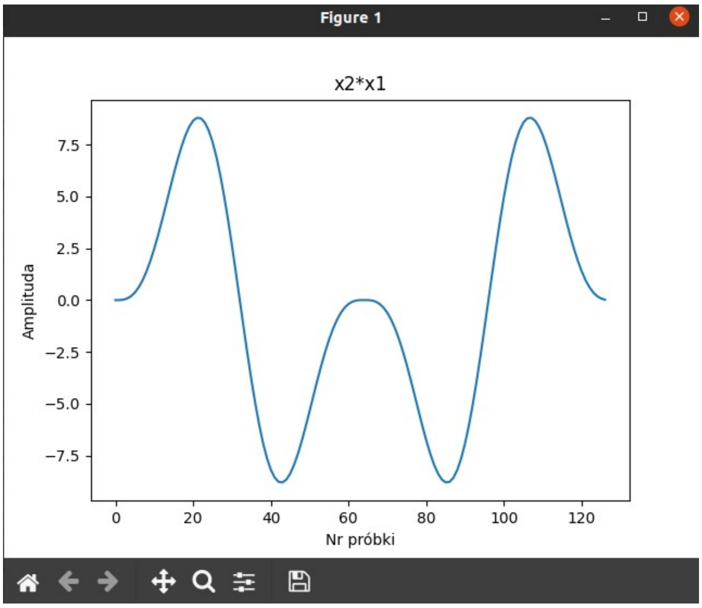

**Zad. 2**

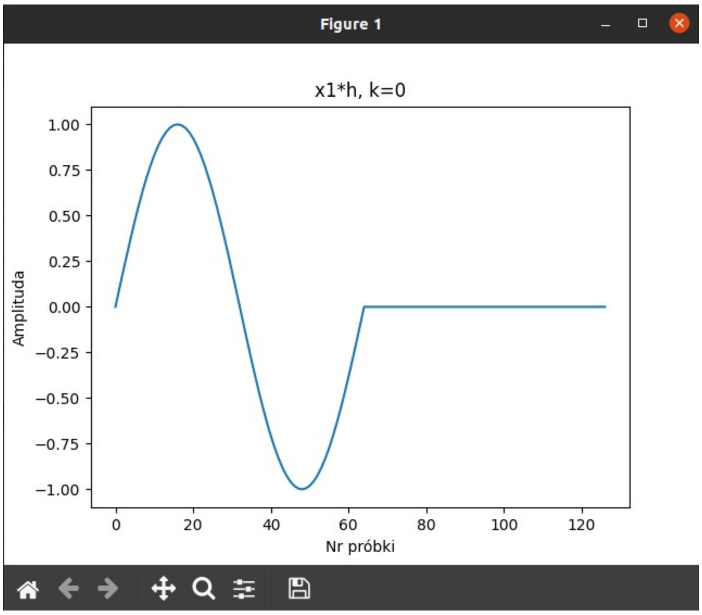
Wygenerować sygnały $x_1[n] = \sin(2\pi n/N)$, $x_2[n] = \sin(4\pi n/N)$ oraz $h[n] = \delta[n - k]$, gdzie $k = \{0, 16, 32\}$, $N = 64$, (założyć, że $0 <= n < N$ ). Wyznaczyć splot liniowy sygnałów $x_1[n]$, $x_2[n]$ z sygnałem $h[n]$ oraz samych ze sobą. Sporządzić wykresy, sprawdzić czy operacje splotu są przemienne oraz liniowe (dla ustalonego sygnału $h[n]$).

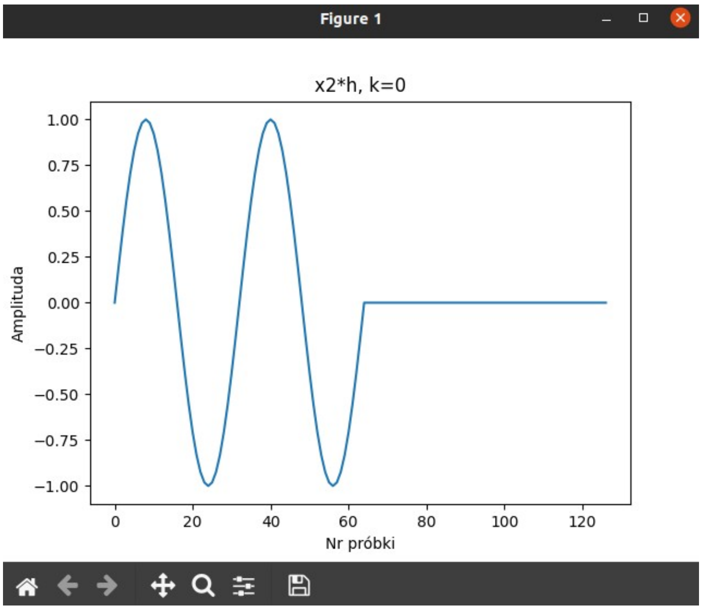
```python
import matplotlib.pyplot as plt
import numpy as np

X = np.array(range(64))
Y1 = np.sin(np.pi*X/32)
Y2 = np.sin(np.pi * X / 16)
H = np.zeros(64)

plt.plot(np.convolve(Y1,Y2))
plt.title('x1*x2')
plt.xlabel('Nr próbki')
plt.ylabel('Amplituda')
plt.show()
plt.plot(np.convolve(Y2, Y1))
plt.title('x2*x1')
plt.xlabel('Nr próbki')
plt.ylabel('Amplituda')
plt.show()
```

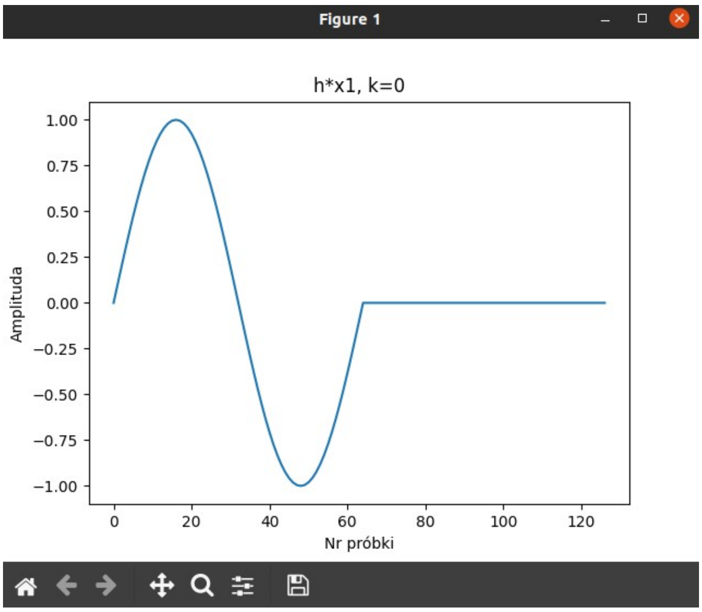


```
for i in range(3):
    H[i*16] = 1
    plt.plot(np.convolve(Y1, H))
    plt.title(('x1*h, k='+str(i*16)))
    plt.xlabel('Nr próbki')
    plt.ylabel('Amplituda')
    plt.show()
    plt.plot(np.convolve(H, Y1))
    plt.title('h*x1, k='+str(i * 16))
    plt.xlabel('Nr próbki')
    plt.ylabel('Amplituda')
    plt.show()
    plt.plot(np.convolve(Y2, H))
    plt.title('x2*h, k='+str(i * 16))
    plt.xlabel('Nr próbki')
    plt.ylabel('Amplituda')
    plt.show()
    plt.plot(np.convolve(H, Y2))
    plt.title('h*x2, k='+str(i * 16))
    plt.xlabel('Nr próbki')
    plt.ylabel('Amplituda')
    plt.show()
    plt.plot(np.convolve(np.convolve(Y1, Y2),H))
    plt.title('(x1*x2)*h, k='+str(i * 16))
    plt.xlabel('Nr próbki')
    plt.ylabel('Amplituda')
    plt.show()
    plt.plot(np.convolve(Y1,np.convolve(Y2, H)))
    plt.title('x1*(x2*h), k='+str(i * 16))
    plt.xlabel('Nr próbki')
    plt.ylabel('Amplituda')
    plt.show()
    H[i * 16] = 0
```



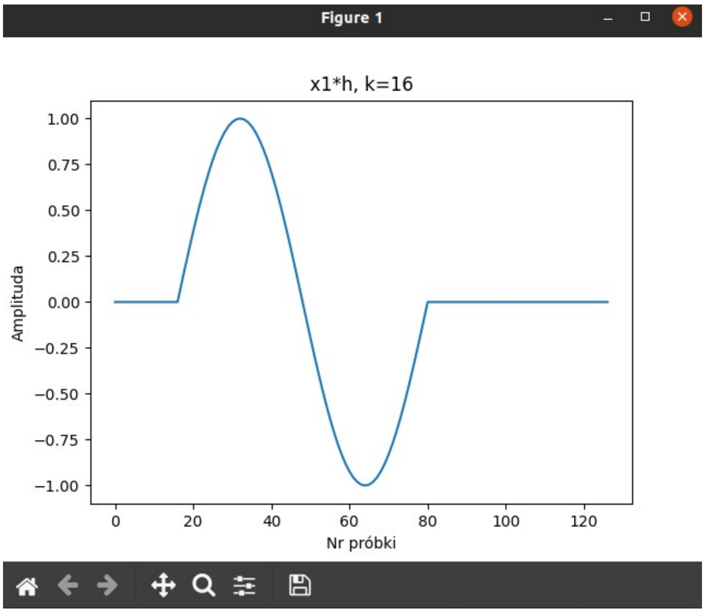


Figure 1
x1*h, k=16
Amplituda
Nr próbki

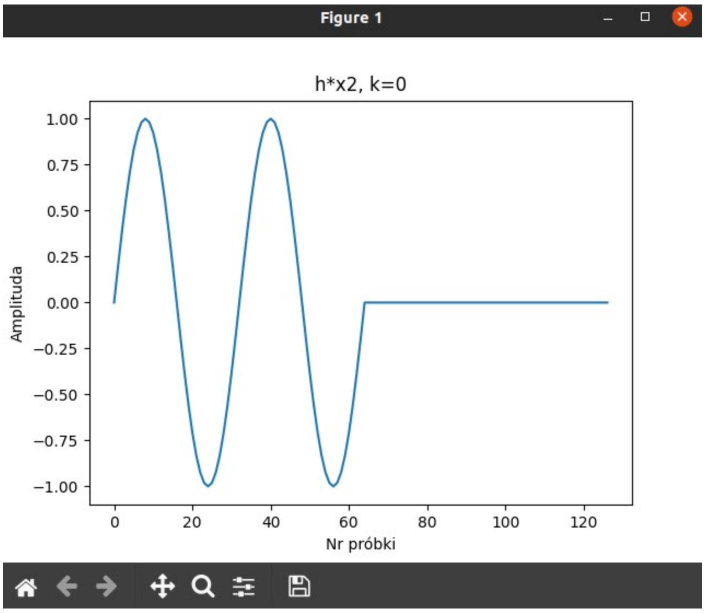


Figure 1
h*x2, k=0
Amplituda
Nr próbki

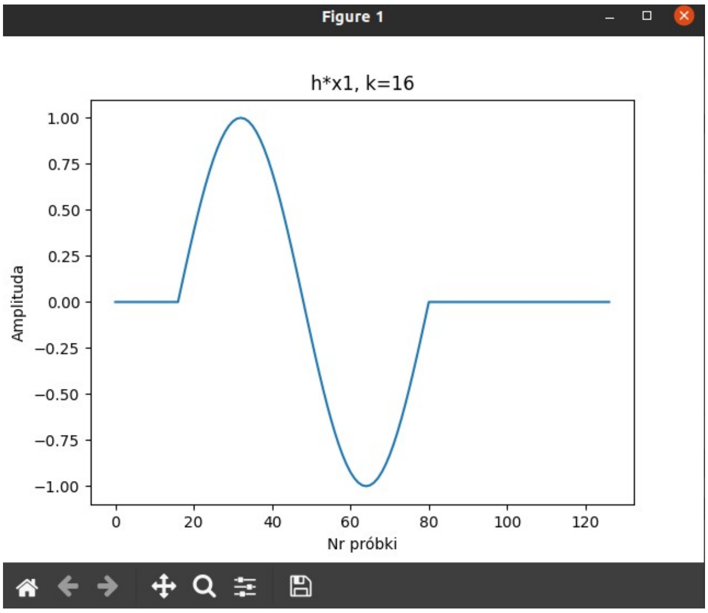


Figure 1
h*x1, k=16
Amplituda
Nr próbki

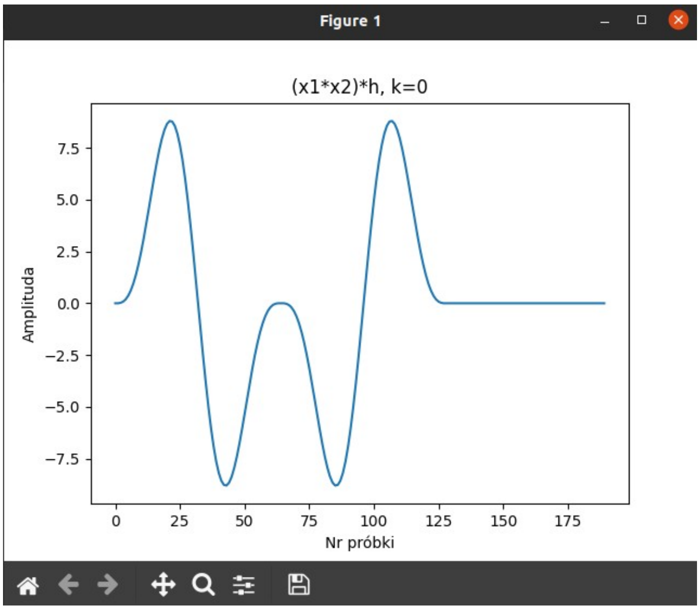


Figure 1
(x1*x2)*h, k=0
Amplituda
Nr próbki

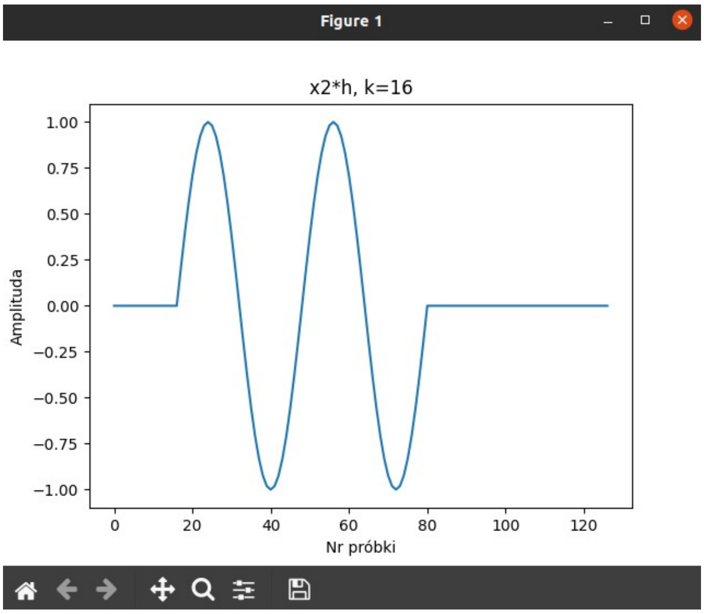


Figure 1
x2*h, k=16
Amplituda
Nr próbki

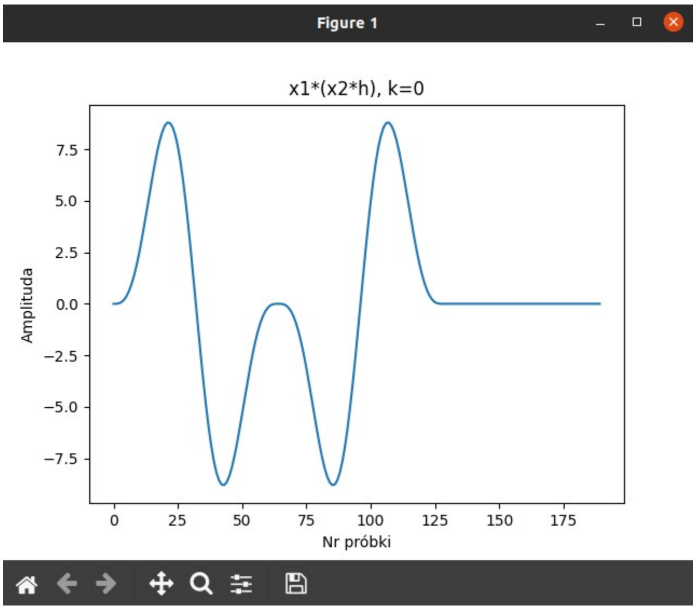


Figure 1
x1*(x2*h), k=0
Amplituda
Nr próbki

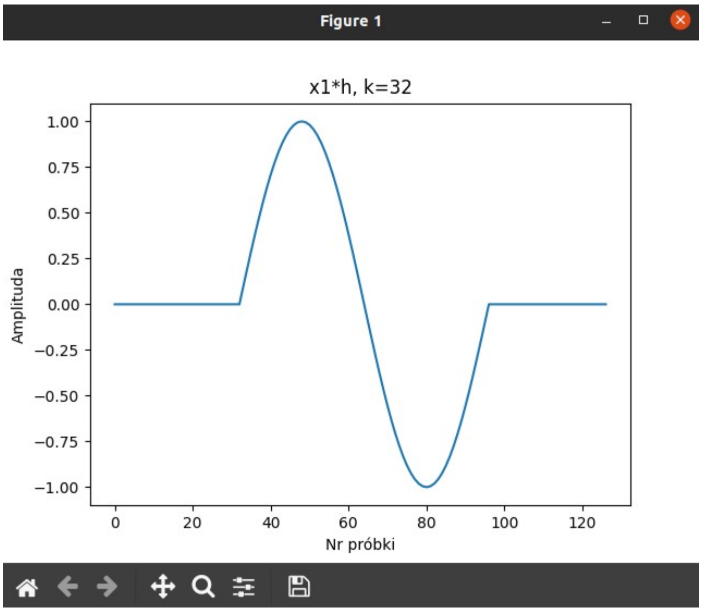


Figure 1
x1*h, k=32
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Amplitude
0
20
40
60
80
100
120
Nr próbki

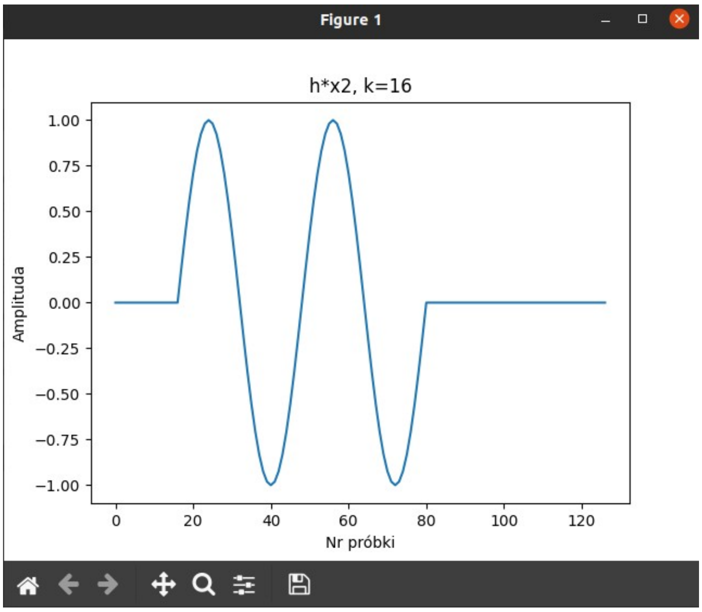



Figure 1
h*x2, k=16
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Amplitude
0
20
40
60
80
100
120
Nr próbki

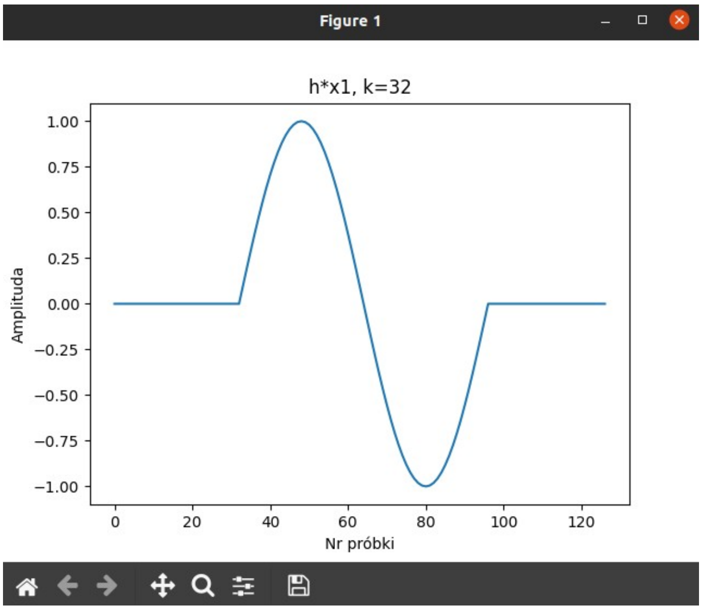



Figure 1
h*x1, k=32
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Amplitude
0
20
40
60
80
100
120
Nr próbki

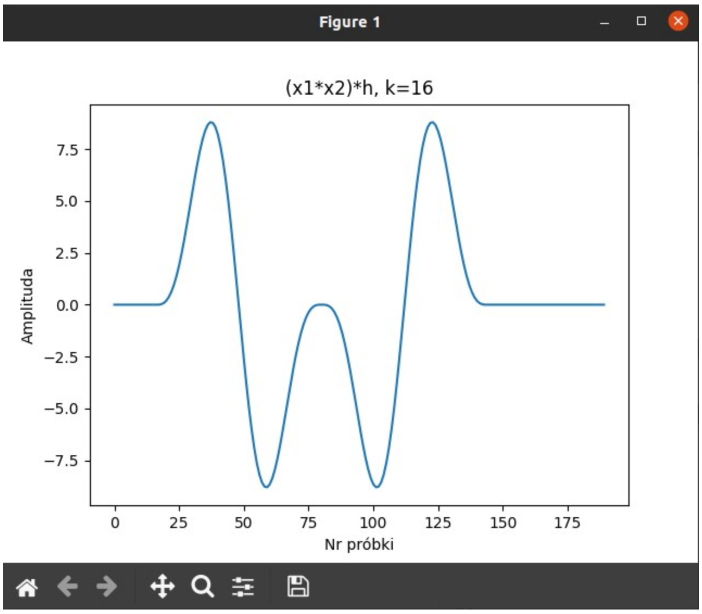



Figure 1
(x1*x2)*h, k=16
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
Amplitude
0
25
50
75
100
125
150
175
Nr próbki

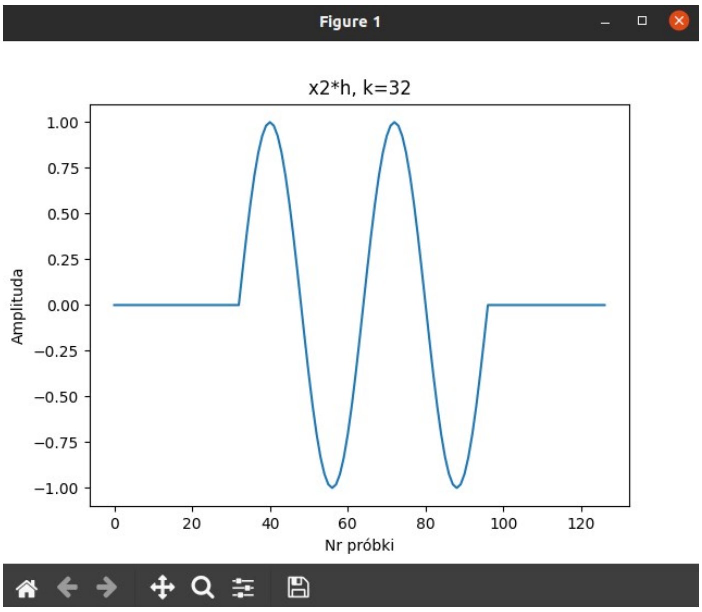



Figure 1
x2*h, k=32
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Amplitude
0
20
40
60
80
100
120
Nr próbki

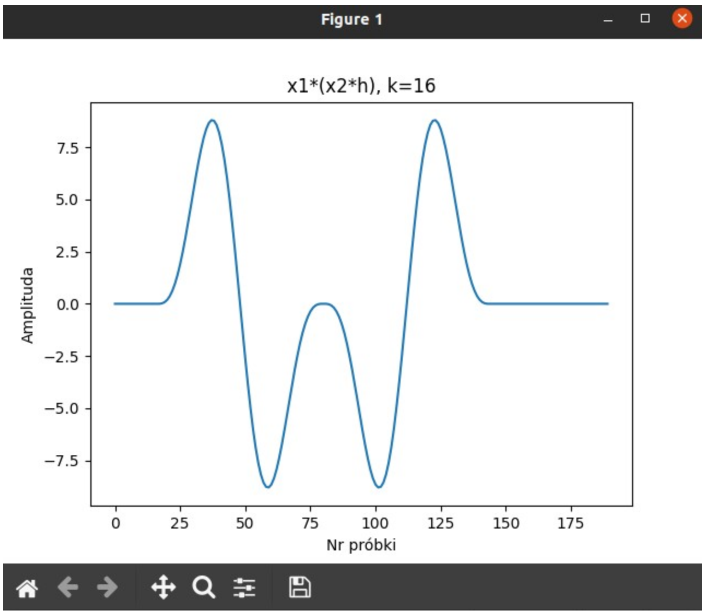
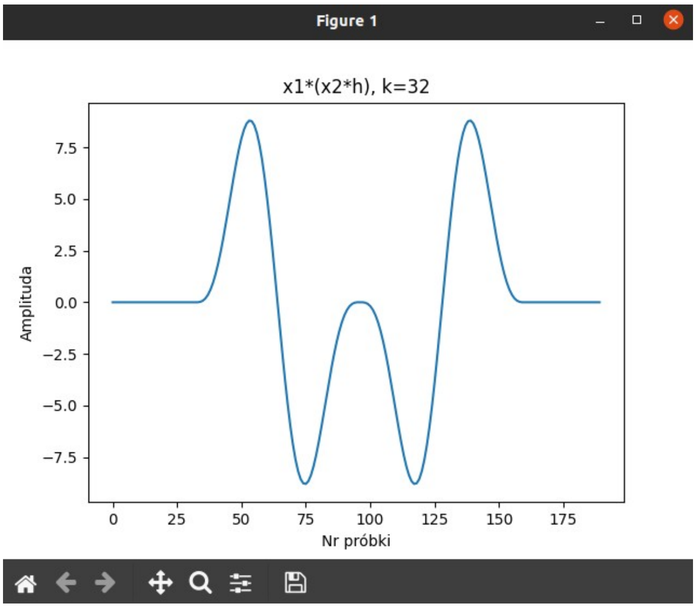
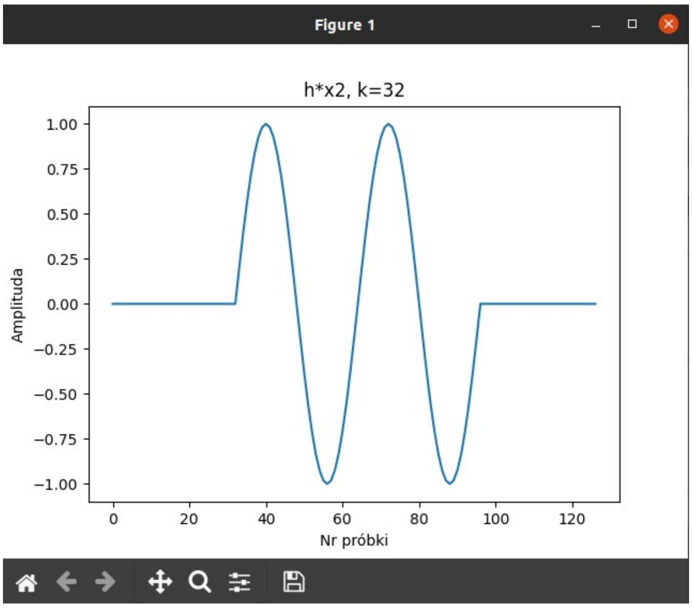
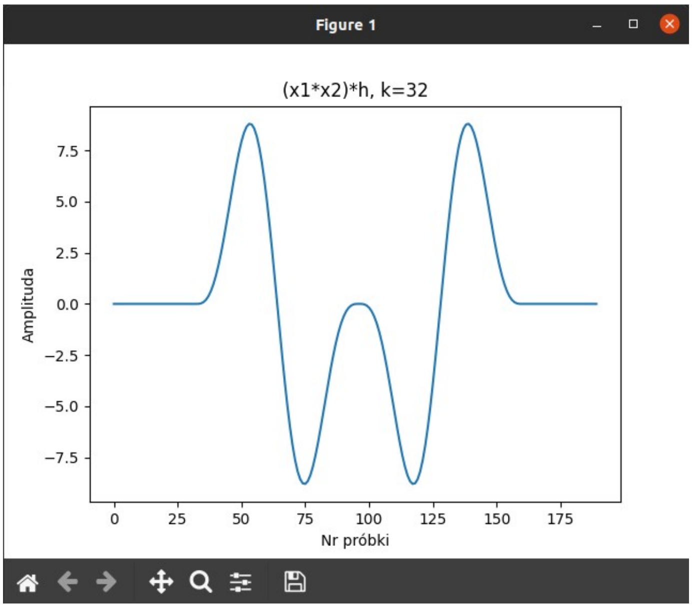



Figure 1
x1*(x2*h), k=16
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
Amplitude
0
25
50
75
100
125
150
175
Nr próbki



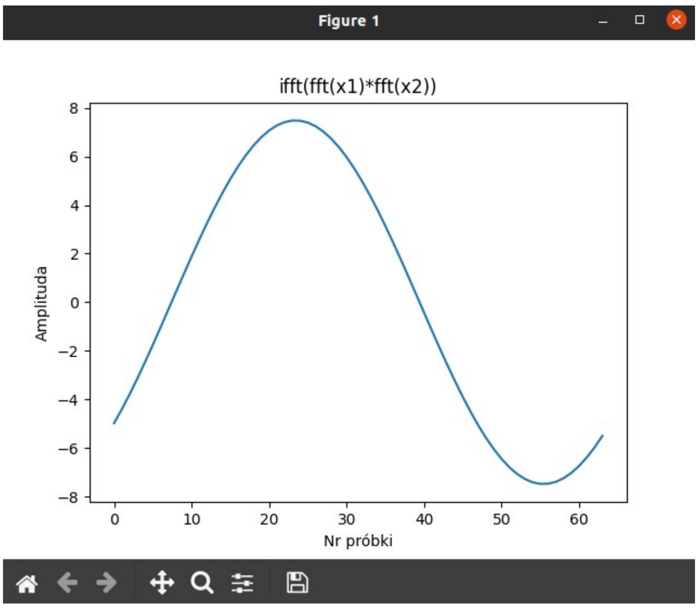

Dla wszystkich sygnałów h splot był przemienny I liniowy



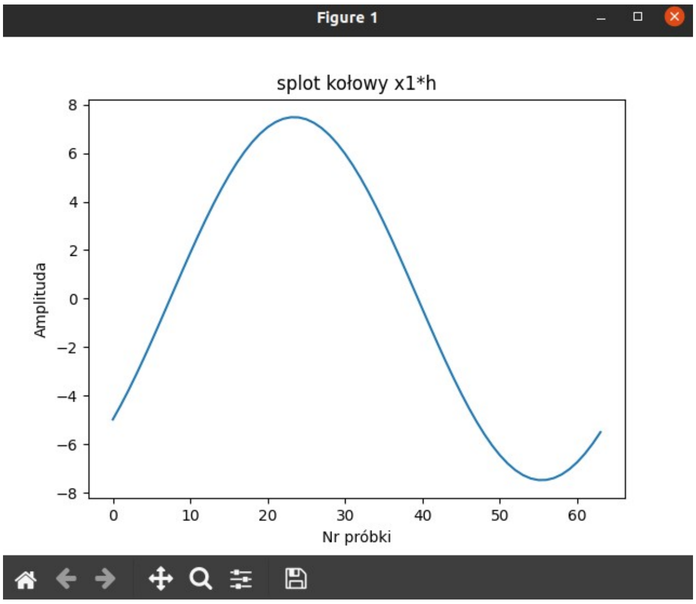

**Zad. 3**

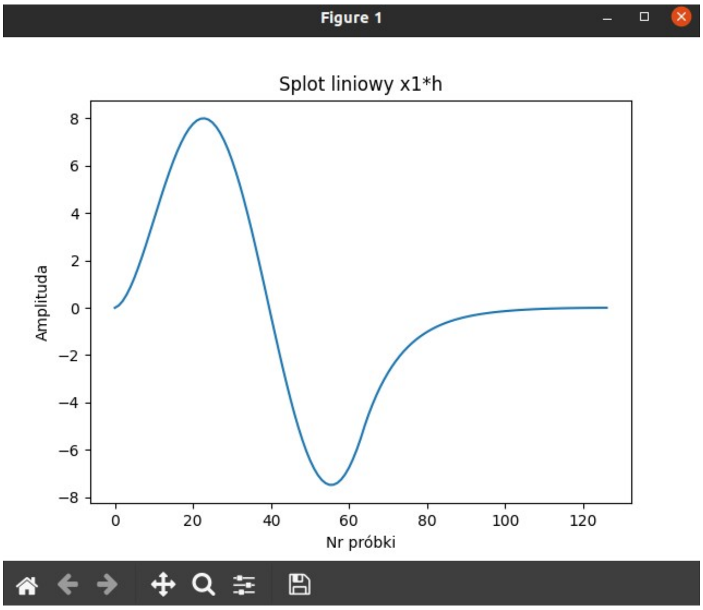
Wyznaczyć 64-punktowe DFT sygnału $x_1[n]$ z zadania 5.2 oraz sygnału $h[n] = \exp(-n/10)$, obliczyć iloczyn widm zespolonych (tj. $G(k) = X_1(k) \cdot H(k)$ dla $k = 0, 1, \ldots, 63$), wyznaczyć IDFT iloczynu $G(k)$. Uzyskany wynik porównać ze splotem liniowym (64-punktowym) sygnałów $x_1[n]$ i $h[n]$.

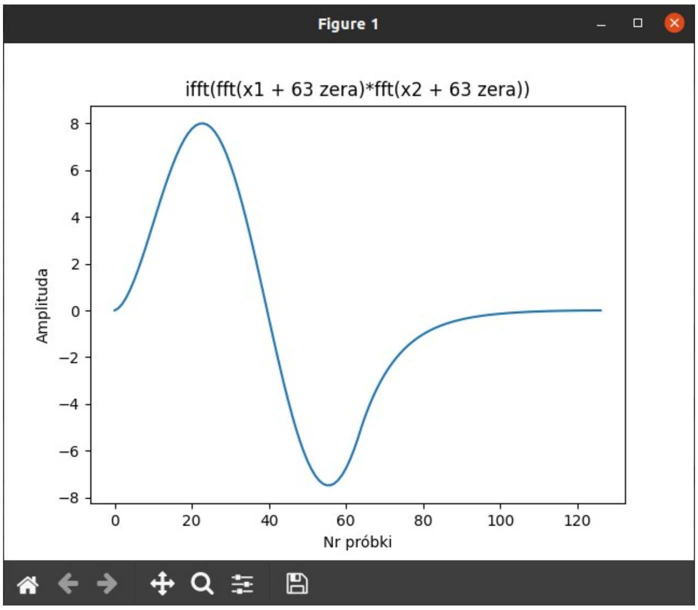
```
import matplotlib.pyplot as plt
import numpy as np

X = np.array(range(64))
Y1 = np.sin(np.pi * X / 32)
H = np.exp(-X/10)
Y1d = np.fft.fft(Y1)
Hd = np.fft.fft(H)
G = Y1d*Hd
Gl = np.fft.fft(np.concatenate([Y1,np.zeros(63)]))\
    *np.fft.fft(np.concatenate([H,np.zeros(63)]))
plt.plot(np.fft.ifft(G))
plt.title('ifft(fft(x1)*fft(x2))')
plt.xlabel('Nr próbki')
plt.ylabel('Amplituda')
plt.show()
plt.plot(np.convolve(Y1, np.concatenate((H[1:], H)),\
'valid'))
plt.title('splot kołowy x1*h')
plt.xlabel('Nr próbki')
```



```python
plt.ylabel('Amplituda')
plt.show()
plt.plot(np.real(np.fft.ifft(GI)))
plt.title('ifft(fft(x1 + 63 zera)*fft(x2 + 63 zera))')
plt.xlabel('Nr próbki')
plt.ylabel('Amplituda')
plt.show()
plt.plot(np.convolve(Y1, H, 'full'))
plt.title('Splot liniowy x1*h')
plt.xlabel('Nr próbki')
plt.ylabel('Amplituda')
plt.show()
```



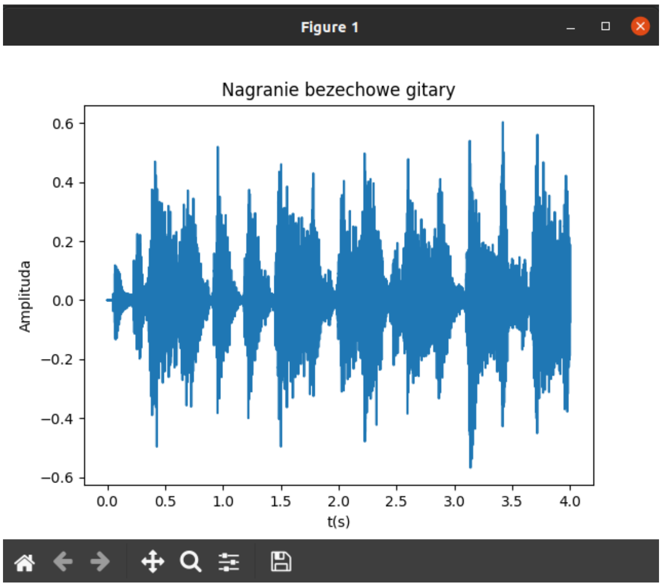

Splot liniowy ma długość N+M-1 gdzie N i M to długości splatanych sygnałów. Niemożliwe więc jest uzyskanie splotu liniowego długości 64 z dwóch sygnałów długości 64. Wykonano splot kołowy, pokrywa się on z odwróconą transformatą iloczynu widm. Splot liniowy natomiast pokrywa się z odwróconą transformatą iloczynu widm rozszerzonych zerami do długości splotu.

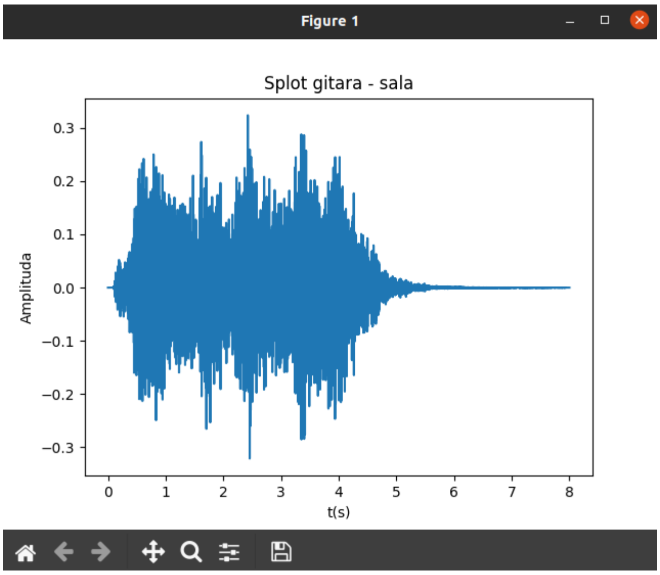
**Zad. 4**

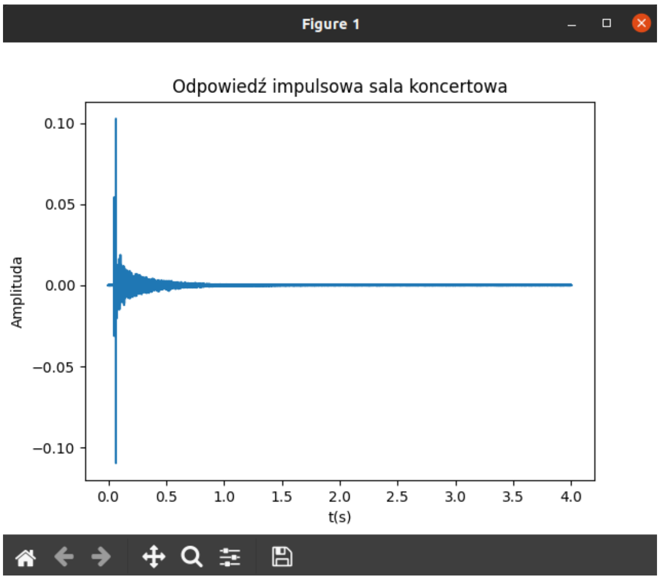
Dokonać splotu nagrania dźwiękowego dokonanego w komorze bezechowej x[n] (ang. anechoic chamber) z akustyczną odpowiedzią impulsową dwóch dowolnie wybranych pomieszczeń h[n] (sala koncertowa, korytarz, itp.). Jak różnią się słuchowo poszczególne sygnały (x[n], h[n] oraz ich splot)?

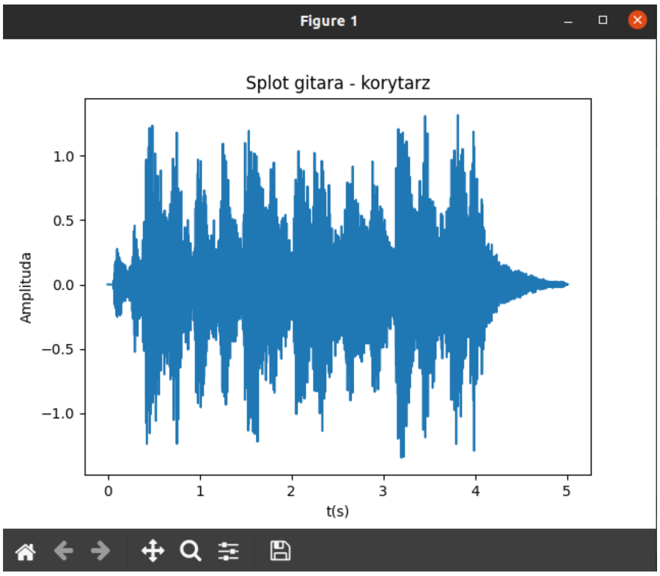
```python
import matplotlib.pyplot as plt
import numpy as np
import sounddevice as sd
import librosa

x = np.linspace(0, 4, 4*44100)
X, srx = librosa.load("./Guitar_Anechoic_4s.wav", 44100)
Y, sry = librosa.load("./s3_r3_o_4s.wav", 44100)
Z, srz = librosa.load("./outdoors.wav", 44100)
G = np.convolve(X,Y)
H = np.convolve(X,Z)
sd.play(X, 44100)
plt.plot(x,X)
```

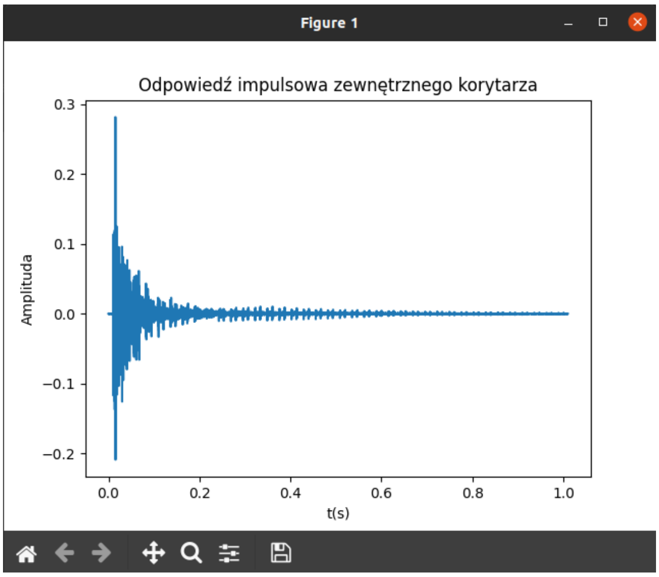
```python
plt.title("Nagranie bezechowe gitary")
plt.xlabel('t(s)')
plt.ylabel('Amplituda')
plt.show()
sd.play(Y, 44100)
plt.plot(x,Y)
plt.title("Odpowiedź impulsowa sala koncertowa")
plt.xlabel('t(s)')
plt.ylabel('Amplituda')
plt.show()
sd.play(Z, 44100)
plt.plot(np.linspace(0,len(Z)/44100,len(Z)), Z)
plt.title("Odpowiedź impulsowa zewnętrznego korytarza")
plt.xlabel('t(s)')
plt.ylabel('Amplituda')
plt.show()
sd.play(G, 44100)
plt.title("Splot gitara - sala")
plt.xlabel('t(s)')
plt.ylabel('Amplituda')
plt.plot(np.linspace(0,(len(G)/44100),len(G)),G)
plt.show()
sd.play(H, 44100)
plt.title("Splot gitara - korytarz")
plt.xlabel('t(s)')
plt.ylabel('Amplituda')
plt.plot(np.linspace(0, (len(H) / 44100), len(H)), H)
plt.show()
```



Nagranie gitary jest czyste, bez echa. Nagrania odpowiedzi impulsowej zawierają jedno krótkie uderzenie i jego echo. Splot sygnału gitary z odpwiedzią sali koncertowej daje wrażenie silnego echa, oraz oddalenia źródła dźwięku. Splot z sygnałem korytarza daje mniejsze wrażenie echa i odbierana odległość od źrodła dźwięku jest bliższa.

**Zad. 5**

Przeprowadzić rozmycie Gaussa oraz wyostrzenie na dowolnym obrazie za pomocą operacji splotu na oknach 3 na 3.

- W jaki sposób można rozwiązać problem wartości brzegowych?
- Jakie obrazy należy wstępnie przetwarzać przed operacjami splotu?
- Przeprowadzić wykrywanie krawędzi za pomocą splotu z dowolną maską wykrywającą krawędzie. (na przykład Sobel, Prewitt, Laplaciany)
- Jak wyostrzanie, rozmywanie i wykrywanie krawędzi wpływa na składowe FFT obrazu?

```
import matplotlib.pyplot as plt
import matplotlib.image as img
import numpy as np
import scipy.fftpack as fp

def imgfft(image):
    Fourier = np.empty(image.shape, dtype='complex128')
    Fourier[:,:,0] = fp.rfft(fp.rfft(image[:, :, 0], axis=0), axis=1)
    Fourier[:,:,1] = fp.rfft(fp.rfft(image[:, :, 1], axis=0), axis=1)
    Fourier[:,:,2] = fp.rfft(fp.rfft(image[:, :, 2], axis=0), axis=1)
    wyn = np.log2(np.abs(Fourier))
    wyn[:,:,0] = wyn[:,:,0]-wyn[:,:,0].min()
    wyn[:,:,0] = wyn[:,:,0]/wyn[:,:,0].max()
    wyn[:, :, 1] = wyn[:, :, 1] - wyn[:, :, 1].min()
    wyn[:, :, 1] = wyn[:, :, 1] / wyn[:, :, 1].max()
    wyn[:, :, 2] = wyn[:, :, 2] - wyn[:, :, 2].min()
    wyn[:, :, 2] = wyn[:, :, 2] / wyn[:, :, 2].max()
    return(wyn)


image = img.imread('image2.jpg')
x, y, z = np.shape(image)
ax = x-1
ay = y-1
conv = np.zeros((x,y,z),dtype=image.dtype)
kernel = np.array([[1, 2, 1],
          [2, 4, 2],
          [1, 2, 1]])
for i in range(z):
    kernel = kernel / 9
    for j in range(2):
        for c in range(2):
            conv[0][0][i] += kernel[j][c]*image[1-j][1-c][i]
            conv[0][ay][i] += kernel[j][c+1] * image[1 - j][ay - c][i]
            conv[ax][0][i] += kernel[j+1][c] * image[ax - j][1 - c][i]
            conv[ax][ay][i] += kernel[j+1][c+1] * image[ax - j][ay - c][i]
    kernel = kernel*3/4
    for j in range(1,ay):
        for a in range(2):
            for b in range(3):
                conv[0][j][i] += kernel[a][b]*image[1-a][j+1-b][i]
                conv[ax][j][i] += kernel[a+1][b]*image[ax-1+a][j+1-b][i]
    for j in range(1,ax):
        for a in range(3):
            for b in range(2):
                conv[j][0][i] += kernel[a][b]*image[j+1-a][1-b][i]
                conv[j][ay][i] += kernel[a][b+1]*image[j+1-a][ay-1+b][i]
    kernel = kernel*3/4
    for j in range(1,ax):
        for c in range(1, ay):
```

```
            for a in range(3):
                for b in range(3):
                    conv[j][c][i] += kernel[a][b]*image[j+1-a][c+1-b][i]
    kernel = kernel*16

plt.imshow(image)
plt.title("obraz")
plt.show()

plt.imshow(imgfft(image))
plt.title("FFT obrazu")
plt.show()

plt.imshow(conv)
plt.title("obraz rozmyty")
plt.show()
plt.imshow(imgfft(conv))
plt.title("FFT rozmytego obrazu")
plt.show()

conv = np.zeros((x, y, z), dtype="float64")
kernel = np.array([[0, -1, 0],
                   [-1, 4, -1],
                   [0, -1, 0]])/8

for i in range(z):
    for j in range(2):
        for c in range(2):
            conv[0][0][i] += kernel[j][c] * image[1 - j][1 - c][i]
            conv[0][ay][i] += kernel[j][c + 1] * image[1 - j][ay - c][i]
            conv[ax][0][i] += kernel[j + 1][c] * image[ax - j][1 - c][i]
            conv[ax][ay][i] += kernel[j + 1][c + 1] * image[ax - j][ay - c][i]

    for j in range(1, ay):
        for a in range(2):
            for b in range(3):
                conv[0][j][i] += kernel[a][b] * image[1 - a][j + 1 - b][i]
                conv[ax][j][i] += kernel[a + 1][b] * image[ax - 1 + a][j + 1 - b][i]
    for j in range(1, ax):
        for a in range(3):
            for b in range(2):
                conv[j][0][i] += kernel[a][b] * image[j + 1 - a][1 - b][i]
                conv[j][ay][i] += kernel[a][b + 1] * image[j + 1 - a][ay - 1 + b][i]

    for j in range(1, ax):
        for c in range(1, ay):
            for a in range(3):
                for b in range(3):
                    conv[j][c][i] += kernel[a][b] * image[j + 1 - a][c + 1 - b][i]
conv = conv+255/2
plt.imshow(image)
plt.title("obraz")
plt.show()
plt.imshow(conv.astype(int))
plt.title("wykrywanie krawędzi")
plt.show()
plt.imshow(imgfft(conv))
plt.title("FFT splotu")
plt.show()
```

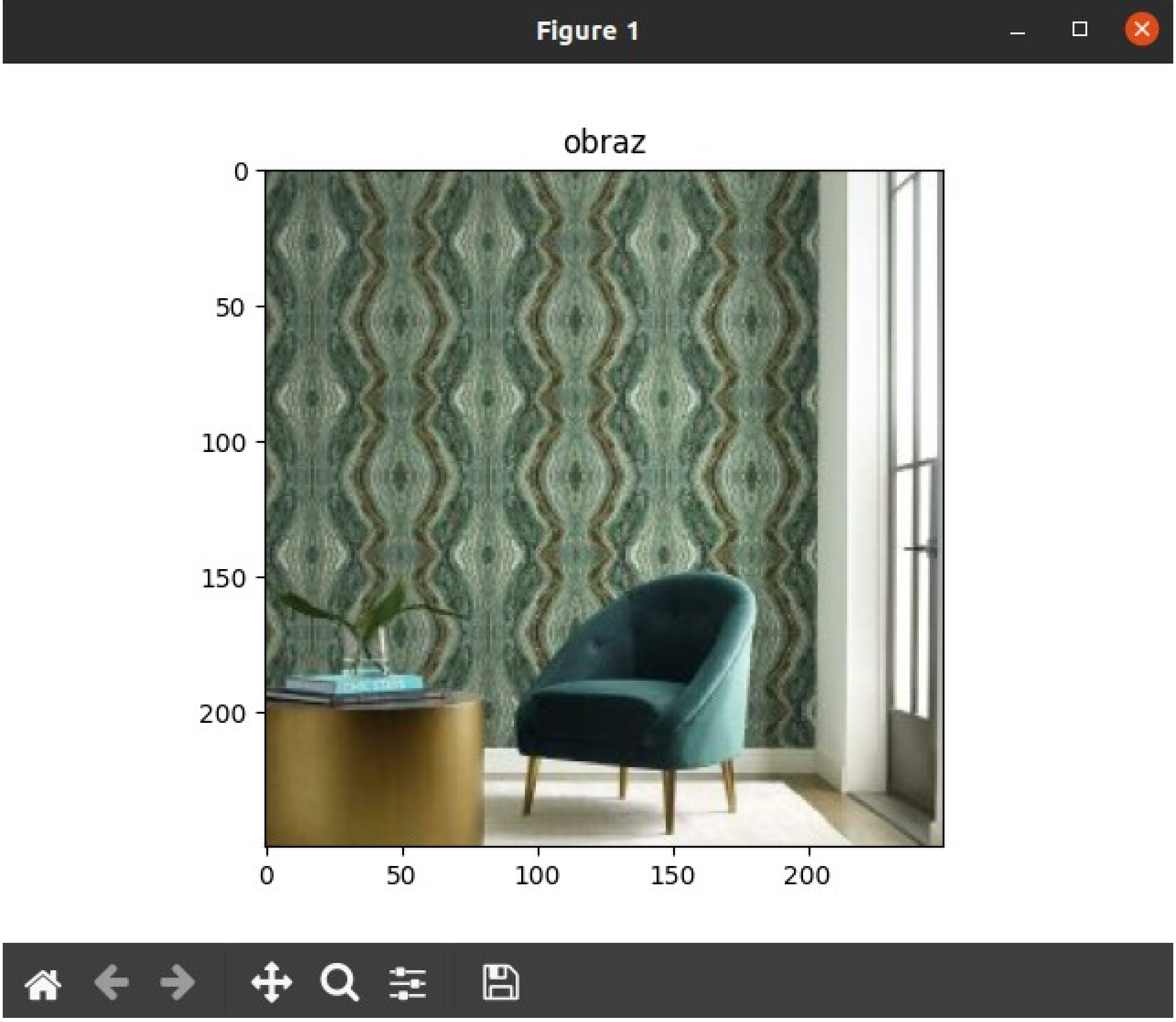


Figure 1
obraz
0
50
100
150
200
0
50
100
150
200

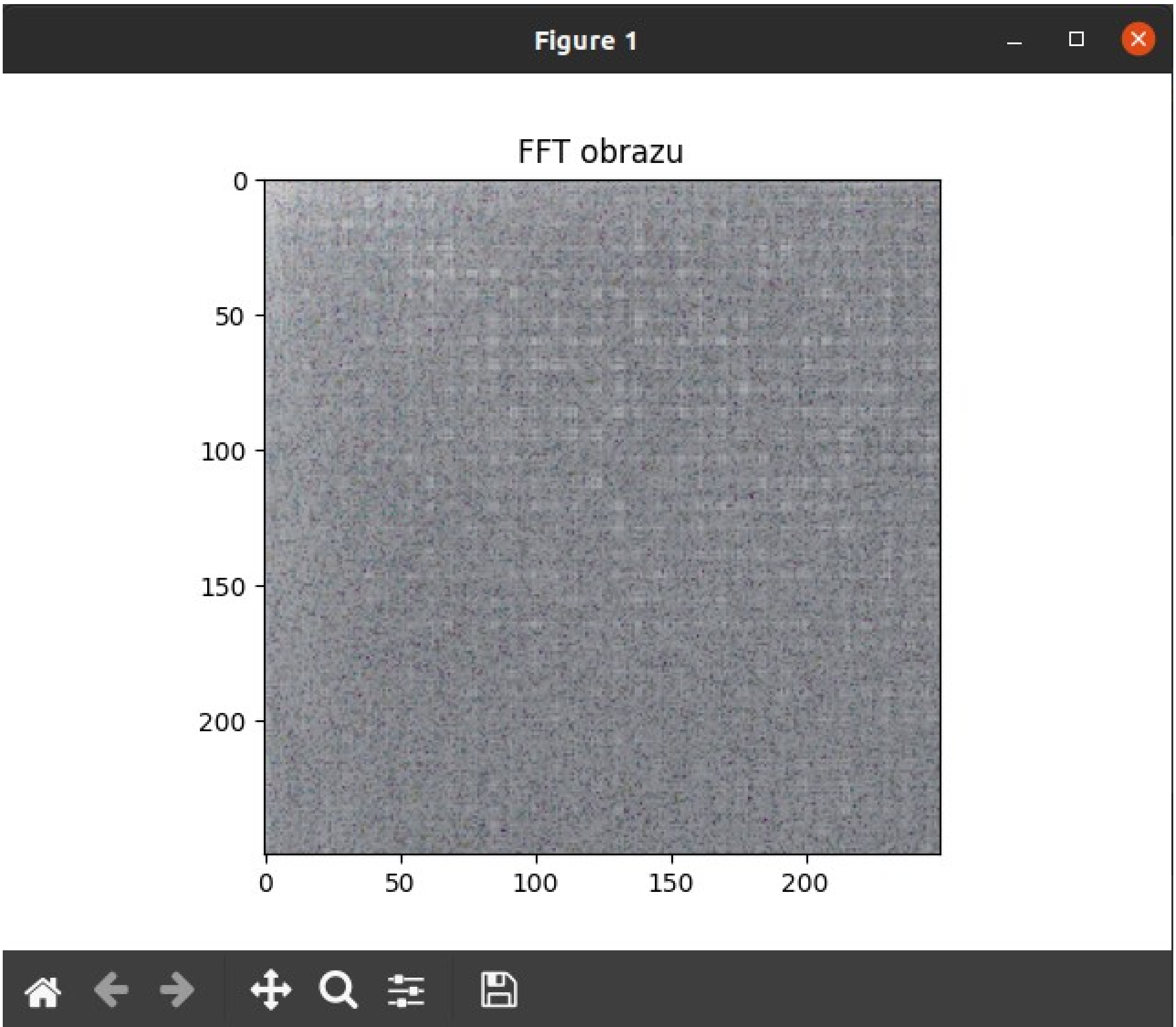


Figure 1
FFT obrazu
0
50
100
150
200
0
50
100
150
200

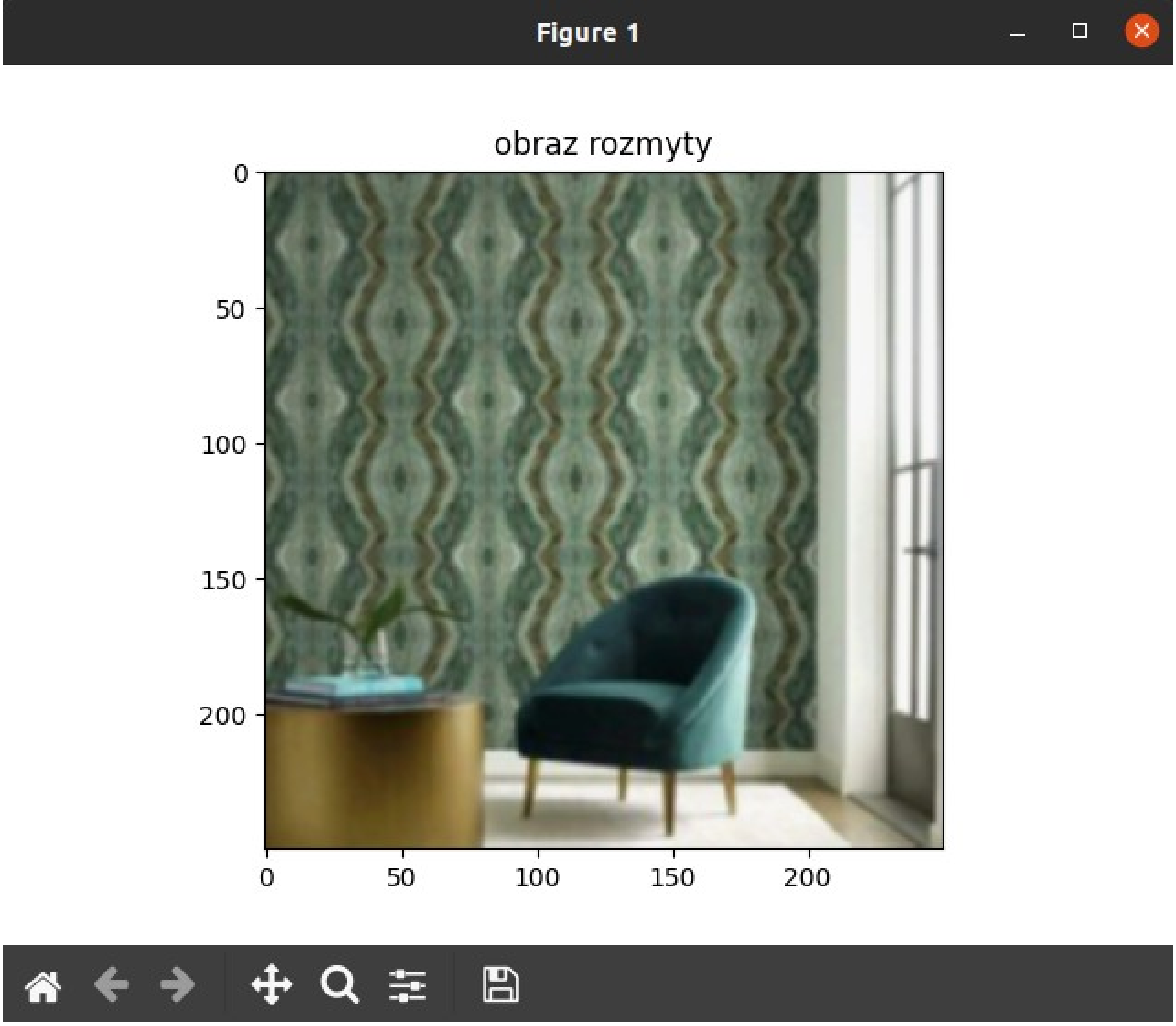


Figure 1
obraz rozmyty
0
50
100
150
200
0
50
100
150
200

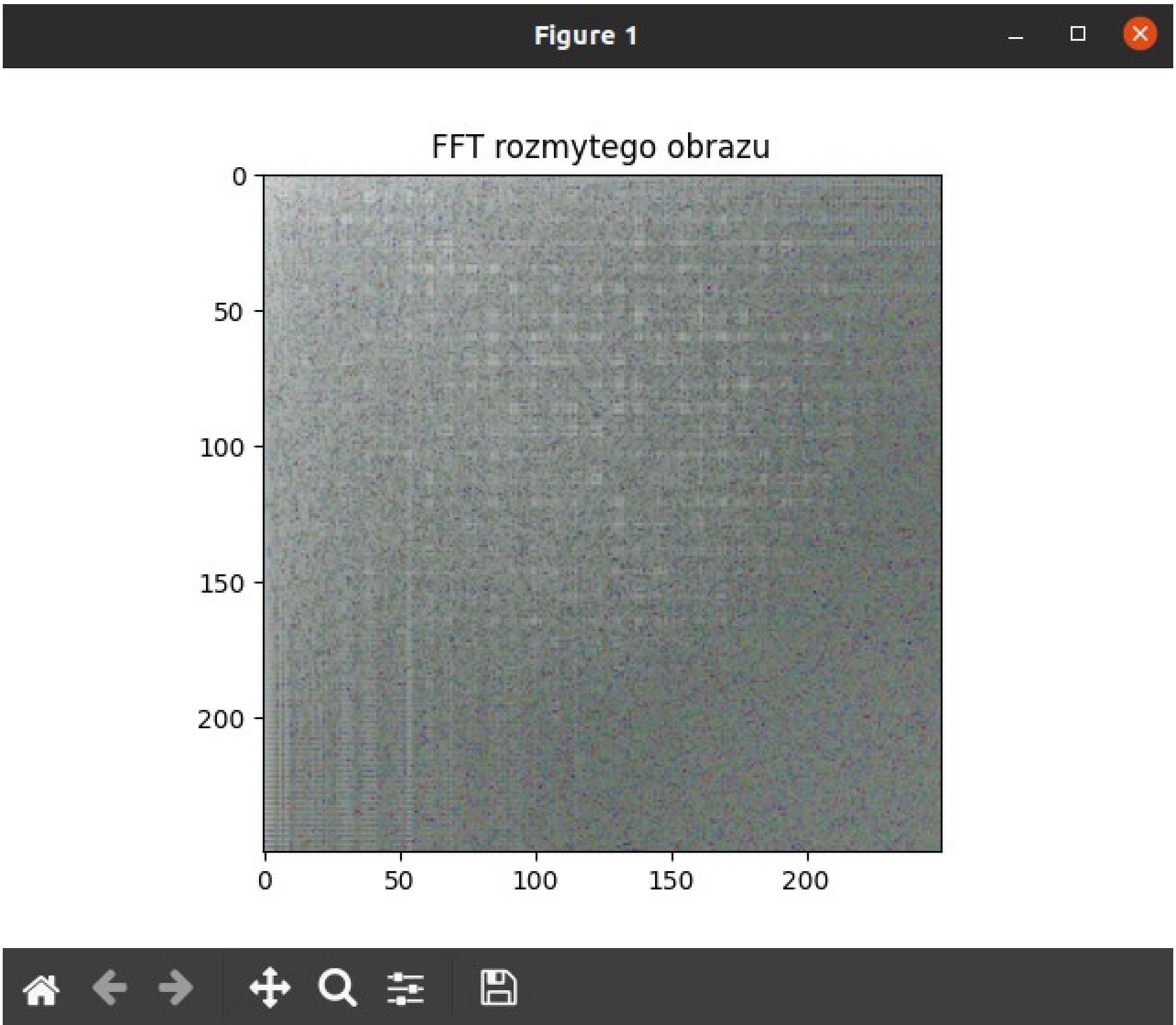


Figure 1
FFT rozmytego obrazu
0
50
100
150
200
0
50
100
150
200

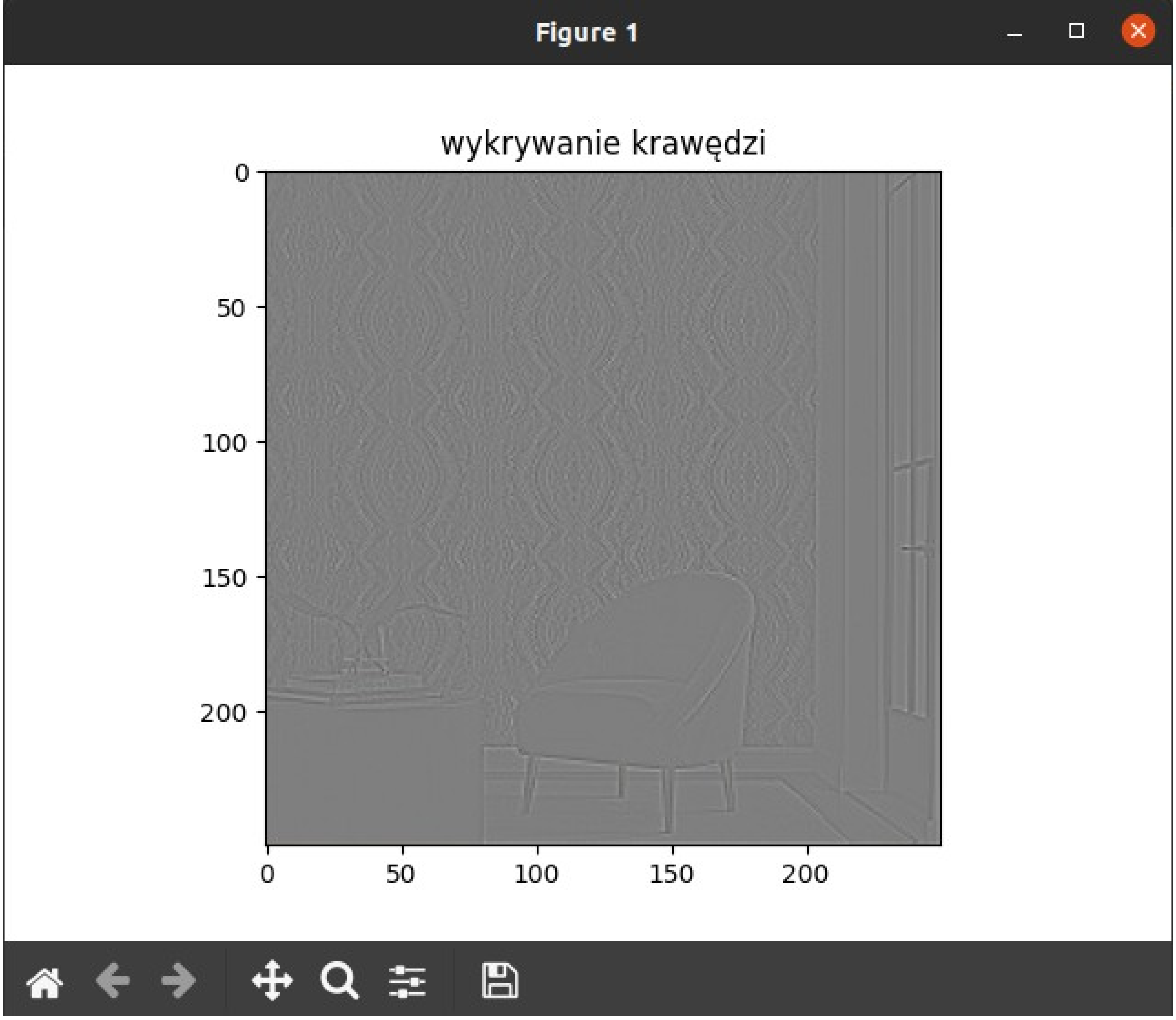


Figure 1
wykrywanie krawędzi
0
50
100
150
200
0
50
100
150
200

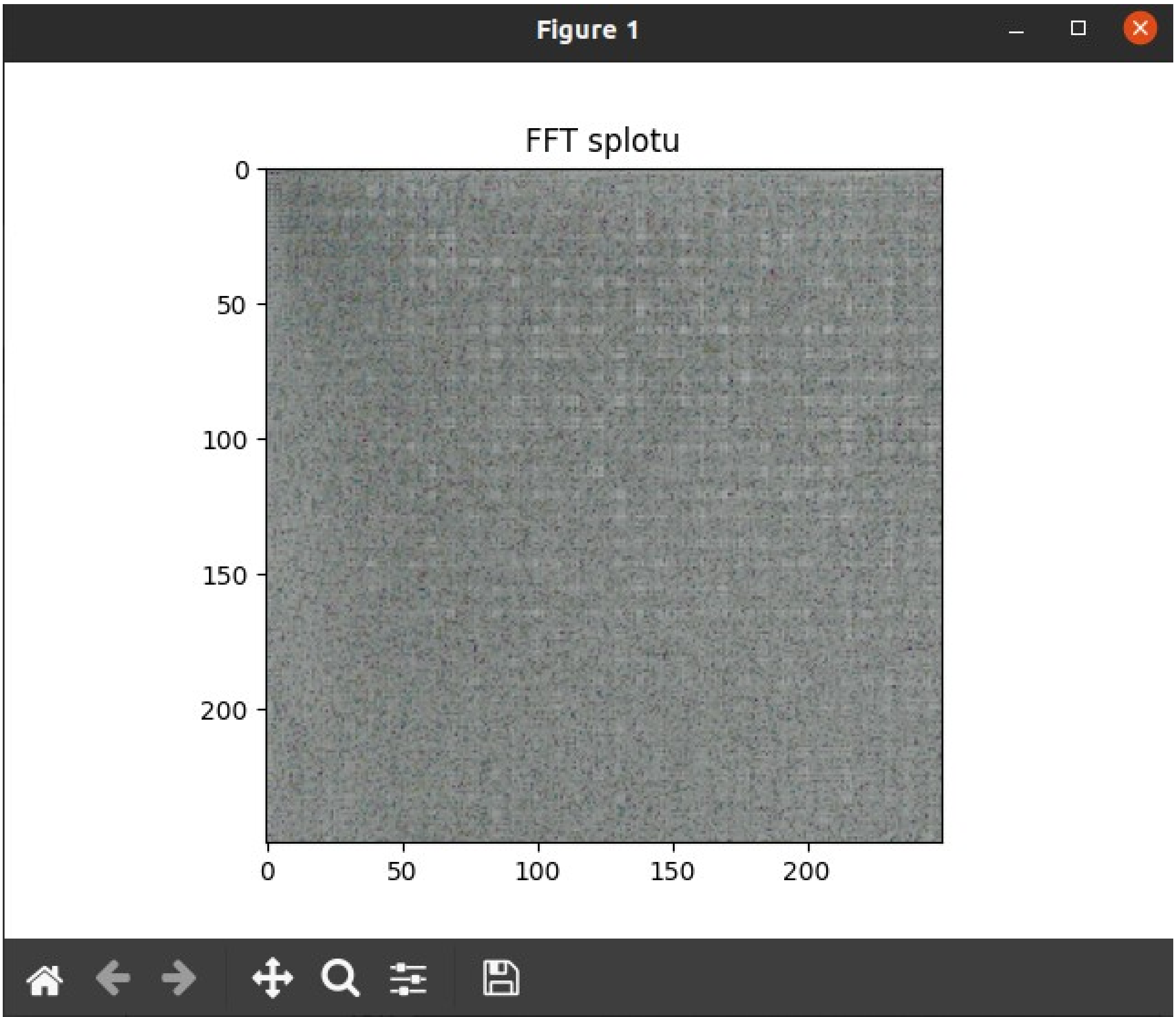


Figure 1
FFT splotu
0
50
100
150
200
0
50
100
150
200

Wartości brzegowe można policzyć rozszerzając obraz powielając brzegowe piksele lub rozszerzyć o ich lustrzane odbicie. Można ‘zawinąć’ obraz tj brać warości z przeciwnego końca obrazu, przyciąć obraz wynikowy lub przyciąć maskę przy przetwarzaniu krawędzi obrazu.

Przy rozmyciu obrazu większy udział w fft mają niskie częstotliwości, przy wyostrzeniu odwrotnie. Wykrywanie krawędzi wygasza widmo, pozostawiając pojedyńcze częstotliwości.

**Zad. 6**

Zaimplementować rozmycie Gaussa na okno o dowolnym rozmiarze. Przetestować okna o różnych rozmiarach.

• Jakie obserwujemy różnice w sile rozmycia na różnych oknach?
• Przetestować okna na białym obrazie z czarną prostą linią o grubości jednego piksela przechodzącą przez górny i dolny środek obrazu.
• Przetestować wykrywanie krawędzi przed i po rozmyciach. Co można zaobserwować w ilości wykrytych krawędzi?

```
import matplotlib.pyplot as plt
import matplotlib.image as img
import numpy as np
import scipy.fftpack as fp

def edge(image):
    x,y,z = image.shape
    ax = x-1
    ay = y - 1
    conv = np.zeros((x, y, z), dtype="float64")
    kernel = np.array([[0.0, -1.0, 0.0],
                       [-1.0, 4.0, -1.0],
                       [0.0, -1.0, 0.0]])/8

    for j in range(2):
        for c in range(2):
            conv[0][0] += kernel[j][c] * image[1 - j][1 - c]
            conv[0][ay] += kernel[j][c + 1] * image[1 - j][ay - c]
            conv[ax][0] += kernel[j + 1][c] * image[ax - j][1 - c]
            conv[ax][ay] += kernel[j + 1][c + 1] * image[ax - j][ay - c]

    for j in range(1, ay):
        for a in range(2):
            for b in range(3):
                conv[0][j] += kernel[a][b] * image[1 - a][j + 1 - b]
                conv[ax][j] += kernel[a + 1][b] * image[ax - 1 + a][j + 1 - b]
    for j in range(1, ax):
        for a in range(3):
            for b in range(2):
                conv[j][0] += kernel[a][b] * image[j + 1 - a][1 - b]
                conv[j][ay] += kernel[a][b + 1] * image[j + 1 - a][ay - 1 + b]

    for j in range(1, ax):
        for c in range(1, ay):
            for a in range(3):
                for b in range(3):
                    conv[j][c] += kernel[a][b] * image[j + 1 - a][c + 1 - b]

    return conv+255/2

def gaussian(x, mu, sigma):
  return np.exp( -(((x-mu)/(sigma))**2)/2.0 )

def Zad6(r):
    kernel_radius = r
    sigma = kernel_radius / 2.

    hkernel = [gaussian(x, kernel_radius, sigma) for x in range(2 * kernel_radius + 1)]
```

```
    vkernel = [x for x in hkernel]
    kernel2d = [[xh * xv for xh in hkernel] for xv in vkernel]

    kernelsum = sum([sum(row) for row in kernel2d])
    kernel2d = [[x / kernelsum for x in row] for row in kernel2d]

    image = img.imread('image2.jpg')
    x, y, z = np.shape(image)
    conv = np.zeros((x, y, z), dtype="float64")
    image = np.pad(image,[(kernel_radius,),(kernel_radius,),(0,)],mode='edge')

    ax = x - 1
    ay = y - 1

    for j in range(kernel_radius,x+kernel_radius):
        for c in range(kernel_radius, y+kernel_radius):
            for a in range(kernel_radius*2+1):
                for b in range(kernel_radius*2+1):
                    conv[j-kernel_radius][c-kernel_radius][:] += kernel2d[a][b] * image[j + kernel_radius - a][c +
kernel_radius - b][:]

    image = image[kernel_radius:ax+kernel_radius,kernel_radius:ay+kernel_radius,:]
    plt.imshow(image)
    plt.title("Obraz")
    plt.show()
    plt.imshow(edge(image).astype(int))
    plt.title("Wykrywanie krawędzi")
    plt.show()
    plt.imshow(conv.astype(int))
    plt.title("Rozmycie")
    plt.show()
    plt.imshow(edge(conv).astype(int))
    plt.title("Wykrywanie krawędzi po rozmyciu")
    plt.show()



    image = img.imread('linia.jpg')
    x, y, z = np.shape(image)
    conv = np.zeros((x, y, z), dtype="float64")
    image = np.pad(image, [(kernel_radius,), (kernel_radius,), (0,)], mode='edge')
    ax = x - 1
    ay = y - 1

    for j in range(kernel_radius, x + kernel_radius):
        for c in range(kernel_radius, y + kernel_radius):
            for a in range(kernel_radius * 2 + 1):
                for b in range(kernel_radius * 2 + 1):
                    conv[j - kernel_radius][c - kernel_radius][:] += kernel2d[a][b] * image[j + kernel_radius - a][
                                                                        c + kernel_radius - b][:]

    image = image[kernel_radius:ax + kernel_radius, kernel_radius:ay + kernel_radius, :]
    plt.imshow(image)
    plt.title("Linia")
    plt.show()
    plt.imshow(conv.astype(int))
    plt.title("Rozmyta linia")
    plt.show()
```

Zad6(2): okno 5x5

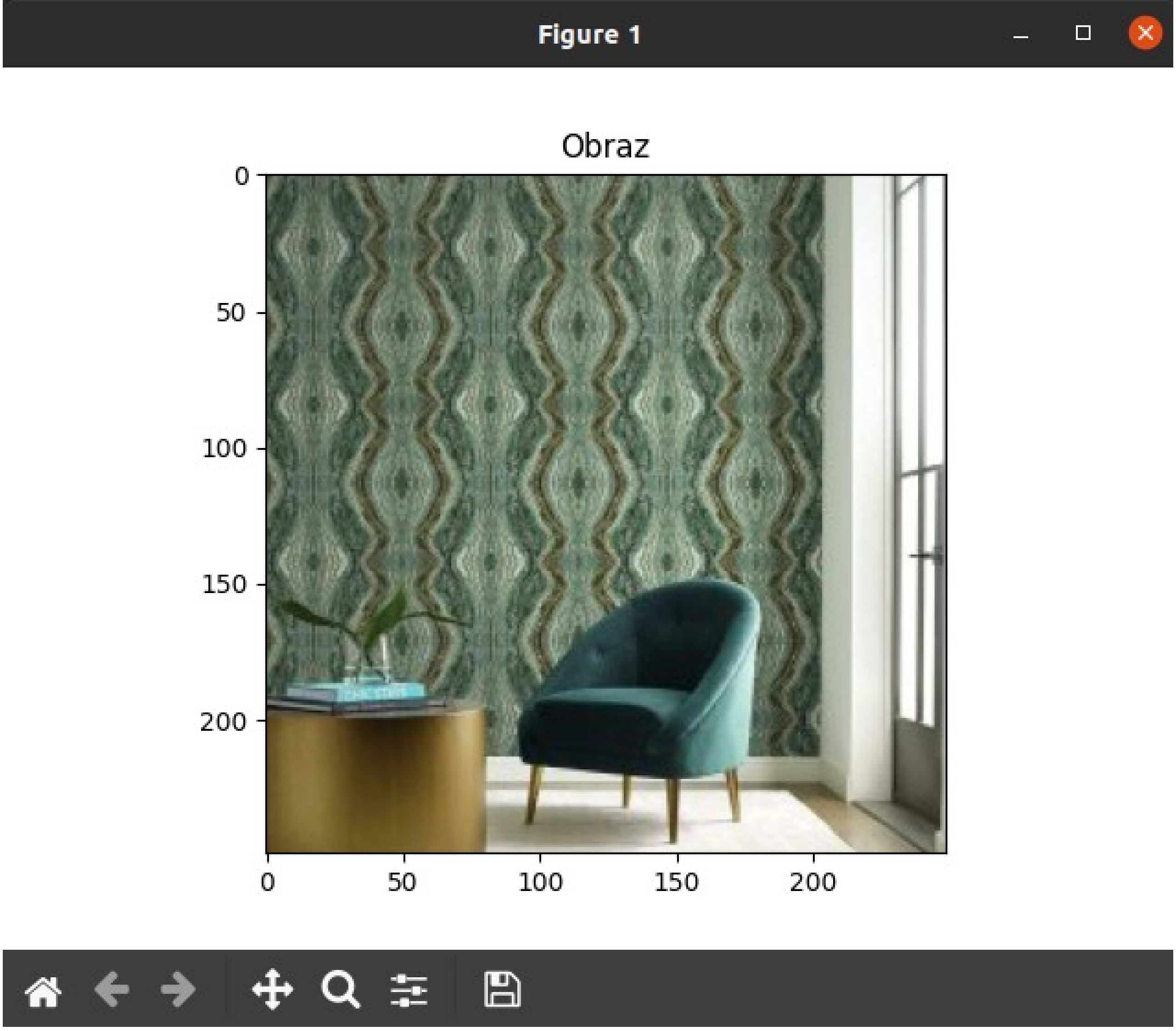


Figure 1
Obraz
0
50
100
150
200
0
50
100
150
200

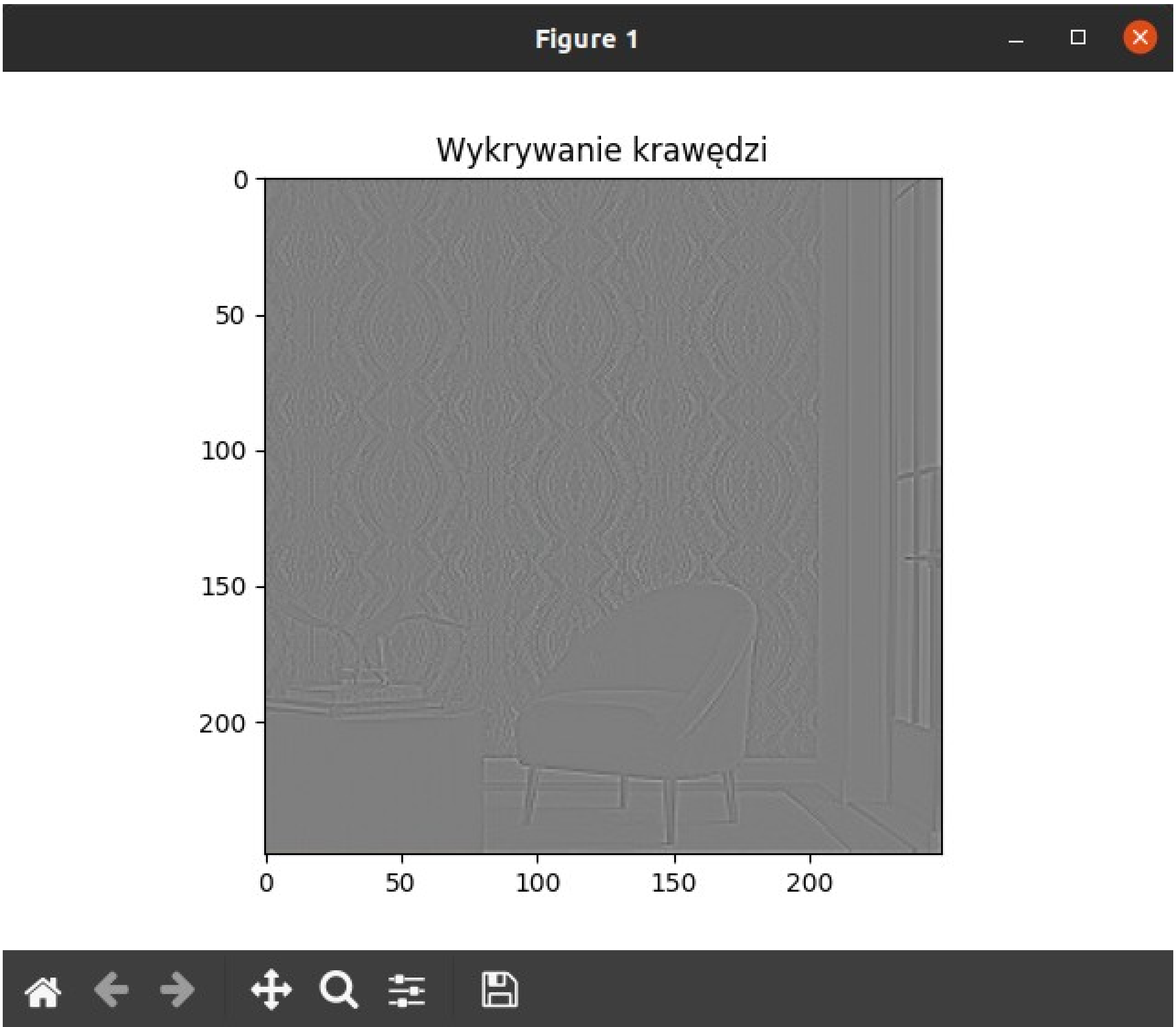


Figure 1
Wykrywanie krawędzi
0
50
100
150
200
0
50
100
150
200

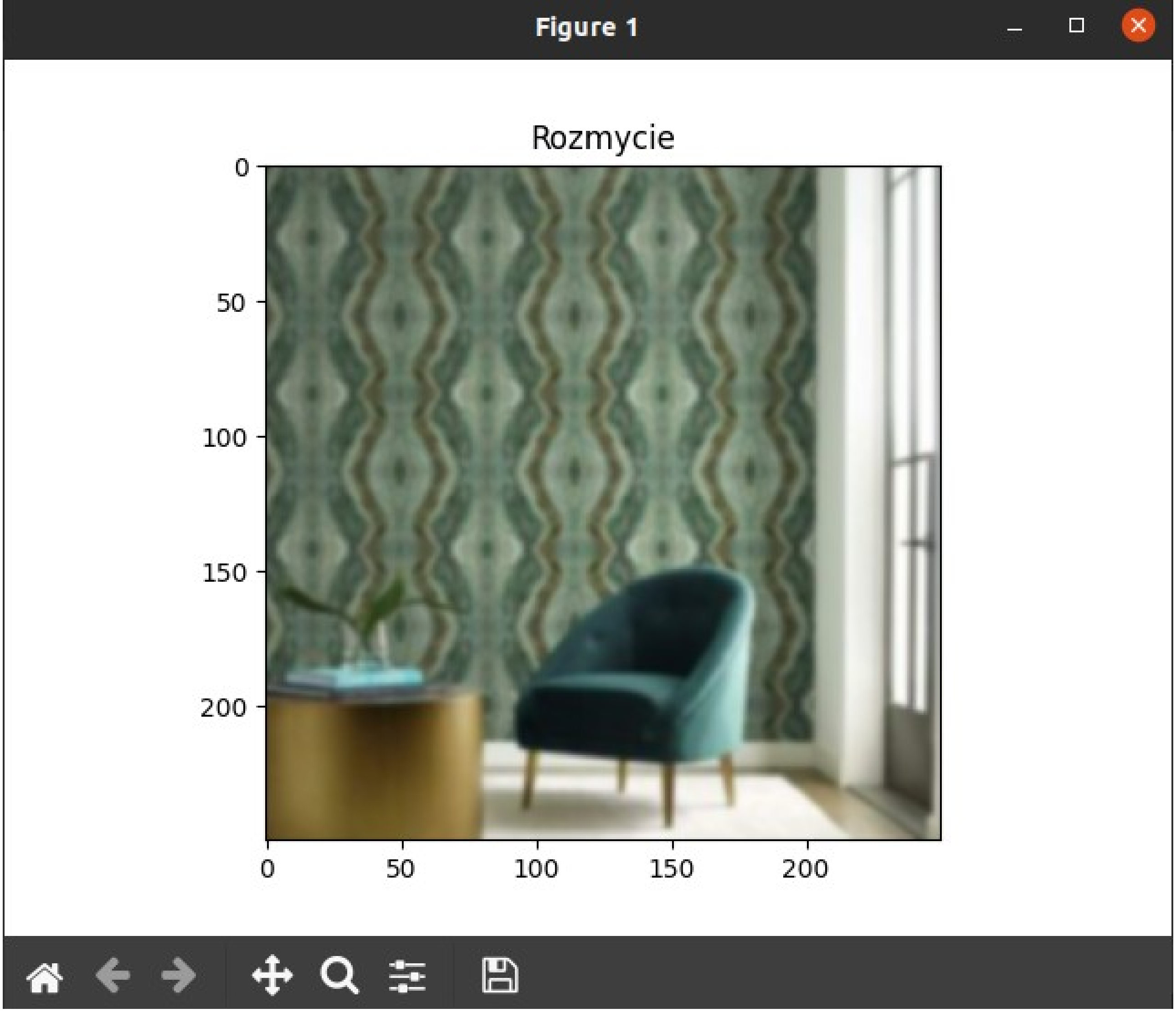


Figure 1
Rozmycie
0
50
100
150
200
0
50
100
150
200

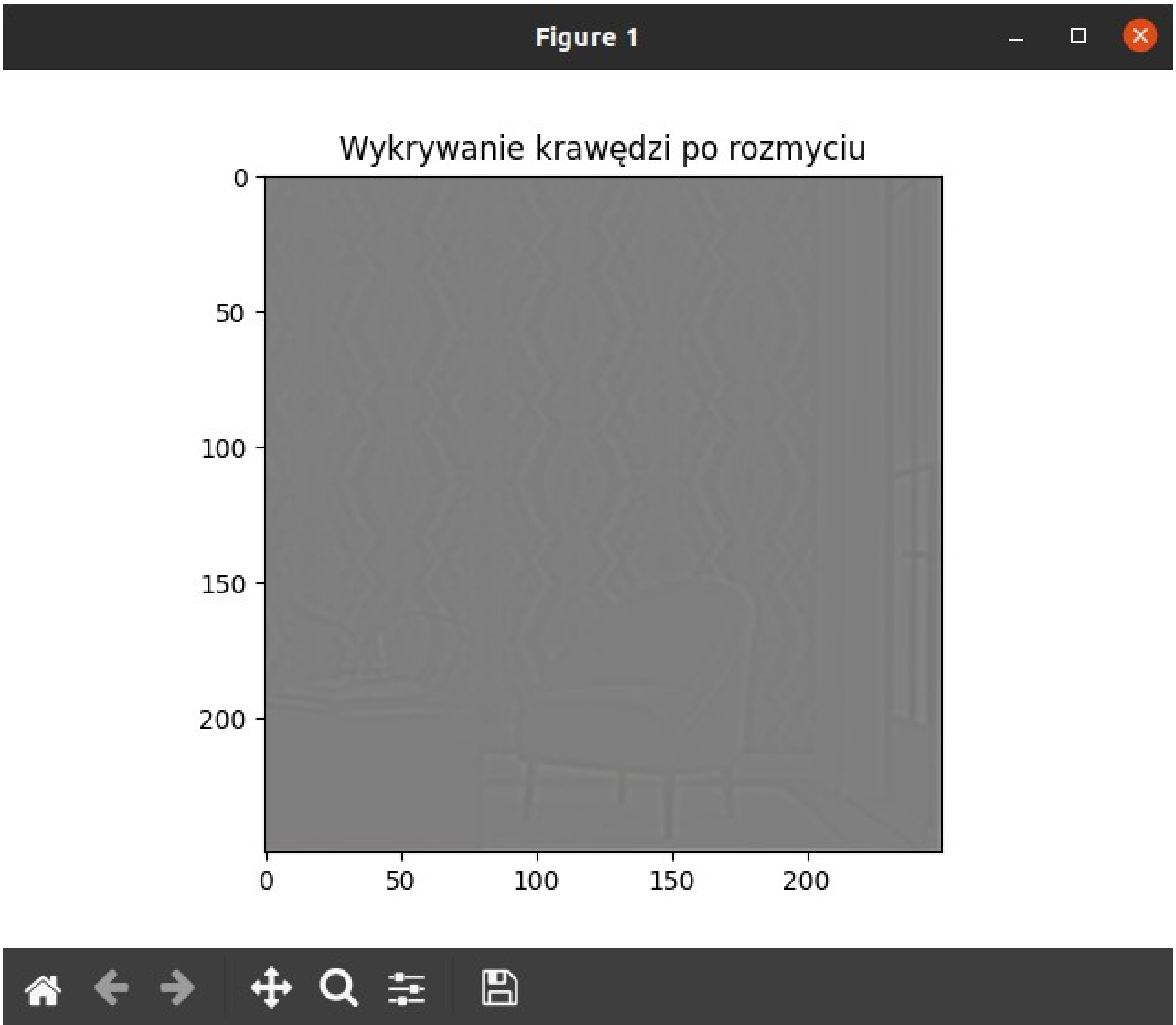


Figure 1
Wykrywanie krawędzi po rozmyciu
0
50
100
150
200
0
50
100
150
200

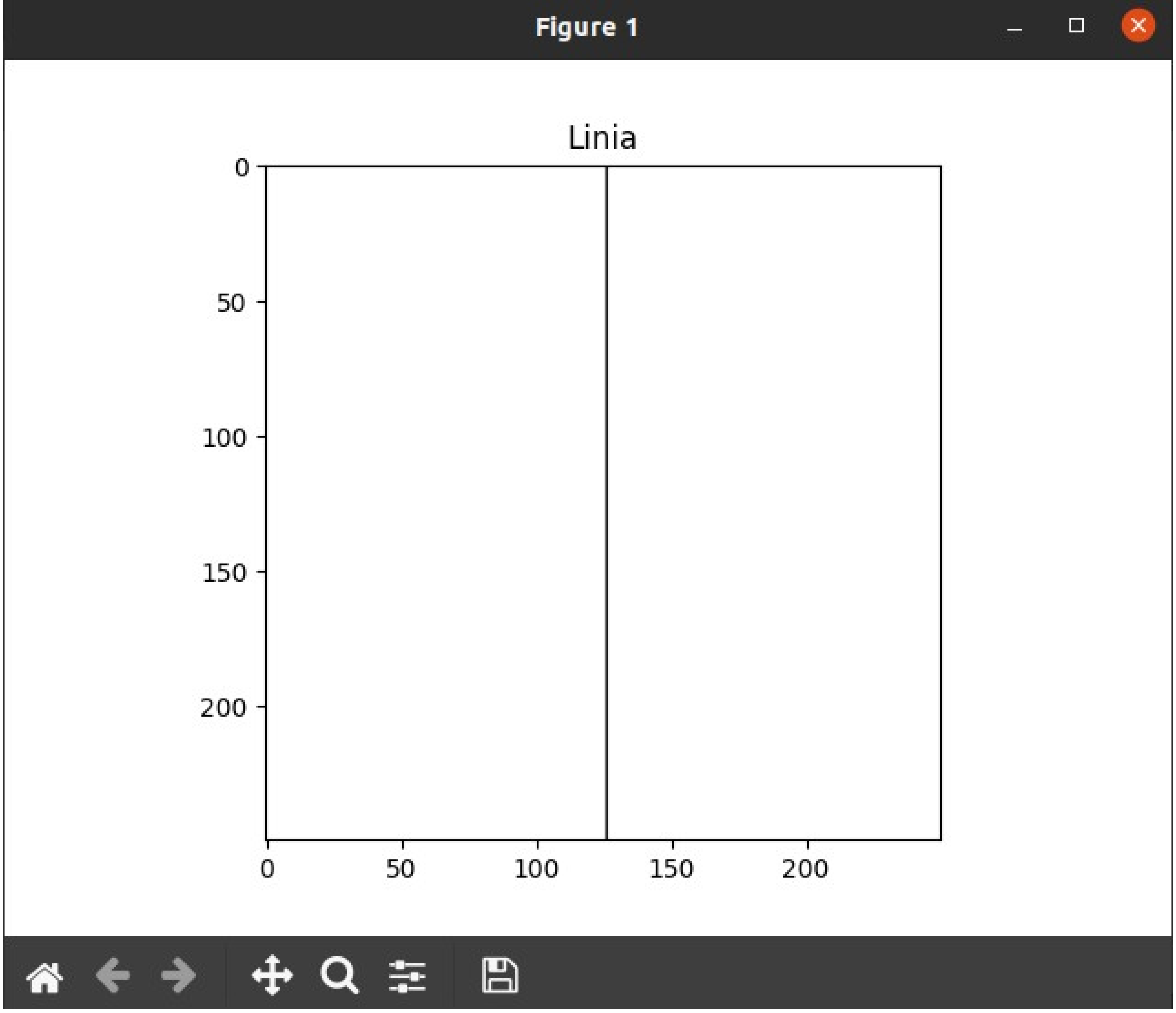


Figure 1
Linia
0
50
100
150
200
0
50
100
150
200

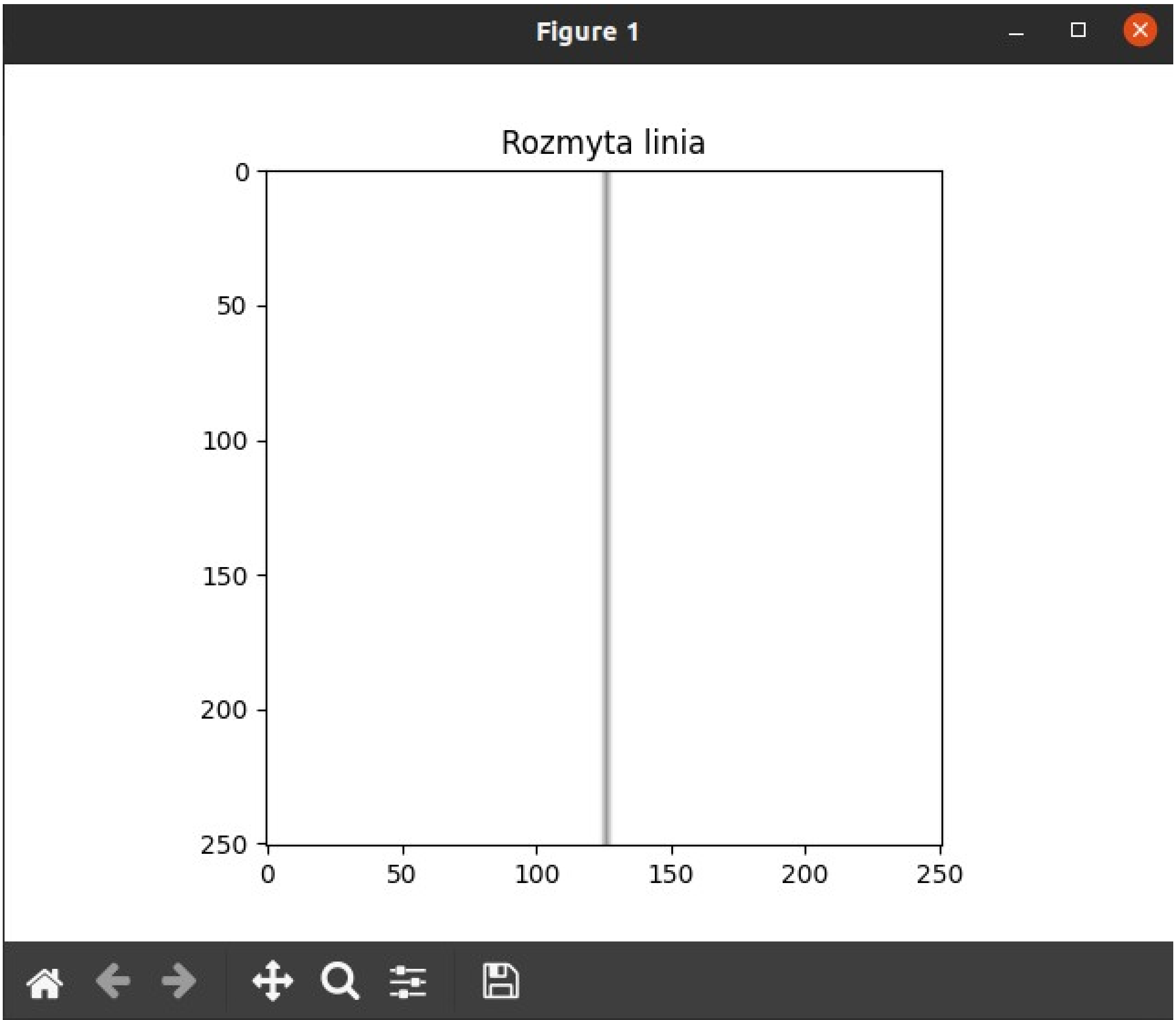
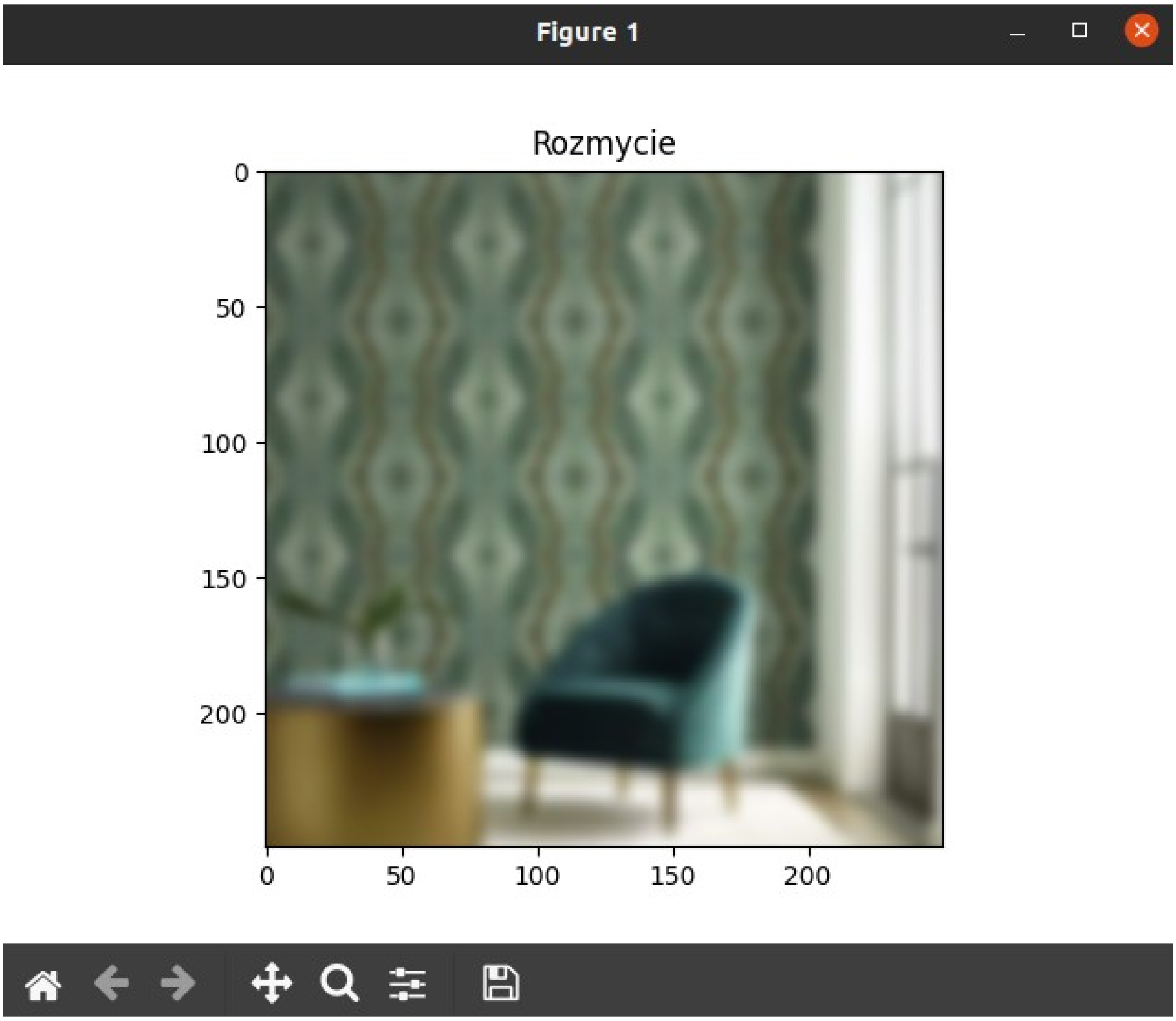
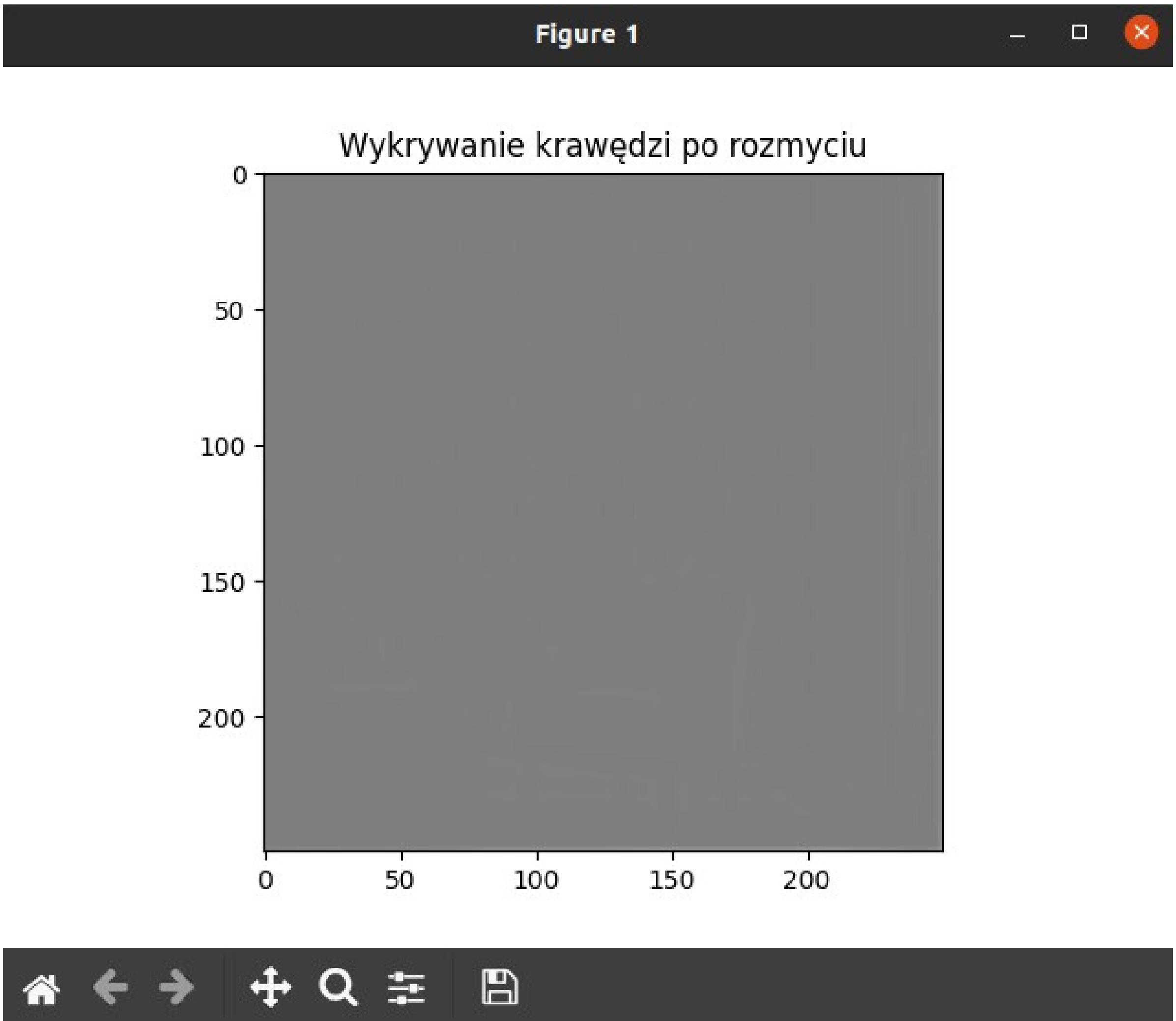
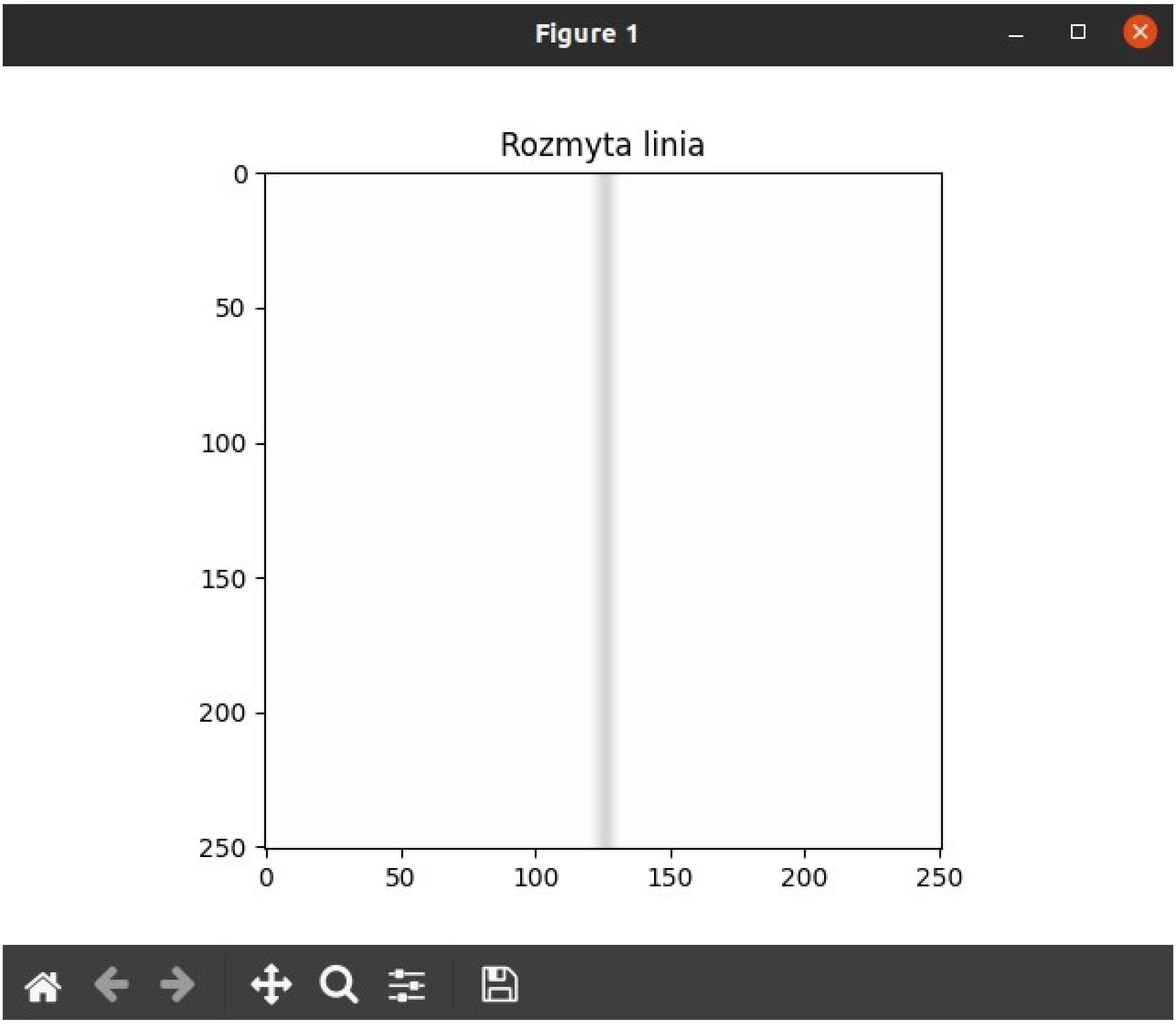



Figure 1
Rozmyta linia
0
50
100
150
200
250
0
50
100
150
200
250

Zad6(5): okno 11x11






Dla większych okien rozmycie jest silniejsze. Po rozmyciu obrazu wykrywanych jest mniej krawędzi.

**Podsumowanie i wnioski:**

System liniowy zawsze da odpowiedź na sumę sygnałów o tej samej wartości co suma odpowiedzi na te sygnały (Zad1). Operacja splotu jest przemienna i liniowa (Zad2). Wykonanie splotu kołowego jest równoważne wymnożeniu widm sygnałów splotu i wykonaniu odwrotnej transformaty Fouriera (Zad3). Operację splotu można wykorzystać do wygenerowania pogłosu przez splot dźwięku z odpowiedzią impulsową pomieszczenia (Zad4). Splot można też wykorzystać do edycji obrazów: poprzez sumowanie elementów splotu maski z macierzą pikseli o rozmiarze maski dla każdego piksela obrazu, można dokonać filtracji obrazu w celach np: wyostrzenia, rozmycia, wykrycia krawędzi (Zad5 i 6).