Sygnaly_Lab1_145204

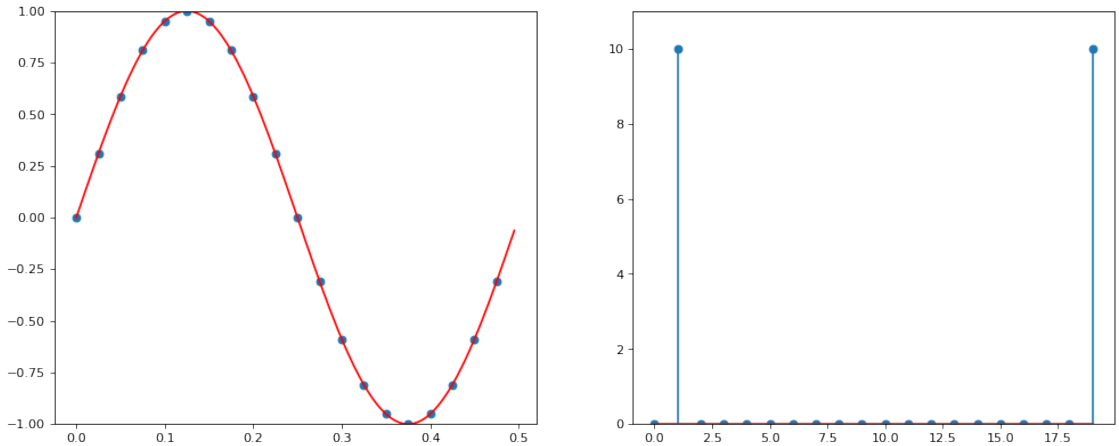
December 1, 2021

1 ZADANIE 1

1.0.1 Wykonaj w Pythonie poniższy skrypt i przeanalizuj go

[103]:
```
from pylab import *
from numpy import *
import math

from ipywidgets import *

#--- Definiujemy sygnal wejsciowy
A = 1         # Amplituda sygnalu
F = 2.0       # Czestotliwosc sygnalu [Hz]
T = 1/F       # Okres sygnalu [s]
f = lambda t : A * sin(2*pi*t*F)    # Def. analizowanej funkcji (sygnalu)

#--- Probkujemy sygnal
LP = 1        # Liczba analizowanych pełnych okresów sygnalu (okresow)
w = 40        # Częstotliwość probkowania [Hz]
TW = 1/w      # Okres probkowania [s] (co ile sekund pobieramy próbkę)

t = np.arange(0, LP*T, TW) # Momenty, w których pobieramy próbki (oś OX)
n = len(t)                 # Liczba próbek

signal = f(t)

#--- Rysujemy sygnał (niebieskie kółka)
fig = plt.figure(figsize=(15, 6), dpi=80)
ax = fig.add_subplot(121)
ax.plot(t, signal, 'o')

#--- Rysujemy sygnał przed spróbkowaniem (dla wizualizacji)
base_t = np.arange(0, LP * T, 1/200)
base_signal = f(base_t)
ax.plot(base_t, base_signal, linestyle='-', color='red')
ax.set_ylim([min(base_signal), max(base_signal)])

#--- Wykonujemy FFT
```

```
signal1 = fft.fft(signal)
signal1 = abs(signal1) # moduł

#--- Rysujemy FFT
ax = fig.add_subplot(122)
ymax = max(signal1)
ax.set_ylim([0.0, max(1.1*ymax, 3.0)])

freqs = range(n)
stem(freqs, signal1, '-*', use_line_collection=True);
```



UWAGA: do dalszych ćwiczeń warto powyższy skrypt przekształcić na funkcję o wielu argumentach, typu: amplituda, częstotliwość próbkowania, liczba przebiegów. Oczywiście dla wygody, należy nadać wartości domyślne argumentom funkcji.

[104]:
```
def wykresy_sygnalow(A = 1, F = 2.0, LP = 1, w = 40):
  #--- Definiujemy sygnal wejsciowy
  # A = 1         # Amplituda sygnalu
  # F = 2.0       # Czestotliwosc sygnalu [Hz]
  T = 1/F       # Okres sygnalu [s]
  f = lambda t : A * sin(2*pi*t*F)     # Def. analizowanej funkcji (sygnalu)

  #--- Probkujemy sygnal
  # LP = 1        # Liczba analizowanych pełnych okrebsów sygnalu (okresow)
  # w = 40        # Częstotliwość probkowania [Hz]
  TW = 1/w      # Okres probkowania [s] (co ile sekund pobieramy próbkę)

  t = np.arange(0, LP*T, TW) # Momenty, w których pobieramy próbki (oś OX)
  n = len(t)                 # Liczba próbek

  signal = f(t)
```

```python
#--- Rysujemy sygnał (niebieskie kółka)
fig = plt.figure(figsize=(15, 6), dpi=80)
ax = fig.add_subplot(121)
ax.plot(t, signal, 'o')

#--- Rysujemy sygnał przed spróbkowaniem (dla wizualizacji)
base_t = np.arange(0, LP * T, 1/200)
base_signal = f(base_t)
ax.plot(base_t, base_signal, linestyle='-', color='red')
ax.set_ylim([min(base_signal), max(base_signal)])
ax.set_xlabel("Time [s]")
ax.set_ylabel("Amplitude")


#--- Wykonujemy FFT
signal1 = fft.fft(signal)
signal1 = abs(signal1) # moduł

#--- Rysujemy FFT
ax = fig.add_subplot(122)
ymax = max(signal1)
ax.set_ylim([0.0, max(1.1*ymax, 3.0)])
ax.set_xlabel("Frequency [Hz]")
ax.set_ylabel("Amplitude")

freqs = np.linspace(0, w, len(signal1))
stem(freqs, signal1, '-*', use_line_collection=True);
```

UWAGA DLA CHĘTNYCH: można wykorzystać 'interact', dzięki któremu można zmieniac parametry danej funkcji i na bieżąco obserwować zmiany. Poniższy kod przedstawia sposób wykorzystania interact:

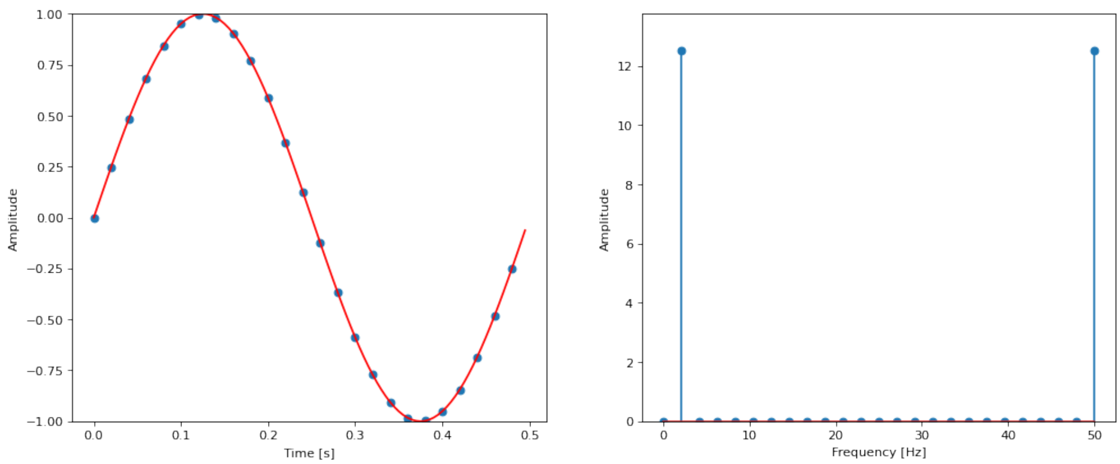
[105]:
```python
def prosta(a=2, b=0):
    x = linspace(-5, 5, 100, endpoint=False) # punkty na osi OX [s]
    f = lambda x : a*x + b
    y = f(x)

    fig = plt.figure(figsize=(6, 3), dpi=80)
    ax = fig.add_subplot(111)
    ax.set_xlim(-5, 5)
    ax.set_ylim(-5, 5)
    ax.plot(x, y)

interact(prosta, a=(-5,5,0.5), b=(-5,5,0.5))
```

```
interactive(children=(FloatSlider(value=2.0, description='a', max=5.0, min=-5.0,␣
↪step=0.5), FloatSlider(value=…
```

[105]:
```
<function __main__.prosta>
```

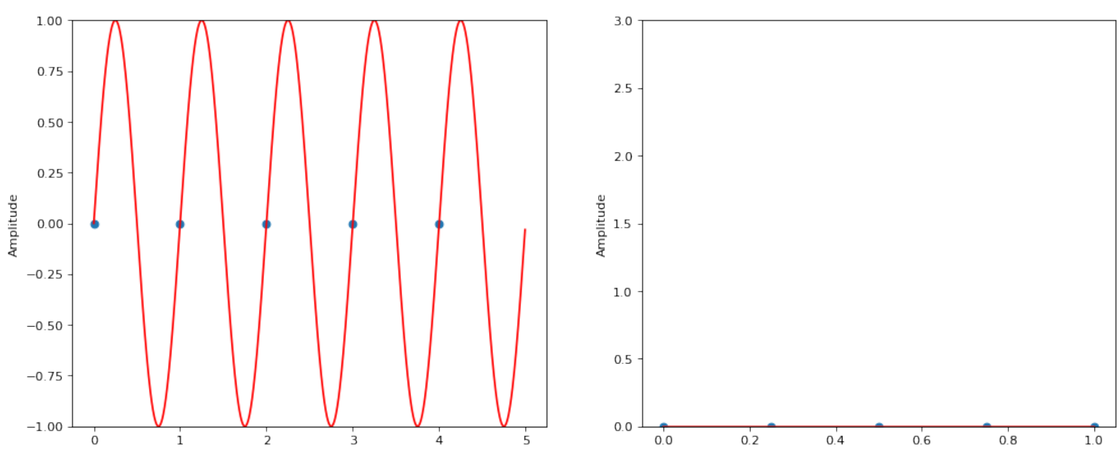
1.0.2 b) Zmień częstotliwość próbkowania na 50Hz.

[106]:
```
wykresy_sygnalow(w = 50)
```



1.0.3 c) Punkty na osi OX spektrum są teraz kolejnymi liczbami naturalnymi, a nie częstotliwościami w Hz. Popraw skrypt (funkcje), tak aby oś OX spektrum była w Hz (podpowiedź: oś OX rozpoczyna się od 0Hz, a kończy się na (prawie!) Hz, gdzie jest częstotliwością próbkowania). Następnie: Upewnij się, że spektrum dla 1Hz-owego sinusa i pięciu (LP=5) analizowanych przebiegów wygląda teraz prawidłowo.

[132]:
```
# popraw oś OX
wykresy_sygnalow(F = 1, LP = 5, w = 1)
```

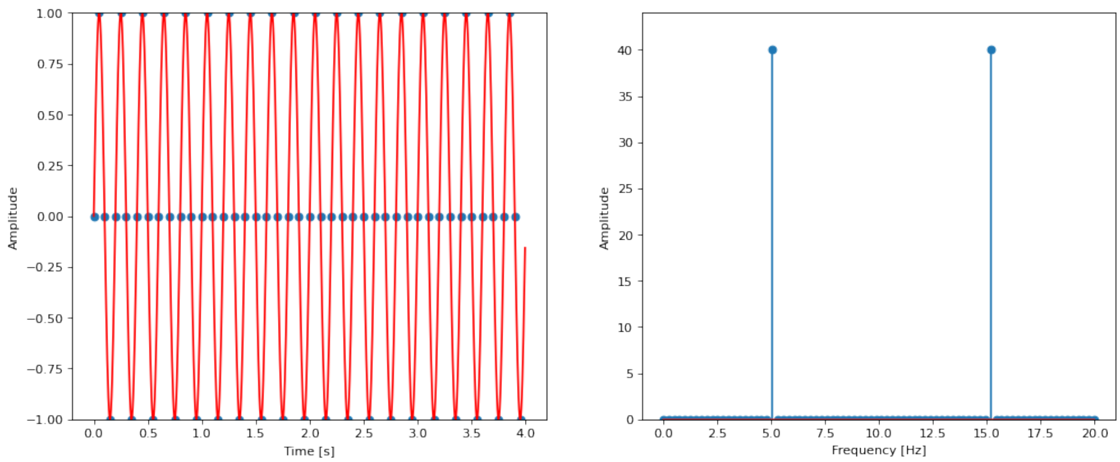


1.0.4 d) Podpisz osie obu wykresów, używając funkcji xlabel() i ylabel(). Pamiętaj o jednostkach.

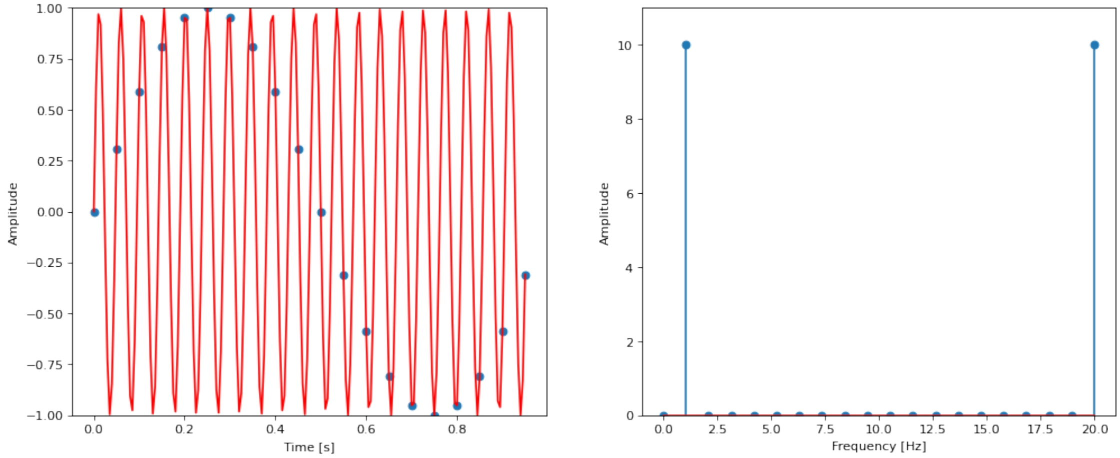
[108]:
```
# Podpisz osie wykresów
```

1.0.5 e) Wygeneruj spektrum dla funkcji sinus o częstotliwościach 5Hz i 21Hz, dla czestotliwości próbkowania 20Hz i 20 (LP=20) analizowanych przebiegów. Czy rozpoznajesz te funkcje patrząc na ich spróbkowane wykresy? Odczytaj w drugim przypadku uzyskaną częstotliwość z FFT. Dlaczego uzyskano taki wynik?

[109]:
```
# F=5, w=20, LP=20
wykresy_sygnalow(F=5, LP=20, w=20)
```

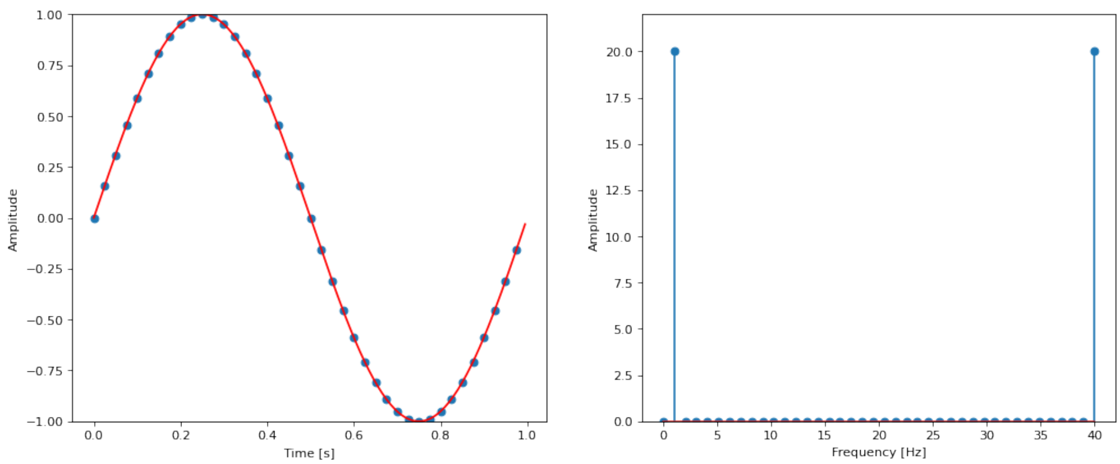


[110]:
```
# F=21, w=20, LP=20
wykresy_sygnalow(F=21, LP=20, w=20)
```

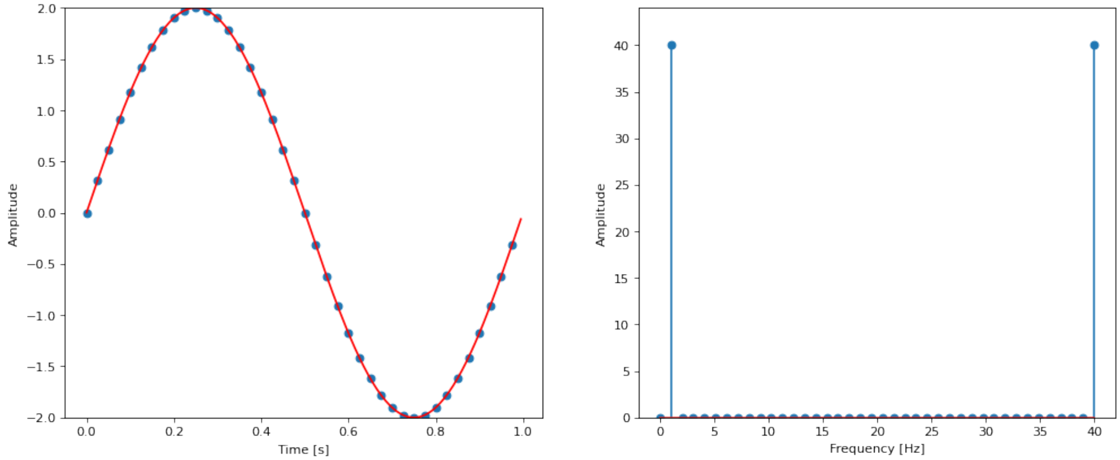


1.0.6 f) Porównaj spektrum funkcji $sin(F*2\pi t)$, $2sin(F*2\pi t)$ i $3sin(F*2\pi t)$. Jak zmienia się wartość na osi OY na wykresie spektrum?

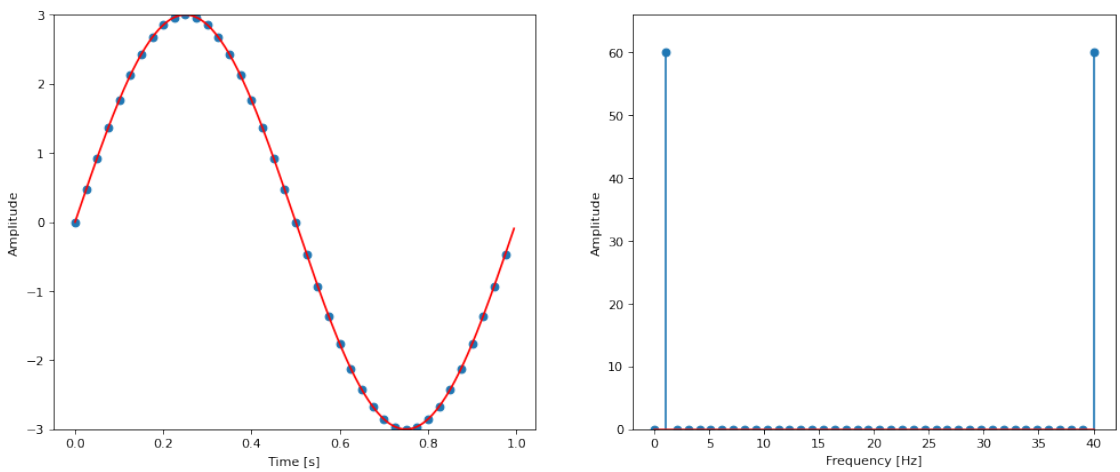
[111]:
```
# F=1, A=1, LP=1
wykresy_sygnalow(A=1, F=1, LP=1)
```



[112]:
```
# F=1, A=2, LP=1
wykresy_sygnalow(A=2, F=1, LP=1)
```

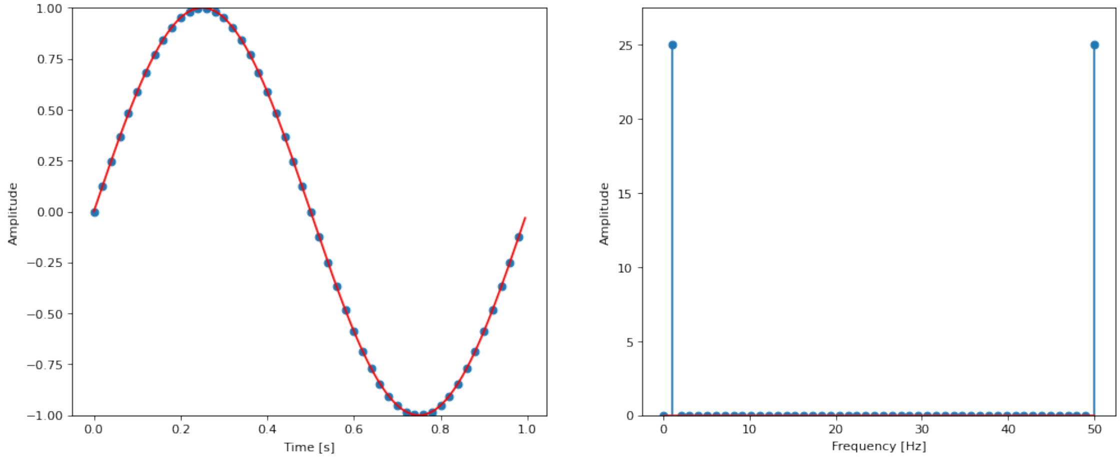


[113]:
```python
# F=1, A=3, LP=1
wykresy_sygnalow(A=3, F=1, LP=1)
```

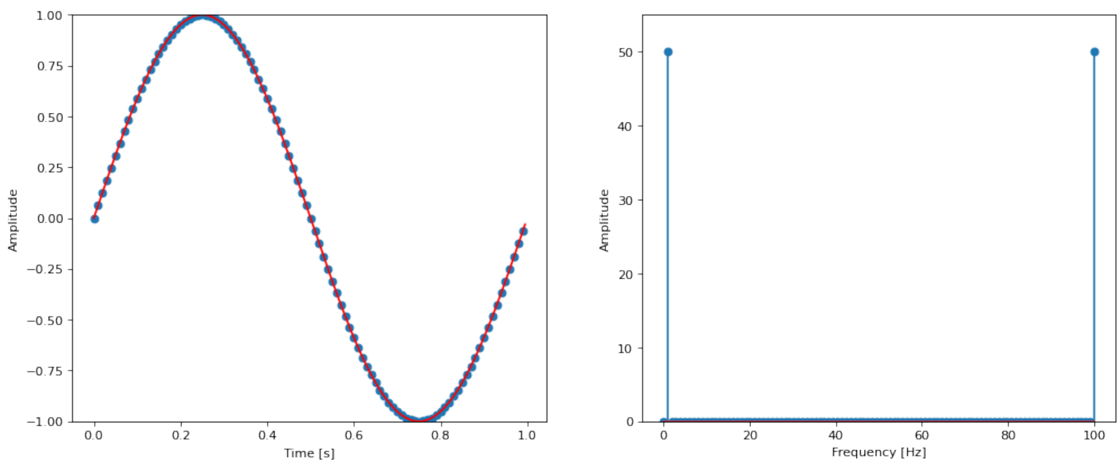


1.0.7 g) Ile punktów jest na wykresach przy częstotliwości próbkowania 50Hz, $T = 1s$? Zwiększ dwukrotnie liczbę próbek poprzez zwiększenie częstotliwości próbkowania. Następnie: dla $sin(F * 2\pi t)$ porównaj wartość na osi OY spektrum uzyskane w tym oraz poprzednim punkcie.

[114]:
```python
# F=1, w=50, LP=1
wykresy_sygnalow(F=1, LP=1, w=50)
```



[115]:
```
# F=1, w=100, LP=1
wykresy_sygnalow(F=1, LP=1, w=100)
```



1.0.8 h) Na podstawie wyników uzyskanych w dwóch poprzednich punktach przeskaluj oś OY spektrum tak, aby wskazywała wartości amplitud badanych sygnałów. Sprawdź wyniki dla kilku wybranych funkcji (tu fajnie użyć interact), częstotliwości próbkowania oraz rozważanych liczb punktów. Pamiętaj o wysokim LP.

[116]:
```
# popraw oś OY
interact(wykresy_sygnalow, A=(1,3), F=(1,21), w = (0, 100) , LP = 20)
```

```
interactive(children=(IntSlider(value=1, description='A', max=3, min=1),␣
↪IntSlider(value=2, description='F', m…
```

[116]:
```
<function __main__.wykresy_sygnalow>
```

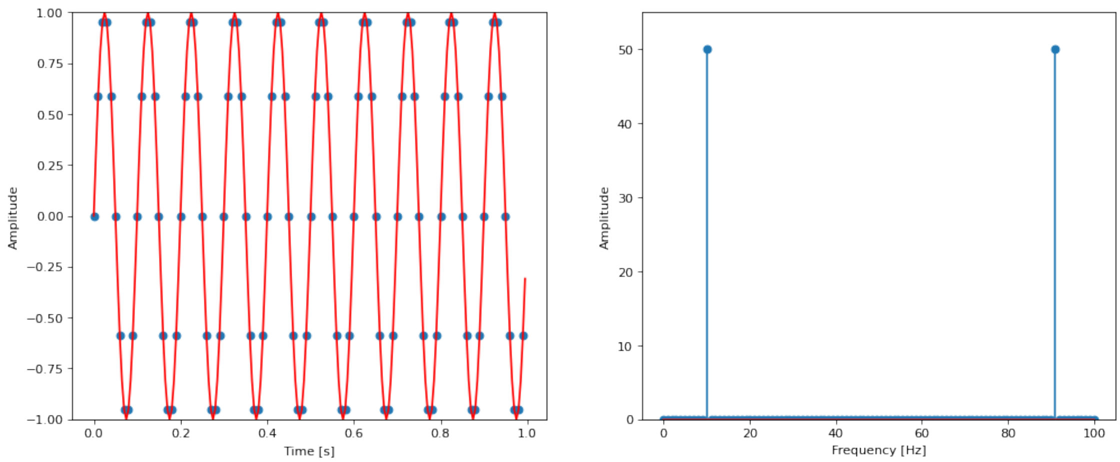
2 Zadanie 2

Zwróć uwagę, że spektrum jest symetryczne (poza pierwszym elementem).

a) Przy w=100Hz, przeanalizuj widma sygnałów:

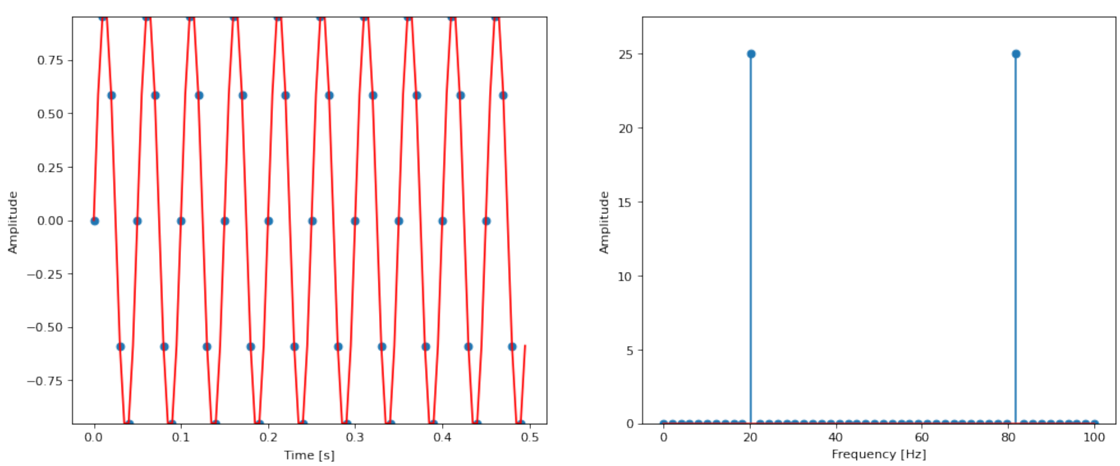
i) $sin(F * 2\pi t)$, $F = 10Hz$

[133]:
```
# f=10, w=100, A=1, LP=10
wykresy_sygnalow(A=1, F=10, w=100, LP=10)
```



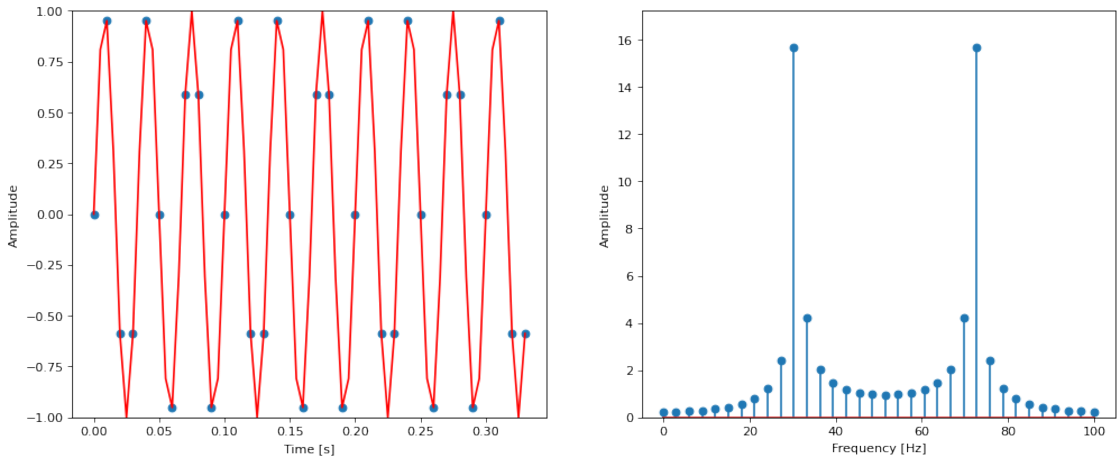
ii) $sin(F * 2\pi t)$, $F = 20Hz$

[118]:
```
# f=20, w=100, A=1, LP=10
wykresy_sygnalow(A=1, F=20, w=100, LP=10)
```



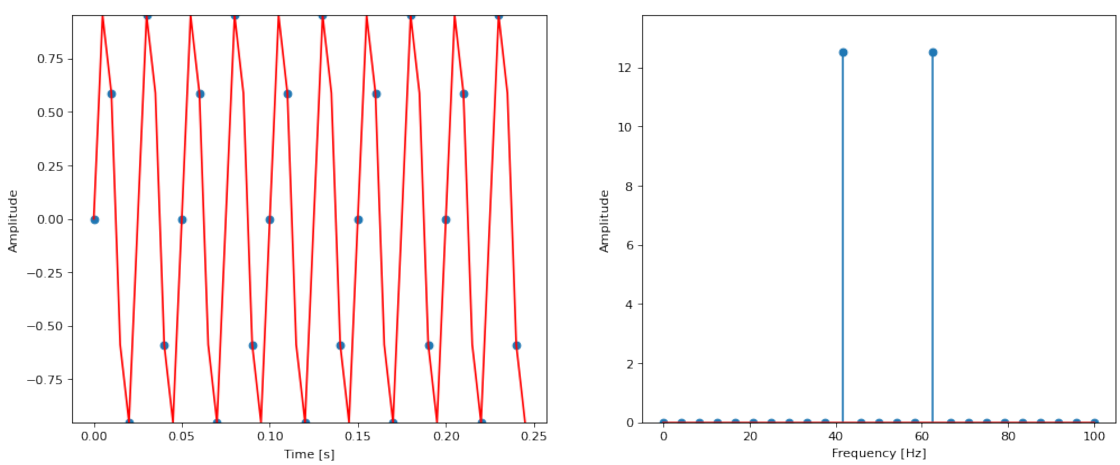
iii) $sin(2 * pi * t * f),\ f = 30Hz$

```
[119]: # f=30, w=100, A=1, LP=10
wykresy_sygnalow(A=1, F=30, w=100, LP=10)
```



iv) $sin(2 * pi * t * f),\ f = 40Hz$

```
[120]: # f=40, w=100, A=1, LP=10
wykresy_sygnalow(A=1, F=40, w=100, LP=10)
```

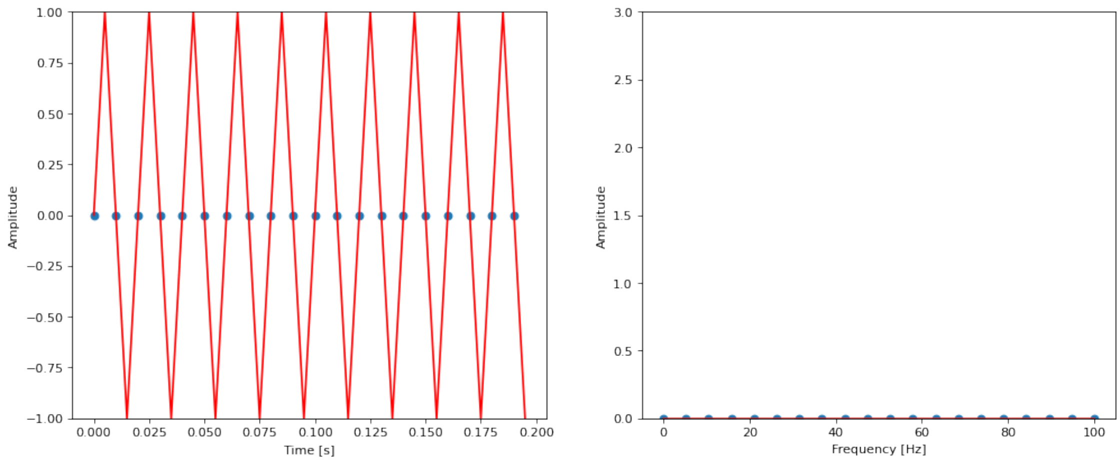


2.0.1 b) Jak się nazywa twierdzenie, którego konsekwencją jest takie zachowanie się spektrum?

Twierdzenie Nyquista-Shannona (o próbkowaniu)

2.0.2 c) Przy F=50Hz, w=100Hz, wygeneruj spektrum dla $sin(2 * pi * f * t)$. Zwróć uwagę na skalę wykresów.

[121]:
```
# f=50, w=100, A=1, LP=10
wykresy_sygnalow(A=1, F=50, w=100, LP=10)
```



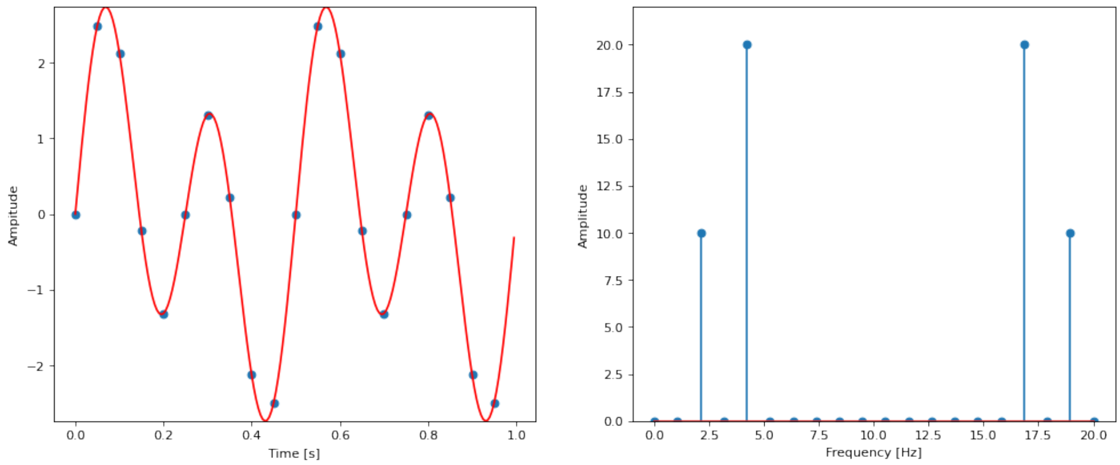
3 Zadanie 3

Poniższe przypadki skłądają się z złożonych sygnałów. Jeżeli stworzyłeś/aś na początku funkcję odpowiedzialną za rysowanie sygnału i FFT, możesz ją zmodyfikować tak, by jako jej argument podawać funkcję lambda, realizującą sygnał.

a) $sin(2 * pi * t * f) + 2 * sin(4 * pi * t * f)$, T=1s, w=20Hz.

[122]:
```
def wykresy_sygnalow_funkcja(f, F = 2.0, LP = 1, w = 40):
  #--- Definiujemy sygnal wejsciowy
  # A = 1         # Amplituda sygnalu
  # F = 2.0       # Czestotliwosc sygnalu [Hz]
  T = 1/F       # Okres sygnalu [s]

  #--- Probkujemy sygnal
  # LP = 1        # Liczba analizowanych pełnych okrebsów sygnalu (okresow)
  # w = 40        # Częstotliwość probkowania [Hz]
  TW = 1/w      # Okres probkowania [s] (co ile sekund pobieramy próbkę)

  t = np.arange(0, LP*T, TW) # Momenty, w których pobieramy próbki (oś OX)
  n = len(t)                 # Liczba próbek

  signal = f(t)

  #--- Rysujemy sygnał (niebieskie kółka)
```

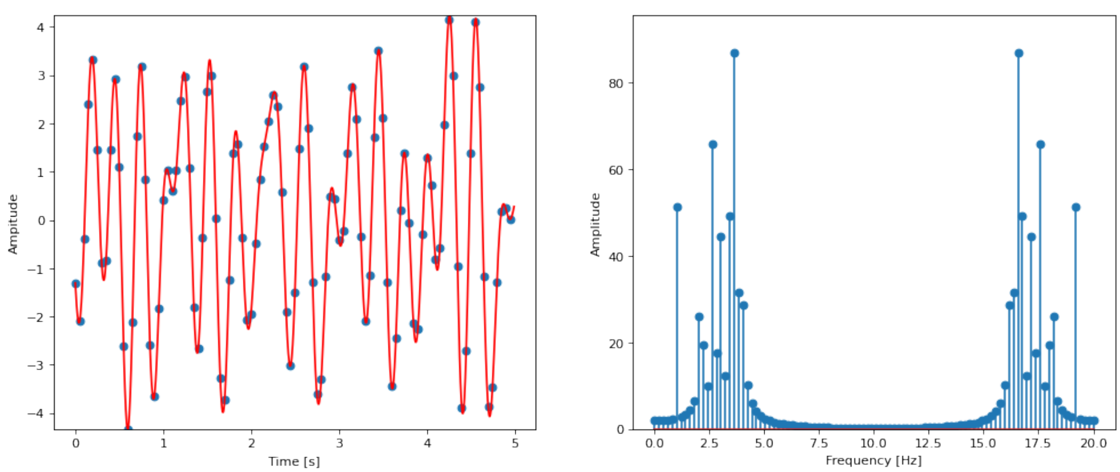
```
    fig = plt.figure(figsize=(15, 6), dpi=80)
    ax = fig.add_subplot(121)
    ax.plot(t, signal, 'o')

    #--- Rysujemy sygnał przed spróbkowaniem (dla wizualizacji)
    base_t = np.arange(0, LP * T, 1/200)
    base_signal = f(base_t)
    ax.plot(base_t, base_signal, linestyle='-', color='red')
    ax.set_ylim([min(base_signal), max(base_signal)])
    ax.set_xlabel("Time [s]")
    ax.set_ylabel("Ampitude")


    #--- Wykonujemy FFT
    signal1 = fft.fft(signal)
    signal1 = abs(signal1) # moduł

    #--- Rysujemy FFT
    ax = fig.add_subplot(122)
    ymax = max(signal1)
    ax.set_ylim([0.0, max(1.1*ymax, 3.0)])
    ax.set_xlabel("Frequency [Hz]")
    ax.set_ylabel("Amplitude")

    freqs = np.linspace(0, w, len(signal1))
    stem(freqs, signal1, '-*', use_line_collection=True);



wykresy_sygnalow_funkcja(lambda t: (sin(2*pi*t*F)+2*sin(4*pi*t*F)), F = 1, LP =␣
 ↪1, w = 20)
```



3.0.1 c) Wygenerujemy trochę szumu. Wychodząc z funkcji sin(2*2*pi*t*f*), dodamy do niej 100 losowych sinusów. Wygenerujemy dla nich losowo amplitudy (z [0.1, 0.3], częstotliwosci: [2.0, 4.0] oraz modyfikacje fazy[0 + math.pi]. Jeżli utworzyłeś/aś funkcje lambda wcześniej, śmiało możesz ją podmienić na 'zwyczajną funkcję', która będzie realizować sumowanie losowych sinusów. Inne parametry: w=20Hz, T=1s. Czy dla powyższych paramerów losowania, jesteś w stanie odnaleźć bazowy przebieg (sin(2*2*pi*t))? (Jeżeli wykorzystujesz podany na początku kod, to nie przejmuj się, że pełen sygnał i sprókowany sobie nie odpowiadają. Każdy jest inny - bo losowy).

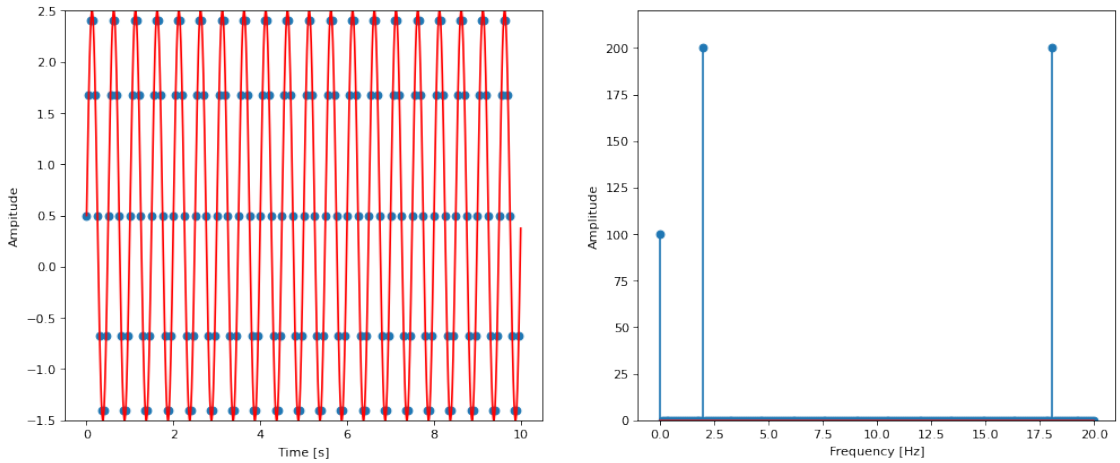
[123]:
```
# Dodaj szum do sygnału
from random import uniform

def random_sin():
    A = uniform(0.1, 0.3)
    f = uniform(2.0, 4.0)
    p = uniform(0, 2*pi)
    return lambda t: A*sin(2*pi*t*f + p)

noise = [random_sin() for _ in range(100)]
func = lambda t: sin(2*pi*t*1) + sum(g(t) for g in noise)
wykresy_sygnalow_funkcja(func, F = 1, LP = 5, w = 20)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11:
DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future
will give a different result. Use np.sum(np.fromiter(generator)) or the python
sum builtin instead.
  # This is added back by InteractiveShellApp.init_path()
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11:
DeprecationWarning: Calling np.sum(generator) is deprecated, and in the future
will give a different result. Use np.sum(np.fromiter(generator)) or the python
sum builtin instead.
  # This is added back by InteractiveShellApp.init_path()
```



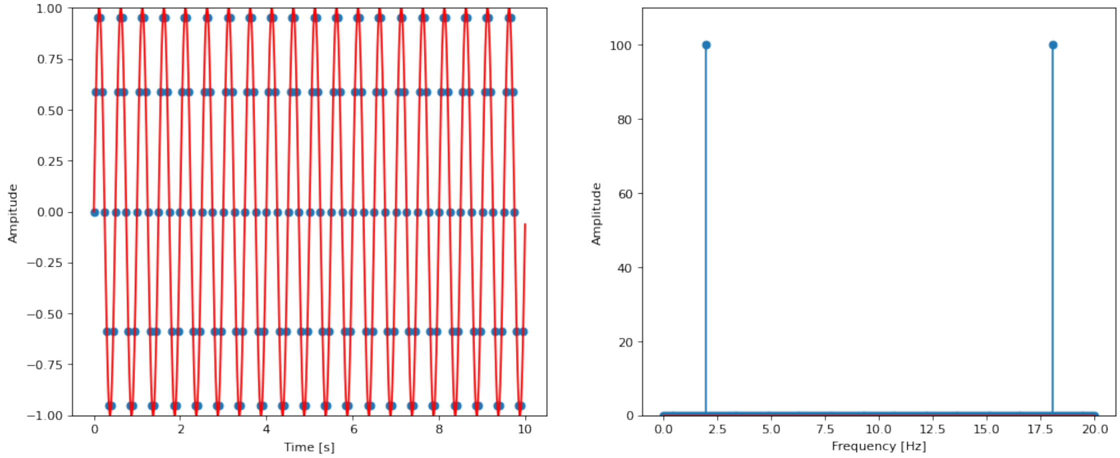
3.0.2 d) $0.5 + 2 * sin(2 * pi * t * f)$, T=1s, w=20Hz, LP=10. Czy amplituda zerowego prążka jest prawidłowa? Dlaczego? (Podpowiedź: zwróć jeszcze raz uwagę na "symetrię" spektrum).

[124]:
```
# Wyrysuj sygnał

wykresy_sygnalow_funkcja(lambda t : (0.5+2*sin(2*pi*t*F)), F = 1, LP = 10, w = 20)
```

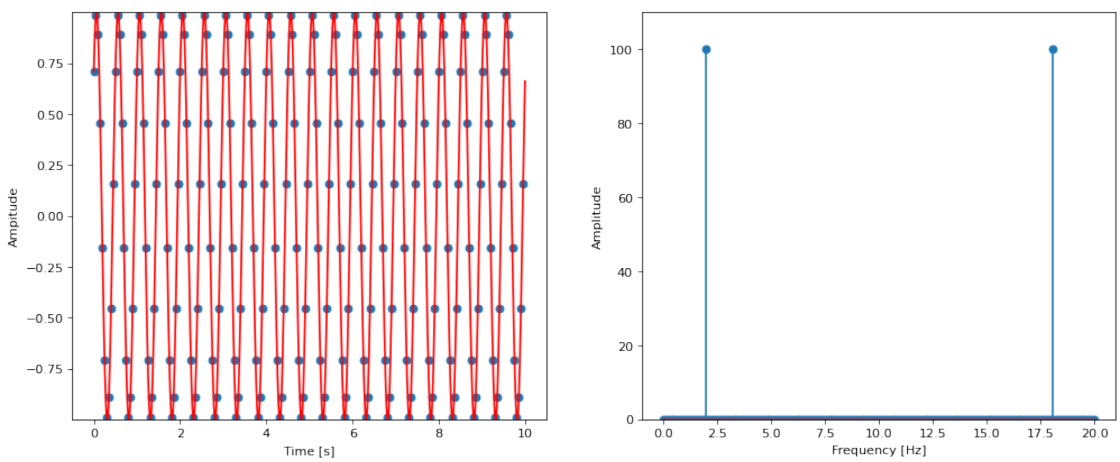


3.0.3 e) $sin(2 * pi * t * f)$ oraz $sin(2 * pi * t * f + pi/4)$ dla T=1s, w=20Hz. Czy informacja o fazie zniknęła? Poszukaj śladów tej informacji w tablicy, będącej wynikiem operacji fft(signal).

[125]:
```
#  faza = 0

wykresy_sygnalow_funkcja(lambda t : sin(2*pi*t*F), F = 1, LP = 10, w = 20)
```



[126]:
```
# faza = pi/4

wykresy_sygnalow_funkcja(lambda t : sin(2*pi*t*F + pi/4), F = 1, LP = 10, w =
 ↪20)
```



4 Zadanie 4

4.0.1 Oblicz wynik ifft(fft(x)), gdzie x=random.random(10). Czy jakaś informacja została stracona? (Uwaga: natkniesz się na problemy numeryczne – rozwiąż je).

[134]:
```
# wygeneruj wektor x
x = random.random(10)
print(x)
```

```
[0.8052232  0.52164715 0.90864888 0.31923609 0.09045935 0.30070006
 0.11398436 0.82868133 0.04689632 0.62628715]
```

[135]:
```
# oblicz fft z x
fft_x = fft.fft(x)
print(fft_x)
```

```
[ 4.56176388+0.j          1.20837678-0.25973067j -0.17790596-0.72882491j
  0.36860091+0.32897322j  0.66183215+1.35176479j -0.63133966+0.j
  0.66183215-1.35176479j  0.36860091-0.32897322j -0.17790596+0.72882491j
  1.20837678+0.25973067j]
```

[136]:
```
# oblicz ifft z x
ifft_x = ifft(fft_x)
print(ifft_x)
```

```
[0.8052232 +0.00000000e+00j 0.52164715-1.11022302e-17j
 0.90864888+6.52572721e-18j 0.31923609+7.35046114e-18j
 0.09045935-1.05588484e-17j 0.30070006-7.91065716e-19j
 0.11398436+1.05588484e-17j 0.82868133-6.07048993e-18j
 0.04689632-6.52572721e-18j 0.62628715+1.06133247e-17j]
```

[137]:
```
np.isclose(ifft_x.real, x)
```

[137]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True])
```