Лабораторная работа №1

Ряды Фурье

Предмет: Частотные методы

Автор: Made by Polyakov Anton, the part of R3236, suir family

Преподаватель: Алексей Алексеевич Перегудин

Исходный блокнот, код

Онлайн версия

Навигация

Вступление, приветствие

Сегодня поговорим о рядах Фурье в комплексном и вещественном случае

Вспомогательные функции и библиотеки, их описание

P.S.1 - Очень часто в коде фигурирует метод *spi.quad* из библиотека scipy, который по сути является взятием определённого интеграла в заданных границах

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy.integrate as spi

from svgpathtools import svg2paths, Path
from xml.dom import minidom
from svg.path import parse_path
from svg.path.path import Line
from xml.dom import minidom

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
# изначально не включен в список стандартных библиотек colab'a, если
# вы онлайн версии, то сначала зпустите эту ячейку
# Раскоментить, если в онлайн версии
# !pip install svgpathtools
```

In [ ]:
```
def plot(x, y):
  plt.plot(x, [y(t) for t in x], label='y_exact', color='blue')
  plt.title(f"Original function")
  plt.xlabel('t')
  plt.ylabel('y(t)')
  plt.grid()
  plt.legend(loc='lower right')
  plt.figure(figsize=(6,4))
  plt.show()
```

Функции для вычисления частичных сумм Фурье, случай $\mathbb{R} \to \mathbb{R}$

In [ ]:
```
# result --> (a0, 0), (a1, b1)...(aN, bN)
def compute_real_fourier_coeffs(func, N, h, T):
    result = []
    # start EQUALS n
    for n in range(N+1):
        an = (2/T) * spi.quad(lambda t: func(t) * np.cos((2 * np.pi * n * t) / T), h, h+T)[
        bn = (2/T) * spi.quad(lambda t: func(t) * np.sin((2 * np.pi * n * t) / T), h, h+T)[
        result.append((np.real(an), np.real(bn)))
    return np.array(result)
```

In [ ]:
```
def fit_func_by_fourier_series_with_real_coeffs(t, AB, T):
    result = 0.
    A = AB[:,0]
    B = AB[:,1]
    result +=  A[0]/2.
    for n in range(1, len(AB)):
            result +=  float(A[n]) * np.cos(2. * np.pi * n * t / T) + float(B[n]) * np.sin(

    return result
```

In [ ]:
```
def F_N(func, x, N, h, T):
    AB = compute_real_fourier_coeffs(func, N, h, T)
    return fit_func_by_fourier_series_with_real_coeffs(x, AB, T)
```

In [ ]:
```
def plot_real_fourier(func, x, N, h, T):
    for i in N:
        y_approx = F_N(func, x, i, h, T)
        plt.figure(figsize = (6, 4))
        plt.plot(x, [func(t) for t in x], label='exact', color='blue')
        plt.plot(x, y_approx, color='red', label='approx',linestyle='dashed')
        plt.title(f"Real Fourier, N={i}")
        plt.xlabel('t')
        plt.ylabel('y(t)')
        plt.legend(loc='lower right')
        plt.grid()
        plt.show()
```

Функции для вычисления частичных сумм Фурь, случай $\mathbb{C} \to \mathbb{R}$

In [ ]:
```
def complex_quad(func, a, b):

    def real_func(x):
        return np.real(func(x))
    def imag_func(x):
        return np.imag(func(x))
    real_integral = spi.quad(real_func, a, b)
    imag_integral = spi.quad(imag_func, a, b)
    return real_integral[0] + 1j*imag_integral[0]
```

In [ ]:
```
#function that computes the complex fourier coefficients c-N,.., c0, .., cN
def compute_complex_fourier_coeffs(func, N, h, T):
    result = []
```

```
    for n in range(-N, N+1):
        cn = (1/T) * complex_quad(lambda t: func(t) * np.exp(-1j * 2 * np.pi * n * t / T),
        result.append(cn)
    return np.array(result)
```

In [ ]:
```
def fit_func_by_fourier_series_with_complex_coeffs(t, C, T):
    result = 0. + 0.j
    L = int((len(C) - 1) / 2)
    for n in range(-L, L+1):
        c = C[n+L]
        result +=  c * np.exp(1j * 2. * np.pi * n * t / T)
    return result
```

In [ ]:
```
def G_N(func, x, N, h,T):
    C = compute_complex_fourier_coeffs(func, N, h, T)
    return fit_func_by_fourier_series_with_complex_coeffs(x, C, T)
```

In [ ]:
```
def plot_complex_fourier(func, x, N, h, T):
    for i in N:
        y_approx = G_N(func, x, i, h, T)
        plt.figure(figsize = (6, 4))
        plt.plot(x, [func(t) for t in x], label='exact', color='blue')
        plt.plot(x, y_approx, color='red', label='approx', linestyle='dashed')
        plt.title(f"Complex Fourier, N={i}")
        plt.xlabel('t')
        plt.ylabel('y(t)')
        plt.legend(loc='lower right')
        plt.grid()
        plt.show()
```

Код для красивого вывода коэффициентов фурье (любых матричных структур)

In [ ]:
```
def printBeauty(matrix, Type=complex):
    A = pd.DataFrame(np.round(matrix,4).astype(Type))
    A.columns = ['']*A.shape[1]
    print(A.to_string(index=False))
```

Код для вывода оригинального графика второго задания

In [ ]:
```
def plot_task2_original(func,x):
  plt.figure(figsize=(6, 4))
  plt.plot(func(x).real, func(x).imag, color='blue', label='exact')
  plt.xlabel('Re(f(t))')
  plt.ylabel('Im(f(t))')
  plt.title('Task2 - original plot')
  plt.legend(loc='lower right')
  plt.grid()
  plt.show()
```

Код для подсчёта коэффициентов Фурье в случае $\mathbb{C} \to \mathbb{C}$, вывода их в удобном виде

In [ ]:
```
def compute_complex_plane_fourier_coeffs(func, N, h,T):
    result = []
    for n in range(-N, N+1):
        cn = (1./T) * complex_quad(lambda t: func(t) * np.exp(-1j * 2 * np.pi * n * t / T),
        result.append(cn)
    return np.array(result)

def fit_func_by_fourier_series_with_complex_plane_coeffs(t, C, T):
    result = 0. + 0.j
    L = int((len(C) - 1) / 2)
    for n in range(-L, L+1):
        c = C[n+L]
        result +=  c * np.exp(1j * 2. * np.pi * n * t / T)
```

```
    return result

def G_N_complex(f,x,N, h,T):
    C = compute_complex_plane_fourier_coeffs(f, N, h,T)
    y_approx = fit_func_by_fourier_series_with_complex_plane_coeffs(x, C, T)

    return [[y.real for y in y_approx], [y.imag for y in y_approx]]
```

In [ ]:
```
def plot_complex_plane_fourier(func, x, N, h, T):
    for i in N:
        G_N = G_N_complex(func,x,i, h,T)
        plt.figure(figsize = (6, 4))
        plt.plot(func(x).real, func(x).imag, label='exact',color='blue')
        plt.plot(G_N[0], G_N[1], color='red', linestyle='dashed',label='approx')
        plt.title('Complex Fourier, R-->C ; ' + f"N={i}")
        plt.xlabel('t')
        plt.ylabel('y(t)')
        plt.legend(loc='lower right')
        plt.grid()
        plt.show()
```

In [ ]:
```
def plot_part(y,x,title,N):
  plt.plot(x, y, label='approx',color='blue')
  plt.title(f"Complex Fourier --> N={N} ; " + title)
  plt.xlabel('t')
  plt.ylabel('y(t)')
  plt.grid()
```

In [ ]:
```
def plot_complex_plane_fourier_part(func, x, N, h,T):
  for n in N:
    G_N = G_N_complex(func, x, n, h,T)
    plt.figure(f"Complex Fourier Parts", figsize=(10,10))
    plt.subplot(2, 2, 1)
    plot_part(G_N[0],x,'Re_G_N(t)',n)
    plt.subplot(2, 2, 2)
    plot_part(func(x).real,x,'Re_f(t)', n)
    plt.subplot(2, 2, 3)
    plot_part(G_N[1],x,'Im_G_N(t)',n)
    plt.subplot(2, 2, 4)
    plot_part(func(x).imag,x,'Im_f(t)',n)


    plt.show()
```

Содержательная часть

Задание 1. Вещественные функции

Придумаем числа a, b, t_0, t_1, t_2 при условии $a, b > 0$, $t_2 > t_1 > t_0 > 0$

In [ ]:
```
a = 1; b = 5; t0 = 1; t1 = 2; t2=4
```

Теперь зададим следующие функции и изучим их детально...

Step-функция

$$T = t_2 - t_0$$

$$f(t) = \begin{cases} a, & t \in [t_0, t_1) \\ b, & t \in [t_1, t_2) \end{cases}$$

In [ ]:
```
T1 = t2-t0
step_func = np.vectorize(lambda t: a if t0 <= t% T1 < t1 else b)
```

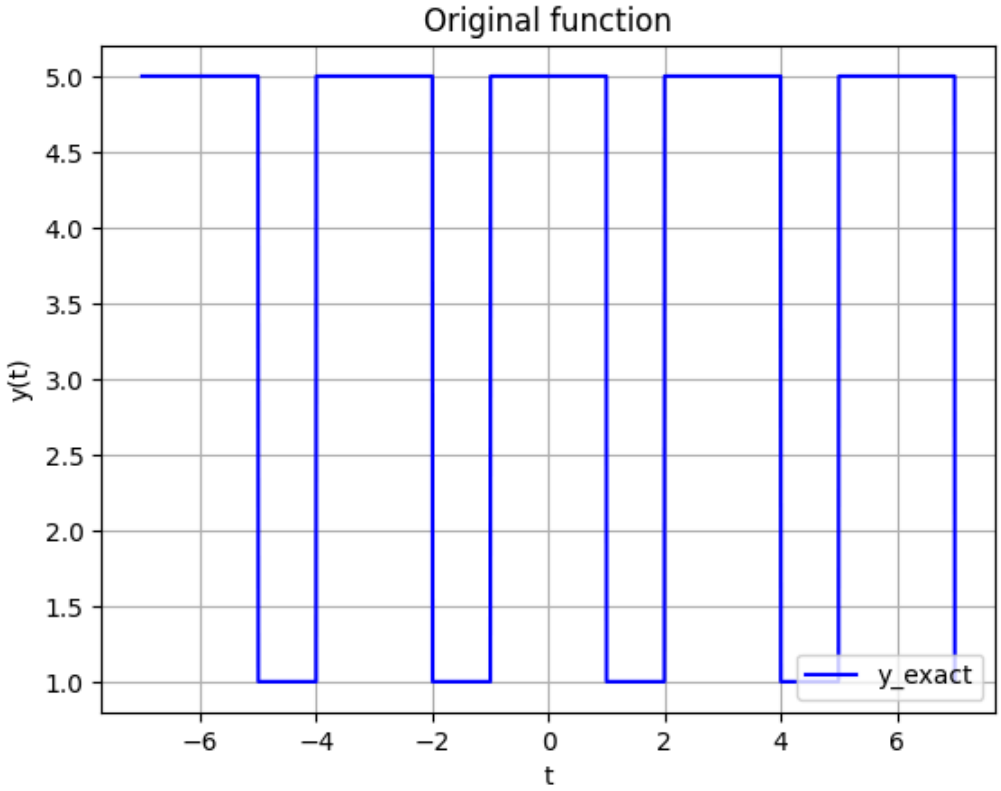
Соответственно, в нашем случае при подстановке это превратится в ...

$$T = 3$$

$$f(t) = \begin{cases} 1, & t \in [1, 2) \\ 5, & t \in [2, 4) \end{cases}$$

Зададим область определения по t, построим график

In [ ]:
```
t = np.linspace(-7, 7, 3000)
plot(t, step_func)
```




```
<Figure size 600x400 with 0 Axes>
```

Посмотрим на частичные суммы следующего вида (вещественные, комплексные)

$$F_N(t) = \frac{a_0}{2} + \sum_{n=1}^{N} (a_n cos(\omega_n t) + b_n sin(\omega_n t)),$$

$$G_N(t) = \sum_{n=-N}^{N} c_n e^{i\omega_n t},$$

где $\omega_n = 2\pi n / T$

Коэффициенты на произвольном отрезке $[h, h + T]$ будут вычисляться по следующим формулам

$$a_n = \frac{2}{T}\int_h^{h+T}\left[f(t)cos(\omega_n t)\right]dt$$
$$b_n = \frac{2}{T}\int_h^{h+T}\left[f(t)sin(\omega_n t)\right]dt$$
$$c_n = \frac{1}{T}\int_h^{h+T}\left[f(t)e^{-i\omega_n t}\right]dt$$

Вычислим коэффициенты a_n, b_n, c_n для $n = \{0, 1, 2\}$ вручную, возьмём длину промежутка $T = 5$, а начало $h = 0$

Так как step-функция всегда константа, обозначим в нашем случае за M, после вынесем её, в итоге вычисления упрощаются до интеграла тригонометрической функции в разных пределах...

$$a_n = \frac{2}{3}\int_0^5 Mcos(\omega_n t)dt = \frac{2}{3}\left[5\int_2^4 cos(\frac{2\pi nt}{3})dt + 1\int_1^2 cos(\frac{2\pi nt}{3})dt\right] = \frac{1}{\pi n}(sin(\frac{4\pi n}{3}) - sin(\frac{2\pi n}{3})) + 5(sin(\frac{8\pi n}{3}) - sin(\frac{4\pi n}{3}))$$

$$b_n = \frac{2}{3}\int_0^5 Msin(\omega_n t)dt = \frac{2}{3}\left[5\int_2^4 sin(\frac{2\pi nt}{3})dt + 1\int_1^2 sin(\frac{2\pi nt}{3})dt\right] = \frac{1}{\pi n}(cos(\frac{2\pi n}{3}) - cos(\frac{4\pi n}{3})) + 5(cos(\frac{4\pi n}{3}) - cos(\frac{8\pi n}{3}))$$

Можно отойти от конкретных коэффициентов, тогда получится такой вид

$$a_n = \frac{1}{\pi n}\left[a(sin(\frac{4\pi n}{3}) - sin(\frac{2\pi n}{3})) + b(sin(\frac{8\pi n}{3}) - sin(\frac{4\pi n}{3}))\right]$$

$$b_n = \frac{1}{\pi n}\left[a(cos(\frac{2\pi n}{3}) - cos(\frac{4\pi n}{3})) + b(cos(\frac{4\pi n}{3}) - cos(\frac{8\pi n}{3}))\right]$$

В случае $f : \mathbb{R} \to \mathbb{R}$ можно облегчить себе жизнь, зная, что $c_i = c_{-i}$ так как компексность должна взаимно уничтожаться, а с веществественными коэффициентами нас ещё связывает следующее соотношение при $n > 0$:

$$c_n = \frac{a_n - ib_n}{2}$$

Решил не писать отдельную программу, а просто вбить такие формулы в вольфрам и ручками менять n, результаты предоставил ниже

Тогда получим следующие коэффициенты

$$\begin{aligned} a_0 &\approx 7.33 & b_0 &\approx 0 \\ a_1 &\approx 2.2 & b_1 &\approx 0 \\ a_2 &\approx -1.1 & b_2 &\approx 0 \\ c_{-2} &\approx -0.551 \\ c_{-1} &\approx 1.11 \\ c_0 &\approx 3.66 \\ c_1 &\approx 1.11 \\ c_2 &\approx -0.551 \end{aligned}$$

Сравним с коэффициентами, которые посчитала программа:

In [ ]:
```
# --> [t0 ; t0+T1]
printBeauty(compute_complex_fourier_coeffs(step_func, 2, t0, T1))
```

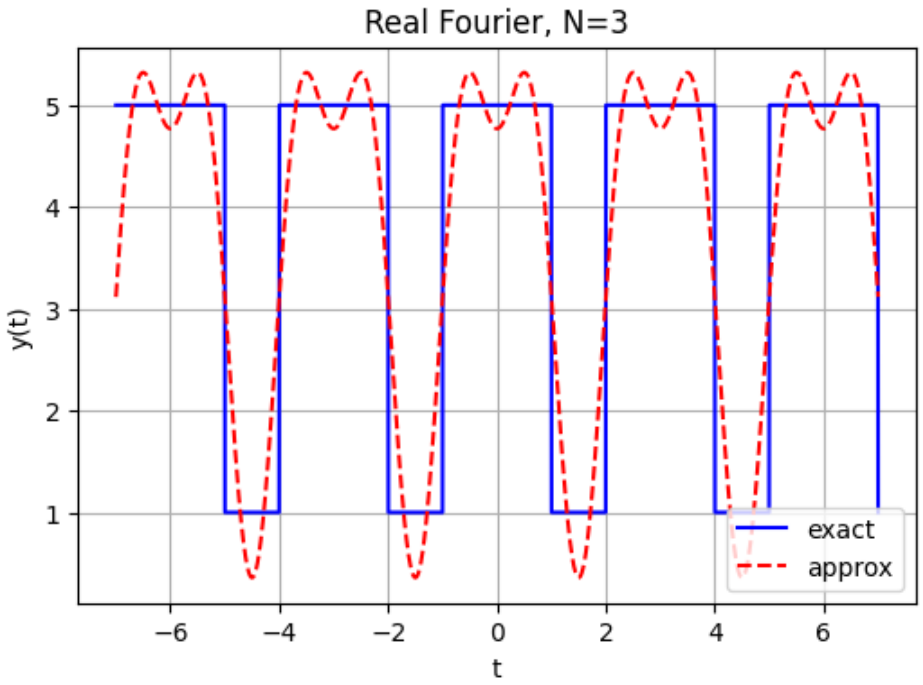
```
printBeauty(compute_real_fourier_coeffs(step_func, 2, t0, T1), float)
```

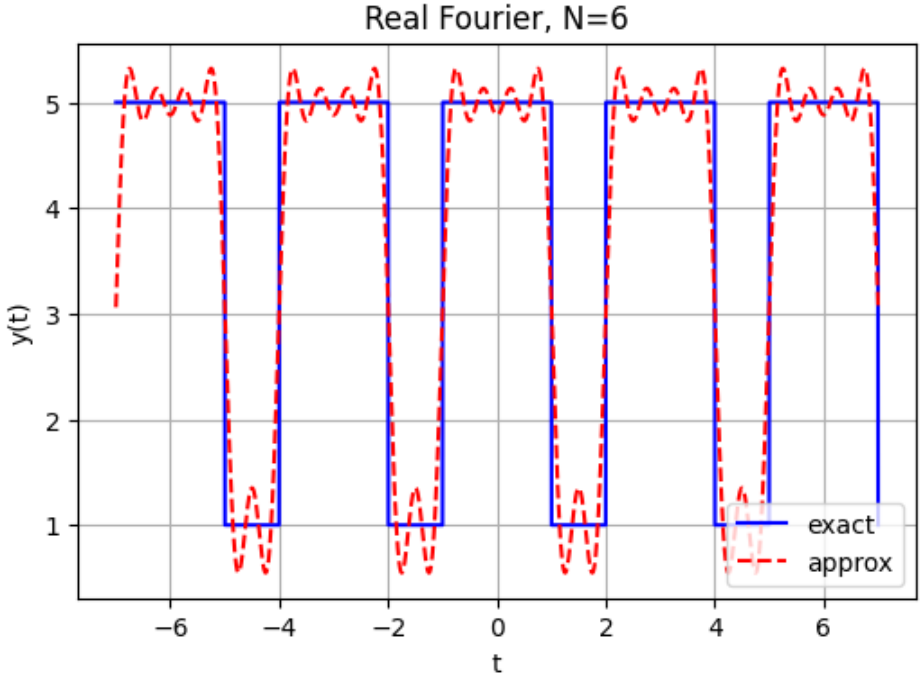
```
-0.5513+0.0000j
 1.1027+0.0000j
 3.6667+0.0000j
 1.1027-0.0000j
-0.5513-0.0000j

 7.3333 0.0
 2.2053 0.0
-1.1027 0.0
```

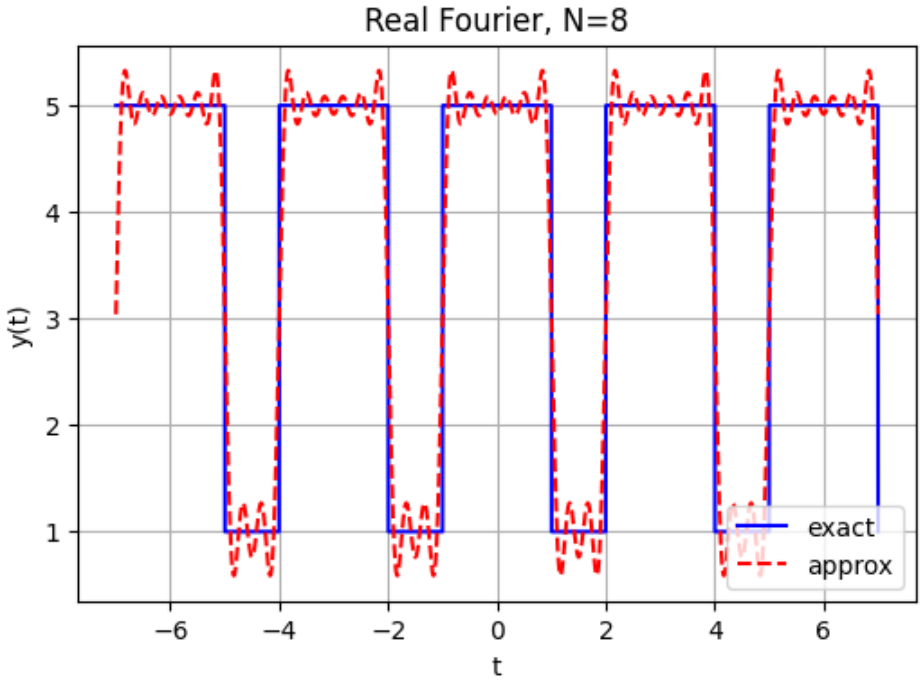
Как можно заметить, мы получили семь комплексных коэффициентов, потому что в коде они идут парами $(c_{-2}, c_{-1}, c_0, c_1, c_2)$, а также парные вещественные коэффициенты (a_n, b_n). И они совпали!

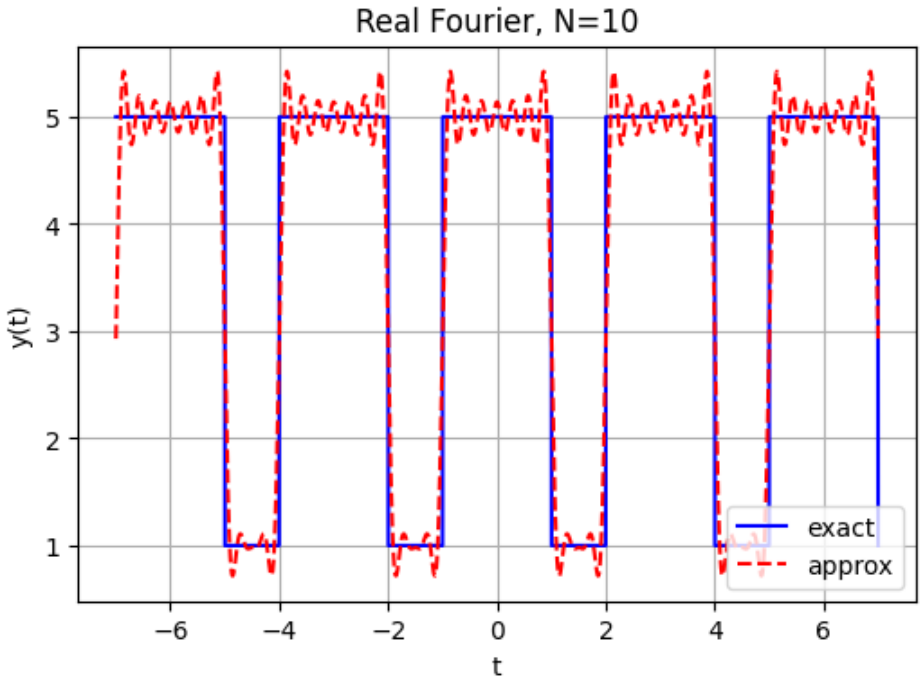
Построим графики $F_N(t)$ и $G_N(t)$ для пяти различных значений N, попробуем посравнивать их между собой и с исходным графиком функции $f(t)$

In [ ]:
```
plot_real_fourier(step_func,t,[3,6,8,10],t0,T1)
```

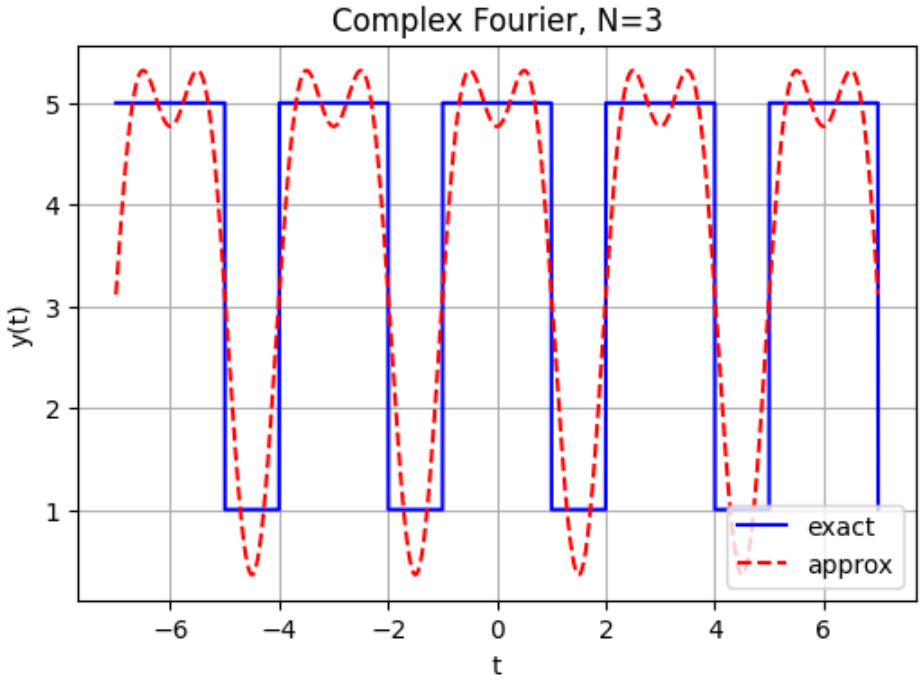


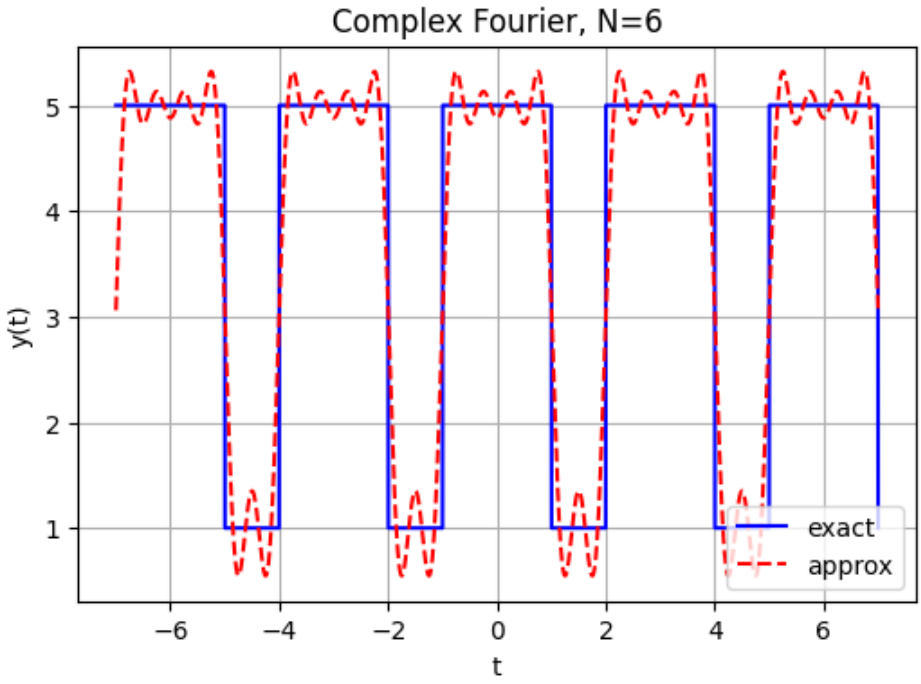

Real Fourier, N=6
y(t)
t
exact
approx

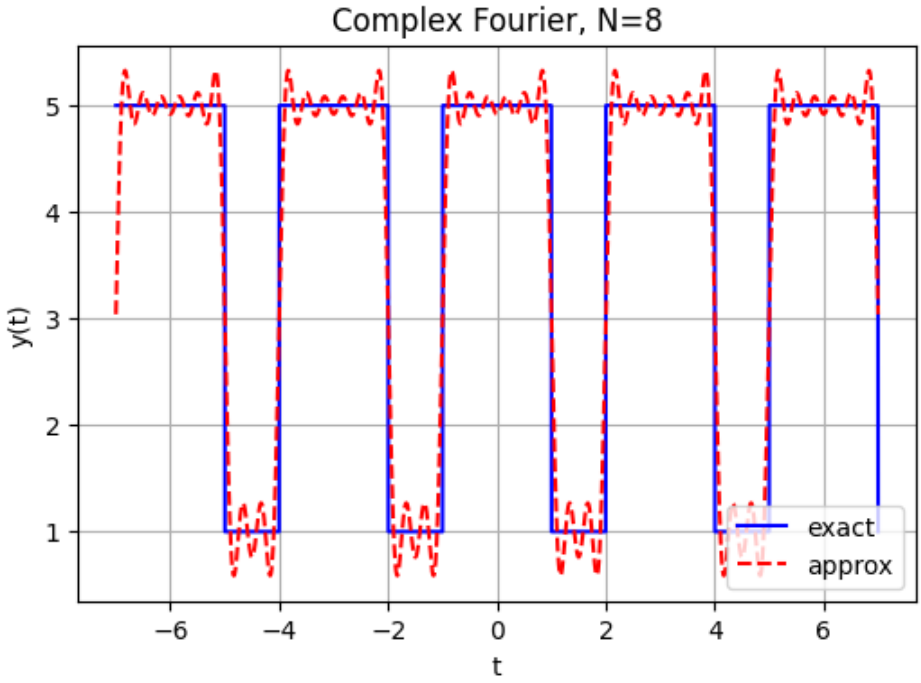

Real Fourier, N=8
y(t)
t
exact
approx

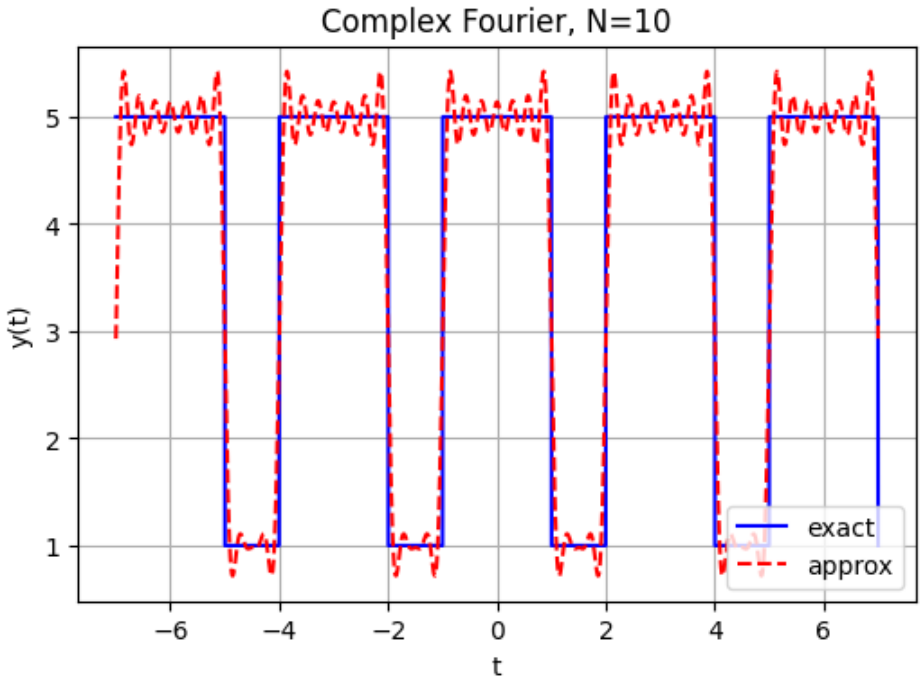


In [ ]:
```
plot_complex_fourier(step_func,t,[3,6,8,10],t0,T1)
```




Complex Fourier, N=6
y(t)
t
exact
approx
5
4
3
2
1
−6
−4
−2
0
2
4
6


Complex Fourier, N=8
y(t)
t
exact
approx
5
4
3
2
1
−6
−4
−2
0
2
4
6



Нетрудно заметить, что комплексная форма идентична по результатам с вещественной. При увеличении количества элементов частичных сумм, как и следовало ожидать, увеличивается количество участвующих и все более уменьшающихся синусоид, которые уточняют график всё лучше и лучше. На последнем графике уже можно заметить небольшое проявление эффекта "Гиббса", так как step-функция имеет резкий скачок.

Теперь проверим выполнение равенства Парсеваля. Так как у нас ортогональный базис, то в правую часть нужно добавить 2π:

$$||f||^2 = 2\pi \sum_{n=-\infty}^{\infty} |c_n|^2$$

Перепишем равенство в более удобной форме для численного сравнения, не забываем, что проверяется это равенство на $[-\pi; \pi]$

$$\left[\int_{-\pi}^{\pi} |f(x)|^2 dx - 2\pi \sum_{n=-\infty}^{\infty} |c_n|^2 \right] \to 0$$

In [ ]:
```
def compute_complex_fourier_coeffs_sum(func, N, h, T):
    return sum([ np.real(cn)**2 + np.imag(cn)**2 for cn in compute_complex_fourier_coeffs(f

def compute_real_fourier_coeffs_sum(func, N, h, T):
    coeffs = compute_real_fourier_coeffs(func, N, h, T)
    number = 0.5 * (coeffs[0][0]**2)
    for an, bn in coeffs[1:]:
        number += an**2 + bn**2

    return np.real(number)


def check_parseval(func, x, h, T, N):
    # norm of the function
    INT = spi.quad(lambda t: func(t)**2, -np.pi, np.pi)[0]
    # squared coeffs
    COEFFS =  2*np.pi * compute_complex_fourier_coeffs_sum(func, N, h, T)
    print(f"N = {N} --> coeff {COEFFS:.5f} and int {INT:.5f}")
    return abs(INT - COEFFS)
```

```
def check_parseval2(func, x, h, T, N):
    INT2 = (1/np.pi)*spi.quad(lambda t: func(t)**2, -np.pi, np.pi)[0]
    COEFFS2 = compute_real_fourier_coeffs_sum(func, N, h, T)
    print(f"N = {N} --> coeff {COEFFS2:.5f} and int {INT2:.5f}")
    return abs(INT2-COEFFS2)

T = 2*np.pi
h = -np.pi
for n in [10,20,50,70,90,100]:
    print(f"delta: {check_parseval(step_func, t, h, T,n):.5f}")
```

```
N = 10 --> coeff 107.31795 and int 109.07988
delta: 1.76193
N = 20 --> coeff 108.03250 and int 109.07988
delta: 1.04738
N = 50 --> coeff 108.69692 and int 109.07988
delta: 0.38295
N = 70 --> coeff 108.80859 and int 109.07988
delta: 0.27129
N = 90 --> coeff 108.88050 and int 109.07988
delta: 0.19938
N = 100 --> coeff 108.90247 and int 109.07988
delta: 0.17741
```

Также на просторах интернета можно найти другую форму равенства Парсеваля, для разнообразия применим её к следующим функциям. В коде её реализовали сверху под именем *check_parseval2*

$$\frac{1}{2}a_0^2 + \sum_{n=1}^{\infty}(a_n^2 + b_n^2) = \frac{1}{\pi}\int_{[-\pi;\pi]} f^2(x)dx$$

При дальнейшем кратном увеличении N вычислительное время становится довольно большим из-за использования интегрирования из сторонних библиотек для поиска коэффициентов и нормы функции, однако это ещё усугубляется тем, что вычисления питона немного сходят с ума из-за подсчитывания большого количества коэффициентов(???), пример такого "безумства" зарегистрировал в таблице ниже. Проверю те же вычисления в матлабе.

N	Delta	Норма	Сумма коэффициентов
500	48388	109.07	48497
1000	90835	109.07	90944
2000	94243	109.07	94352

Чётная функция

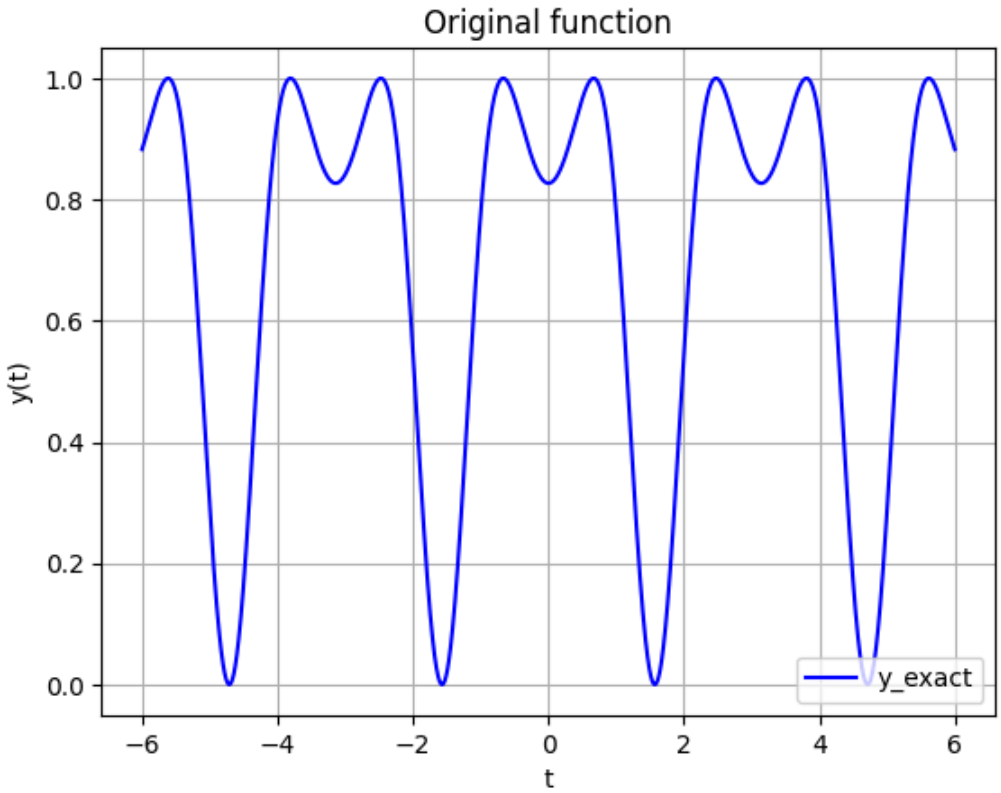
Не будем брать простые фунции, а поиграемся с композицией... Не трудно заметить, что период $T = 2\pi$ останется тот же.

$$f(t) = sin^2(2cosx)$$

In [ ]:
```
T2 = 2*np.pi
def even_func(t):
   return ( np.sin(2*np.cos(t)) )**2
```

Зададим область определения по t, построим график

In [ ]:
```
t = np.linspace(-6, 6, 7000)
plot(t,  even_func)
```




```
<Figure size 600x400 with 0 Axes>
```

Коэффициенты на произвольном отрезке $[h, h+T]$ будут вычисляться по следующим формулам

$$a_n = \frac{2}{T}\int_h^{h+T}\left[sin^2(2cosx)cos(\omega_n t)\right]dt$$
$$b_n = \frac{2}{T}\int_h^{h+T}\left[sin^2(2cosx)sin(\omega_n t)\right]dt = 0$$
$$c_n = \frac{1}{T}\int_h^{h+T}\left[sin^2(2cosx)e^{-i\omega_n t}\right]dt$$

Для удобства работы с тригонометрией здесь и далее будем работать с промежутком $[-\pi;\pi]$, а значит можно будет формулы переписать следующим образом:

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi}\left[sin^2(2cosx)cos(nt)\right]dt$$
$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi}\left[sin^2(2cosx)sin(nt)\right]dt = 0$$
$$c_n = \frac{1}{2\pi}\int_{-\pi}^{\pi}\left[sin^2(2cosx)e^{-int}\right]dt$$

Все коэффициенты b_n автоматически будут зануляться вследствии того, что заданная функция - чётная, иными словами - скалярное произведение ортогональных функций (sin & cos) равно нулю.

Коэффициенты посчитать вручную будет проблематично - интеграл неберущийся(по словам wolfram alpha), да и не удивительно, композиция функций обычно к добру не ведёт. Тогда

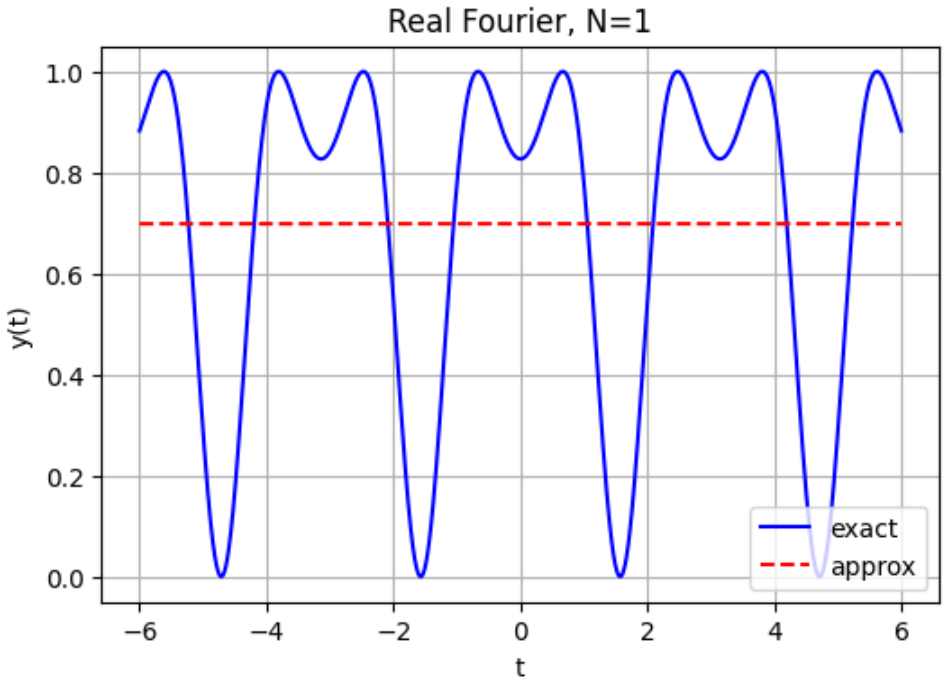
воспользуемся программой: получим семь комплексных коэффициентов и четыре пар вещественных, при $n = \{0, 1, 2, 3\}$

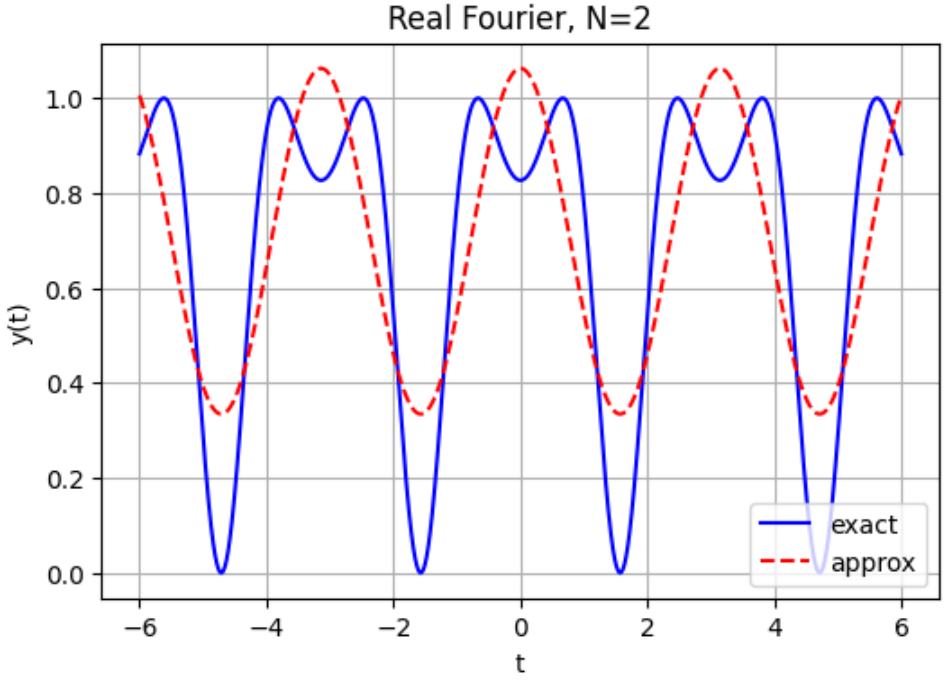
In [ ]:
```
printBeauty(compute_complex_fourier_coeffs(even_func, 3, -np.pi, 2*np.pi))
printBeauty(compute_real_fourier_coeffs(even_func, 3, -np.pi, 2*np.pi), float)
```

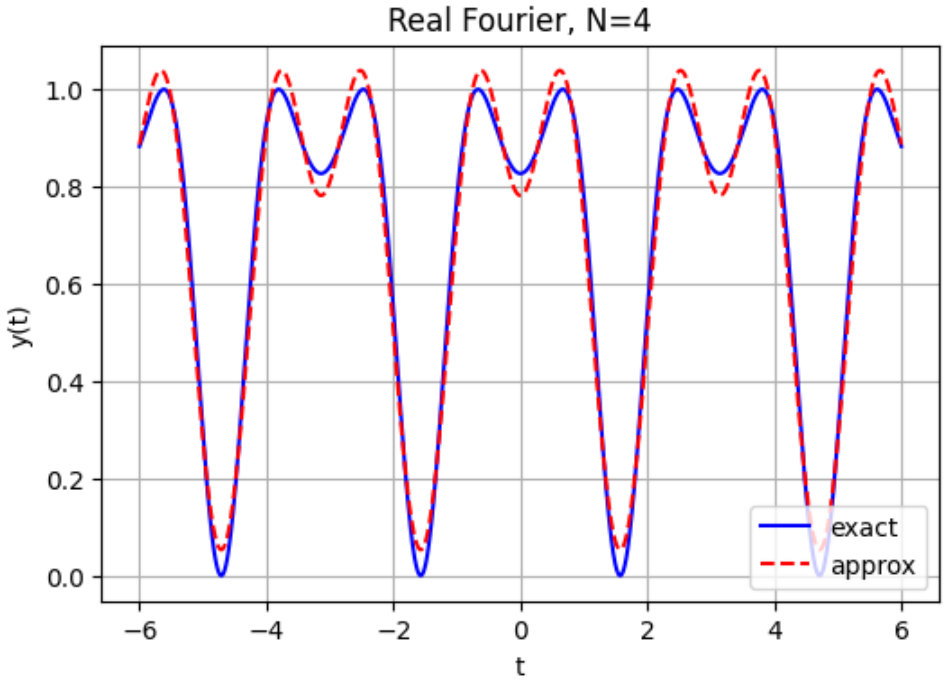
```
 0.0000+0.0000j
 0.1821+0.0000j
-0.0000+0.0000j
 0.6986+0.0000j
-0.0000+0.0000j
 0.1821+0.0000j
 0.0000+0.0000j

 1.3971 0.0
-0.0000 0.0
 0.3641 0.0
 0.0000 0.0
```

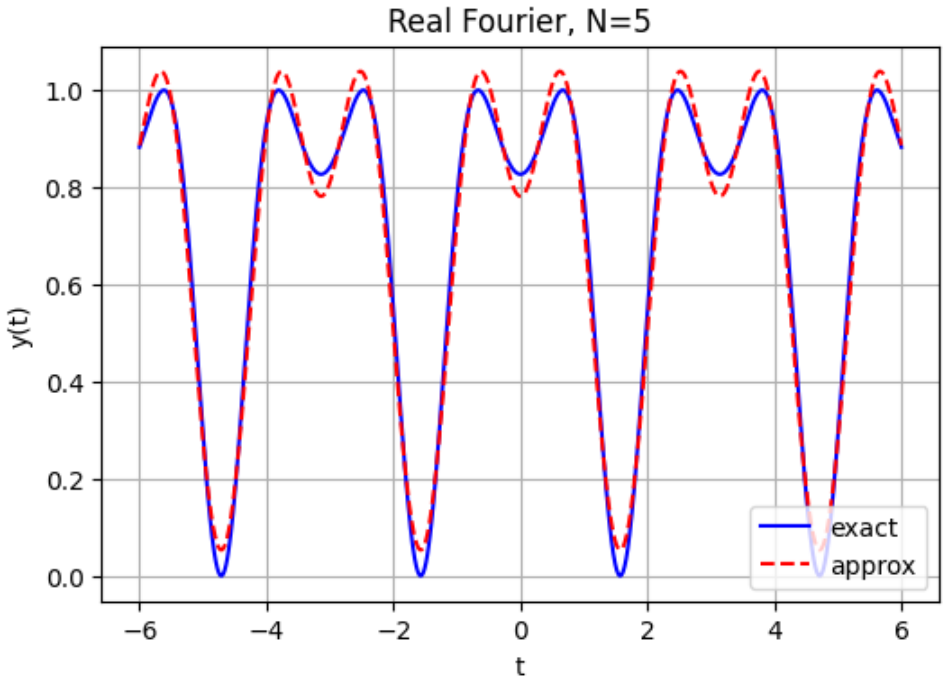
Построим графики частичных сумм при $n = \{1, 2, 3, 4, 5\}$

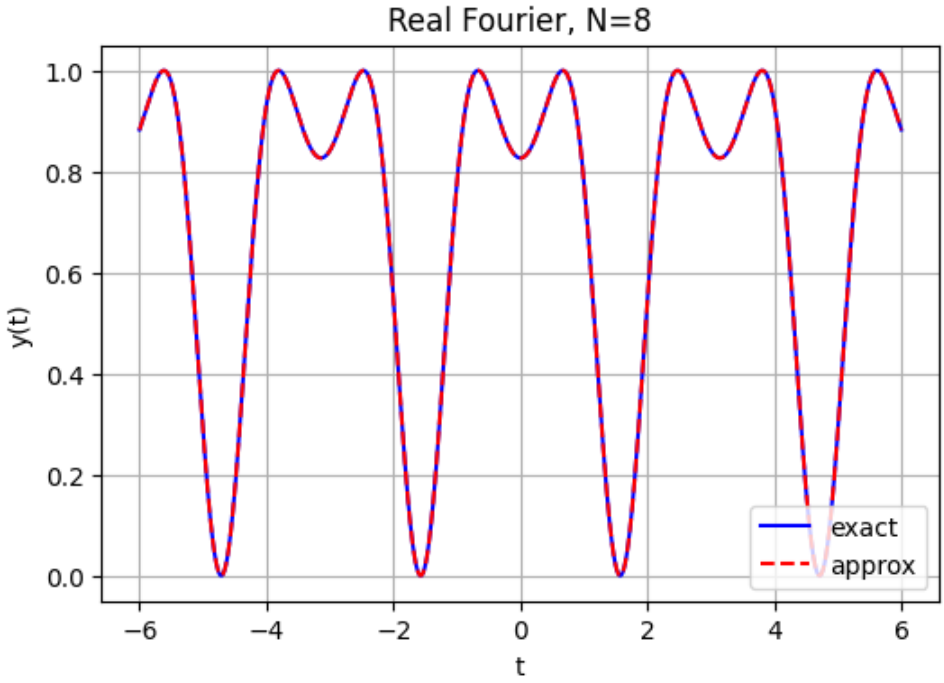
In [ ]:
```
plot_real_fourier(even_func,t,[1,2,4,5,8],-np.pi,2*np.pi)
```




Real Fourier, N=2
y(t)
t
exact
approx
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6

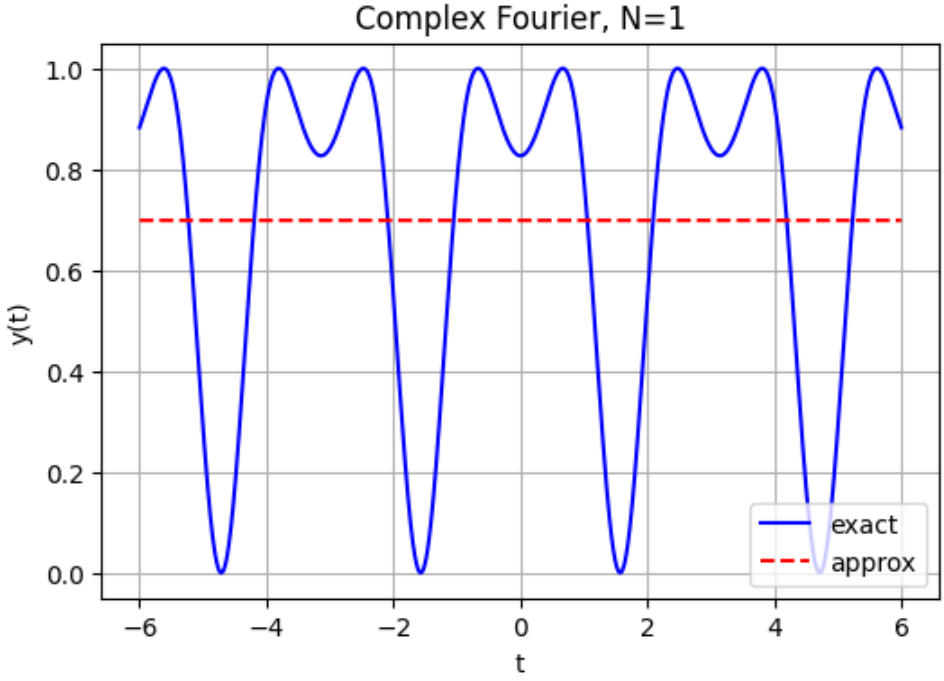

Real Fourier, N=4
y(t)
t
exact
approx
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6

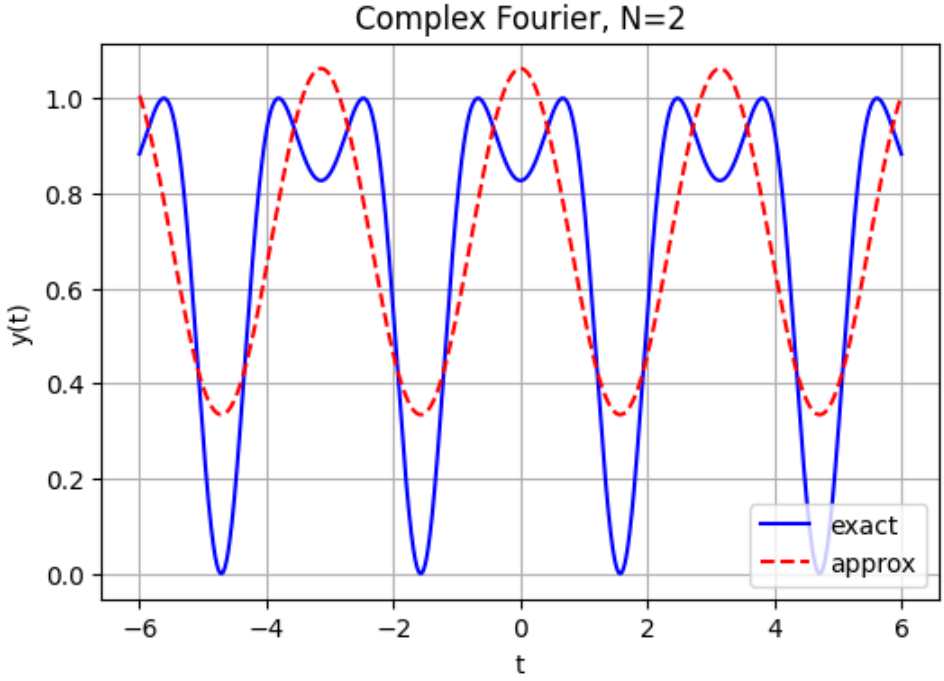


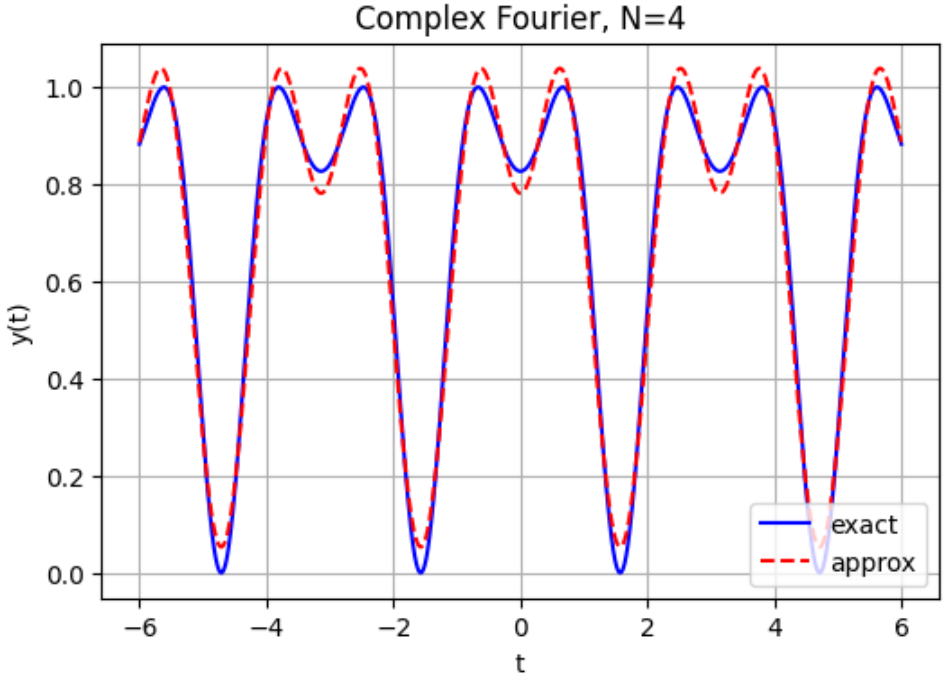


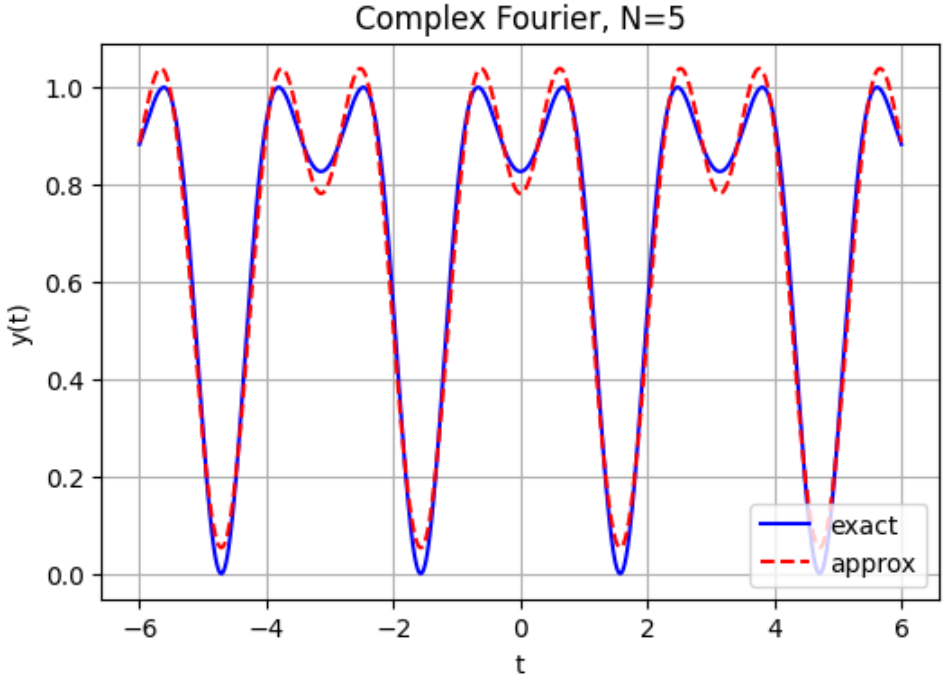
Заметим, что при увеличении N график частичной суммы ряда Фурье довольно быстро приближается к исходной функции. Всего восьми слагаемых хватило, чтобы стать неотличимым от функции.

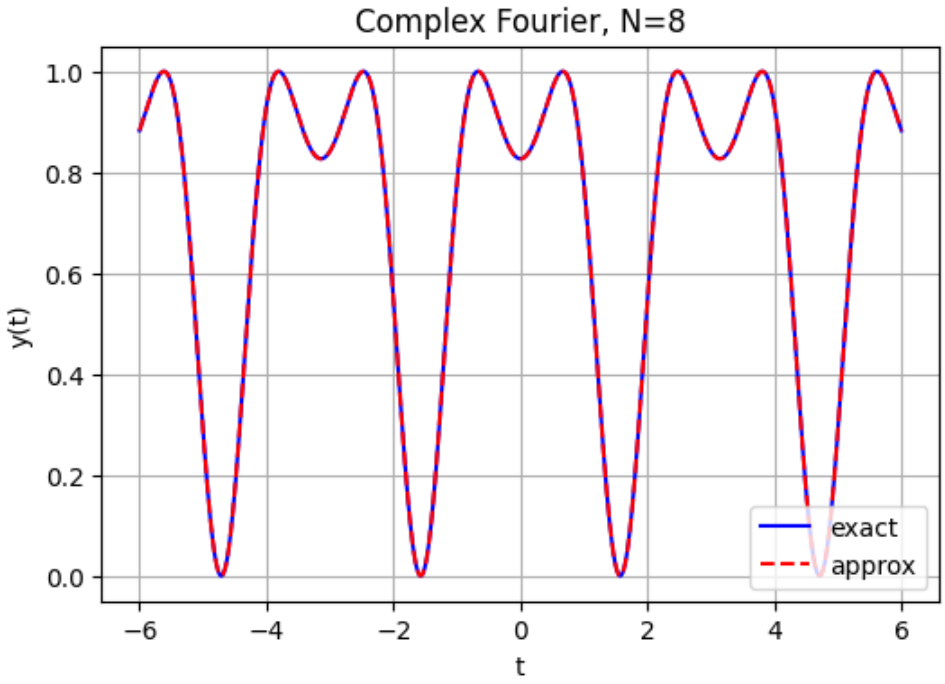
In [ ]:
```
plot_complex_fourier(even_func,t, [1,2,4,5,8],-np.pi,2*np.pi)
```


Complex Fourier, N=1
1.0
0.8
0.6
0.4
0.2
0.0
y(t)
−6
−4
−2
0
2
4
6
t
exact
approx


Complex Fourier, N=2
1.0
0.8
0.6
0.4
0.2
0.0
y(t)
−6
−4
−2
0
2
4
6
t
exact
approx


Complex Fourier, N=4
y(t)
t
exact
approx
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6


Complex Fourier, N=5
y(t)
t
exact
approx
1.0
0.8
0.6
0.4
0.2
0.0
−6
−4
−2
0
2
4
6



В комплексном случае получаем аналогичные картинки, приближения идентичны по причине того, что комплексная запись - идентична с вещественной, обобщает её. В дальнейшем мы ещё не раз убедимся в этом, потому что такие совпадения будут везде при $f : \mathbb{R} \to \mathbb{R}$.

Проверим равенства Парсеваля по формуле ниже с помощью функции *check_parseval2*

$$\frac{1}{2}a_0^2 + \sum_{n=1}^{\infty}(a_n^2 + b_n^2) = \frac{1}{\pi}\int_{[-\pi;\pi]} f^2(x)dx$$

In [ ]:
```
for n in [50,70,90,200,500,1000]:
    print(f"delta: {check_parseval2(even_func, t, h, T,n):.4f}")
```

```
N = 50 --> coeff 1.19006 and int 1.19006
delta: 0.0000
N = 70 --> coeff 1.19006 and int 1.19006
delta: 0.0000
N = 90 --> coeff 1.19006 and int 1.19006
delta: 0.0000
N = 200 --> coeff 1.20772 and int 1.19006
delta: 0.0177
N = 500 --> coeff 4.43310 and int 1.19006
delta: 3.2430
N = 1000 --> coeff 32.09752 and int 1.19006
delta: 30.9075
```

Как можно заметить, тенденция с расхождением при увеличении N - сохранилась, что очень печально :(

Однако до $\approx$ сотого слагаемого нам удаётся добиться схождения аж до пяти знаков - не знаю даже радоваться или плакать

N	Delta	Норма	Сумма коэффициентов
90	0	1.19006	1.19006
200	0.0177	1.1900	1.2007
500	3	1.19	3.24
...	...	...	...

Нечётная функция

Возьмём произведение функций, с периодом $T = 4\pi$

$$f(t) = cos(2x)sin(x/2)$$

In [ ]:
```
T3 = 4*np.pi
def odd_func(t):
    return np.cos(2*t)*np.sin(t/2)
```

In [ ]:
```
# t = (-14,14, 5000)
t = np.linspace(-6, 6, 7000)
```

На промежутке $[-\pi; \pi]$ коэффициенты ряда Фурье будут вычисляться следующим образом

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi}\Big[cos(2x)sin(x/2)cos(nt)\Big]dt = 0$$
$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi}\Big[cos(2x)sin(x/2)sin(nt)\Big]dt$$
$$c_n = \frac{1}{2\pi}\int_{-\pi}^{\pi}\Big[cos(2x)sin(x/2)e^{-int}\Big]dt$$

В отличии от прошлой функции, здесь мы уже можем вычислить неопредёленный интеграл и получить общий вид для вычисления коэффициентов, но так как по заданию это не требуется, то перейдём сразу к программному методу. Не забываем, что из-за нечётности функции, все коэффциенты a_n зануляться по той же причине, что и в прошлом пункте.

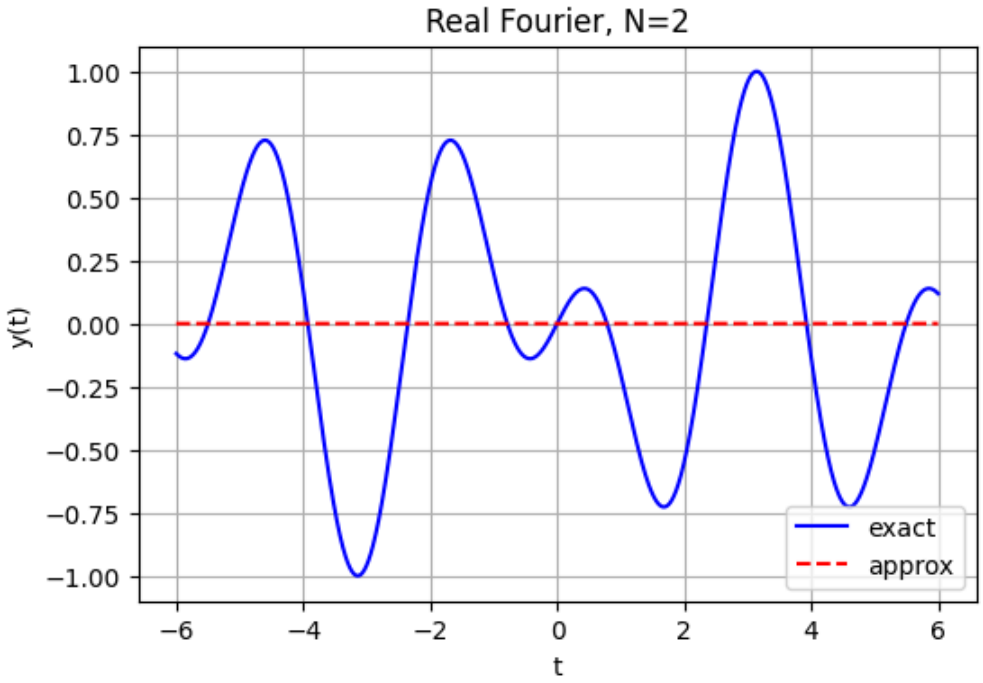
Получим коэффициенты Фурье с помощи программы при $n = \{0, 1, 2, 3\}$

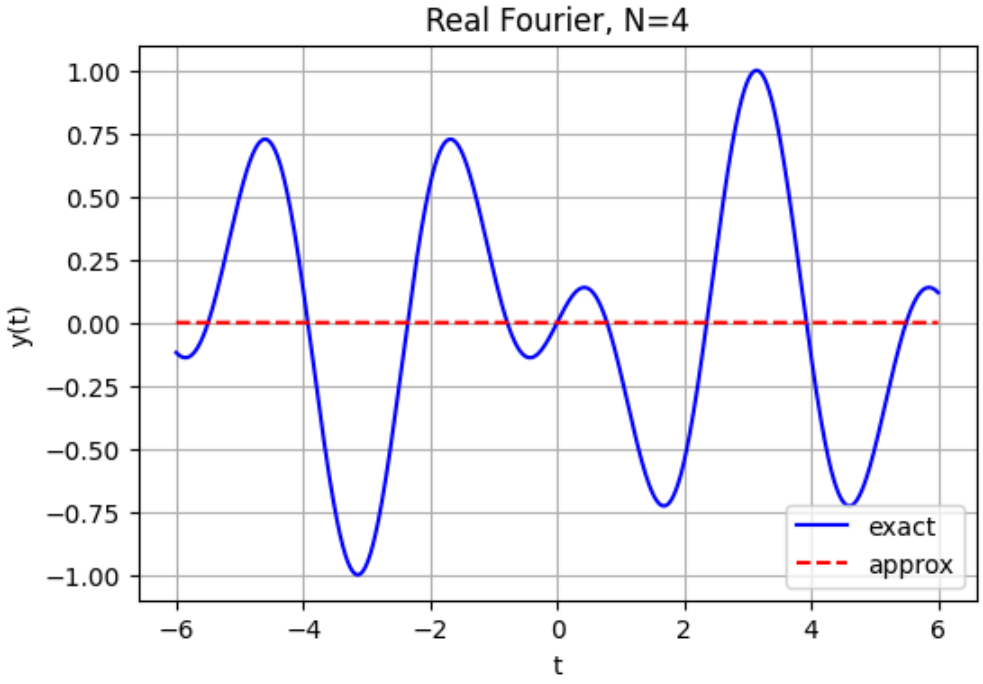
In [ ]:
```
printBeauty(compute_complex_fourier_coeffs(odd_func, 3, -np.pi, 2*np.pi))
printBeauty(compute_real_fourier_coeffs(odd_func, 3, -np.pi, 2*np.pi), float)
```

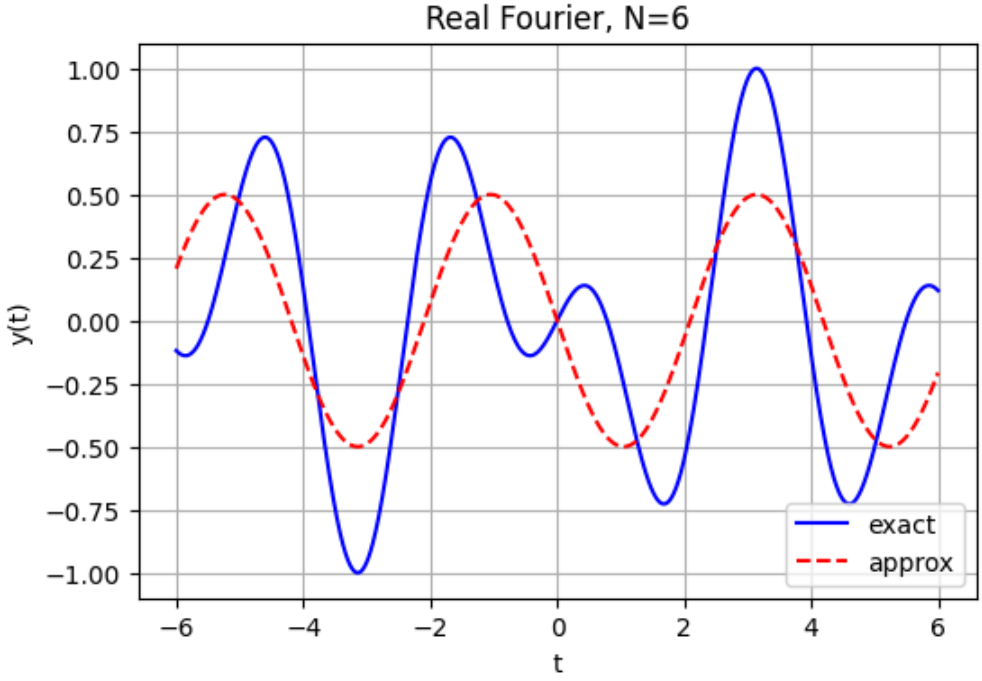
```
0.0000+0.2444j
0.0000-0.0404j
0.0000-0.1576j
0.0000+0.0000j
0.0000+0.1576j
0.0000+0.0404j
0.0000-0.2444j

0.0  0.0000
0.0 -0.3153
0.0 -0.0808
0.0  0.4887
```

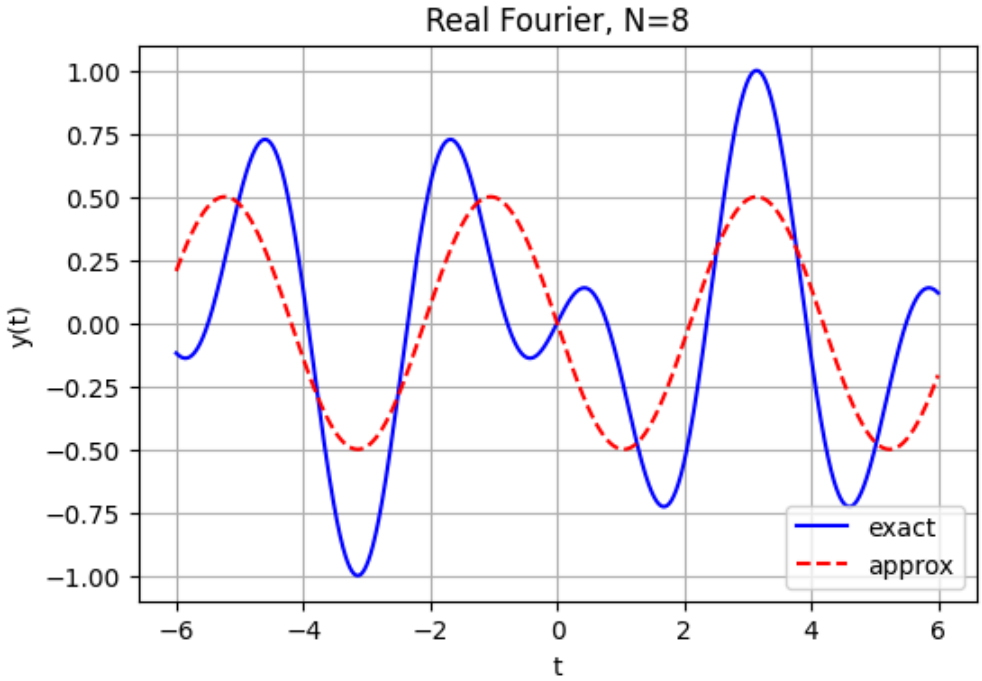
Построим графики частичных сумм при $n = \{1, 2, 3, 4, 5\}$

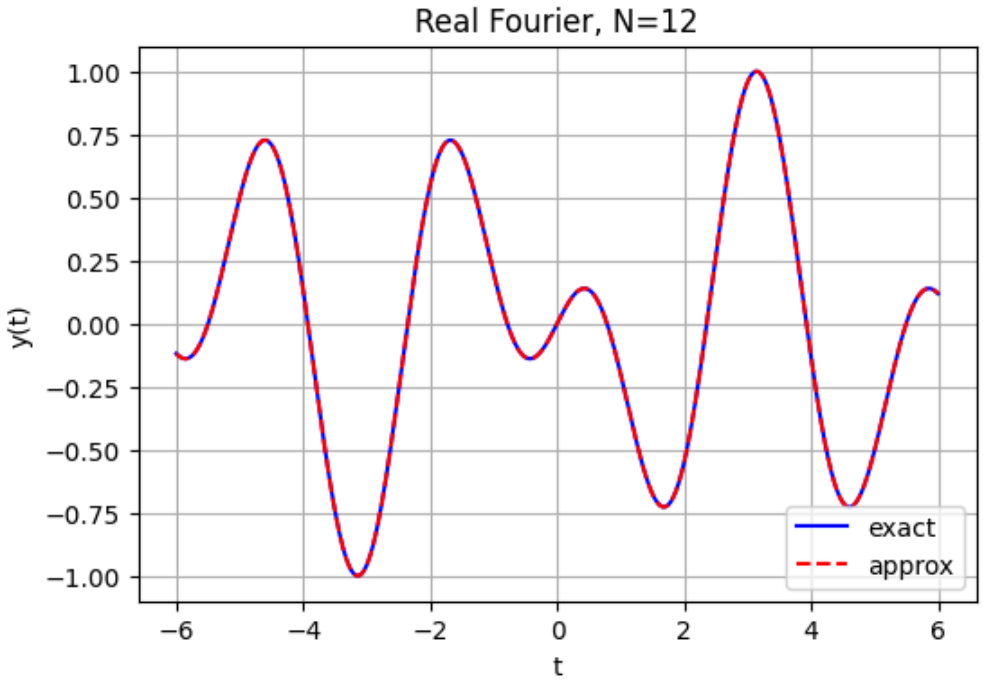
In [ ]:
```
plot_real_fourier(odd_func,t,[2,4,6,8,12],-T3, 2*T3)
```



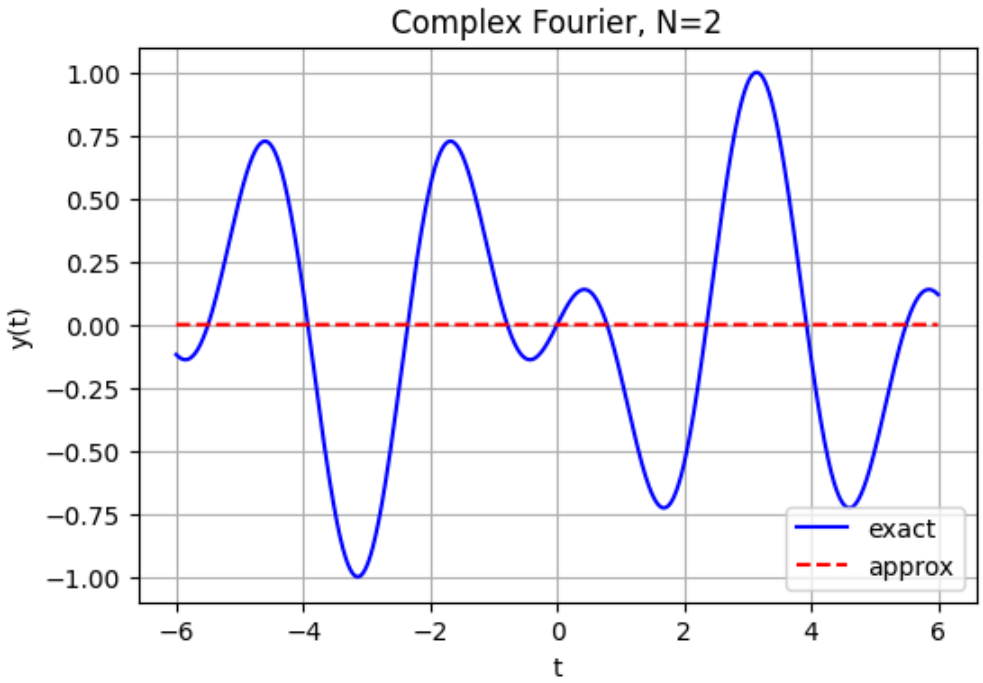


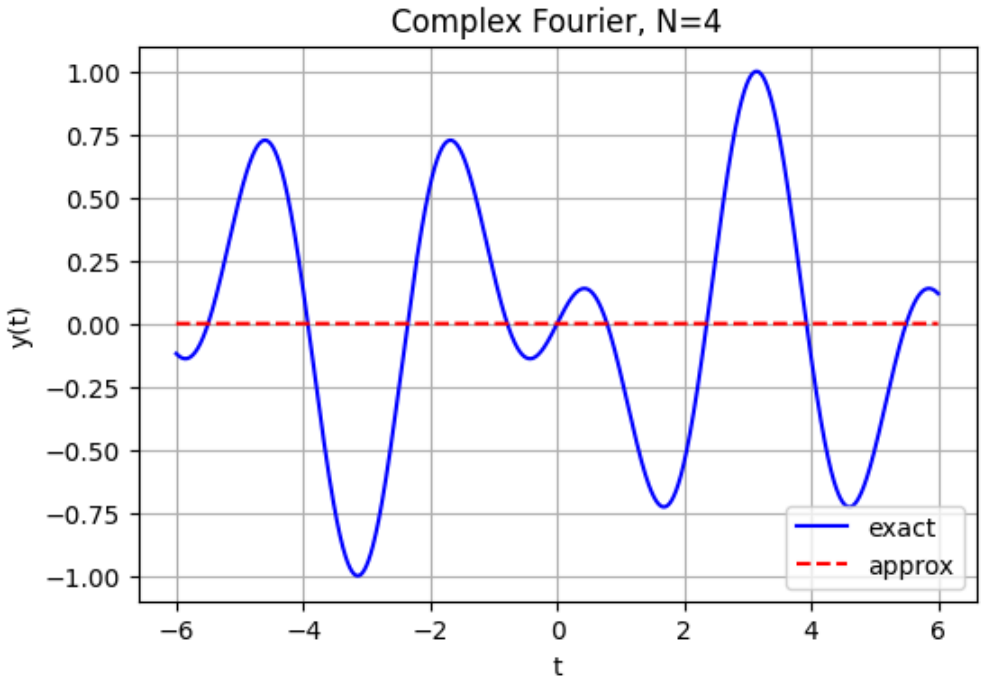

Real Fourier, N=6
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y(t)
−6
−4
−2
0
2
4
6
t
exact
approx

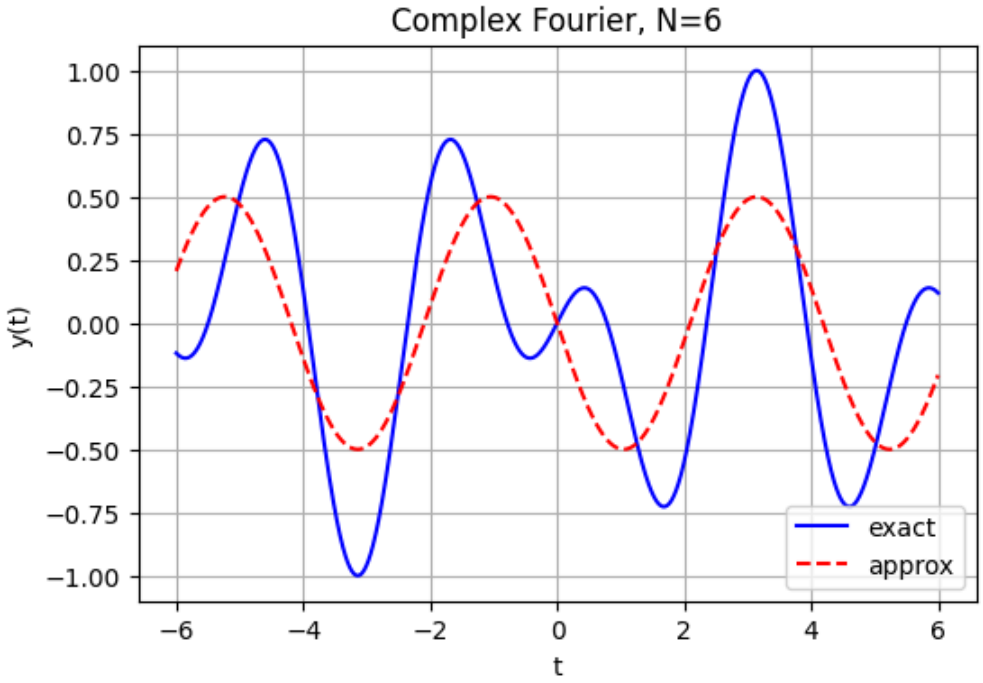

Real Fourier, N=8
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y(t)
−6
−4
−2
0
2
4
6
t
exact
approx

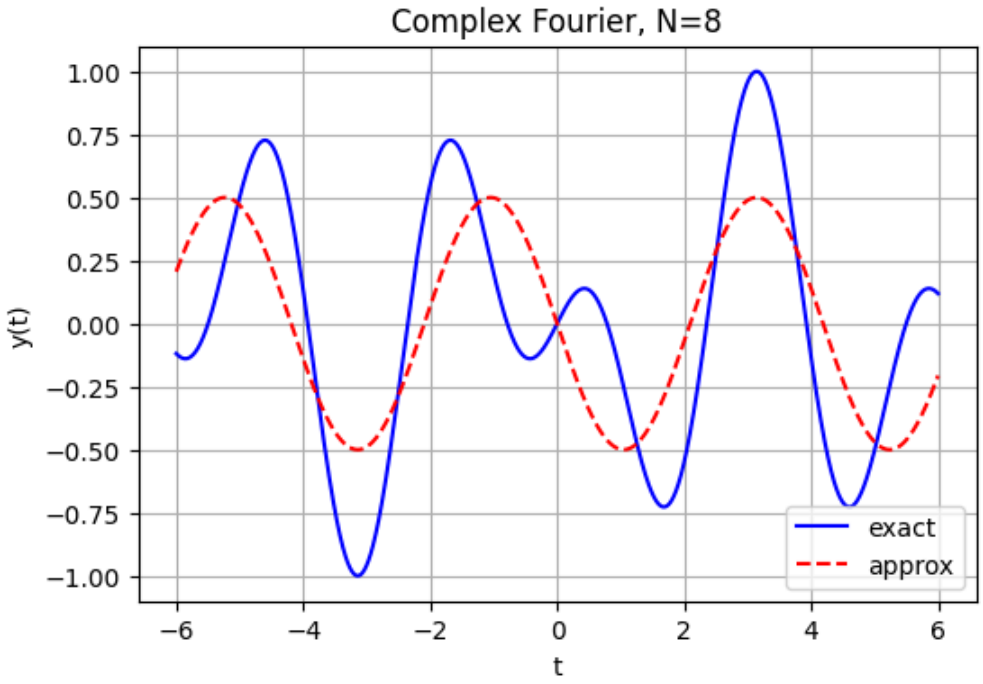


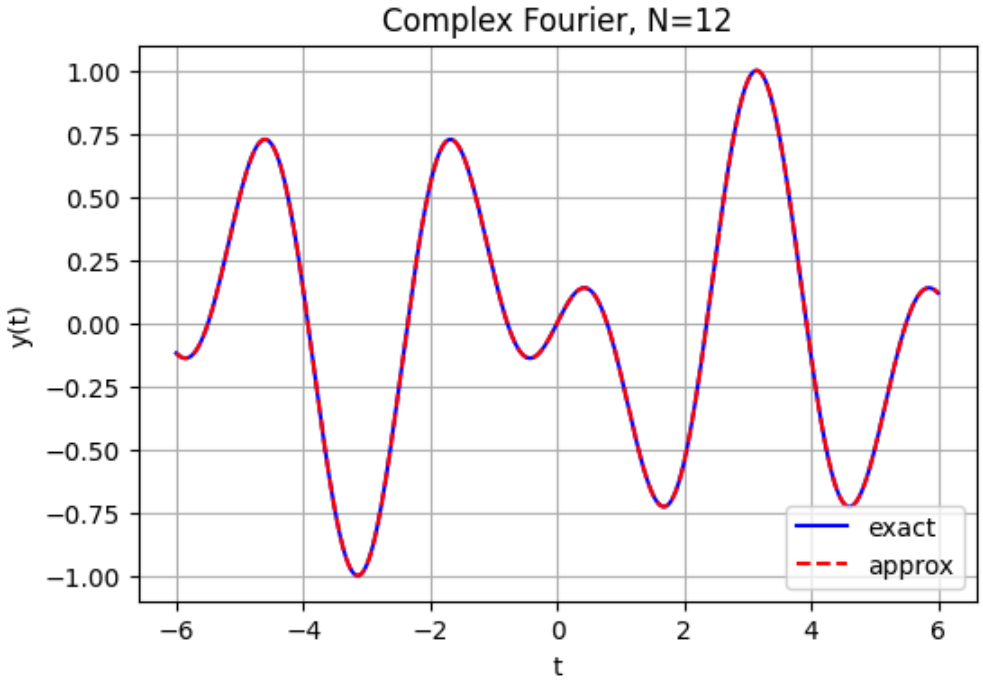
In [ ]:
```
plot_complex_fourier(odd_func,t, [2,4,6,8,12],-T3,2*T3)
```




Complex Fourier, N=4
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y(t)
−6
−4
−2
0
2
4
6
t
exact
approx


Complex Fourier, N=6
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y(t)
−6
−4
−2
0
2
4
6
t
exact
approx





В Этом случае мы также смогли хорошо приблизиться к исходной функции, но уже потребовалось $N = 12$ пар чётных-нечётных гармоник. Вероятно потребовалось больше, потому что функция стала куда сложнее, ведь можно заметить, что до достижения $N = 8$ приближения даже и близко не принимало форму оригинальной функции, "думало" долго

Проверим равенства Парсеваля по формуле ниже с помощью функции *check_parseval2*

$$\frac{1}{2}a_0^2 + \sum_{n=1}^{\infty}(a_n^2 + b_n^2) = \frac{1}{\pi}\int_{[-\pi;\pi]} f^2(x)dx$$

In [ ]:
```
for n in [50,70,90,200,300,500]:
    print(f"delta: {check_parseval2(odd_func, t, h, T,n):.4f}")
```

```
N = 50 --> coeff 0.49197 and int 0.50000
delta: 0.0080
N = 70 --> coeff 0.49425 and int 0.50000
delta: 0.0058
N = 90 --> coeff 0.49552 and int 0.50000
delta: 0.0045
N = 200 --> coeff 0.50423 and int 0.50000
delta: 0.0042
N = 300 --> coeff 1.33376 and int 0.50000
delta: 0.8338
N = 500 --> coeff 1.86551 and int 0.50000
delta: 1.3655
```

В итоге выделим следующие ключевые этапы приближения к равенству:

N	Delta	Норма	Сумма коэффициентов
200	0.0042	0.5	0.5042
300	0.8338	0.5	1.3337
500	1.3655	0.5	1.8655

Ни чётная, ни нечётная функция

$$f(t) = cos^3(t - \frac{3}{4})$$

In [ ]:
```
T4 = 2*np.pi
def not_even_odd_func(t):
    return (np.cos(t - 0.75))**3
```

В отличии от прошлой функции, здесь мы уже можем вычислить неопредёленный интеграл и получить общий вид для вычисления коэффициентов, но так как по заданию это не требуется, то перейдём сразу к программному методу. Не забываем, что из-за нечётности функции, все коэффициенты a_n зануляться по той же причине, что и в прошлом пункте.

На промежутке $[-\pi; \pi]$ коэффициенты ряда Фурье будут вычисляться следующим образом

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi}\left[cos(t - \frac{3}{4})cos(nt)\right]dt$$
$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi}\left[cos(t - \frac{3}{4})sin(nt)\right]dt$$
$$c_n = \frac{1}{2\pi}\int_{-\pi}^{\pi}\left[cos(t - \frac{3}{4})e^{-int}\right]dt$$

В данном случае мы не можем сказать о занулении каких-то коэффициентов заранее, поэтому формула будет максимально общая

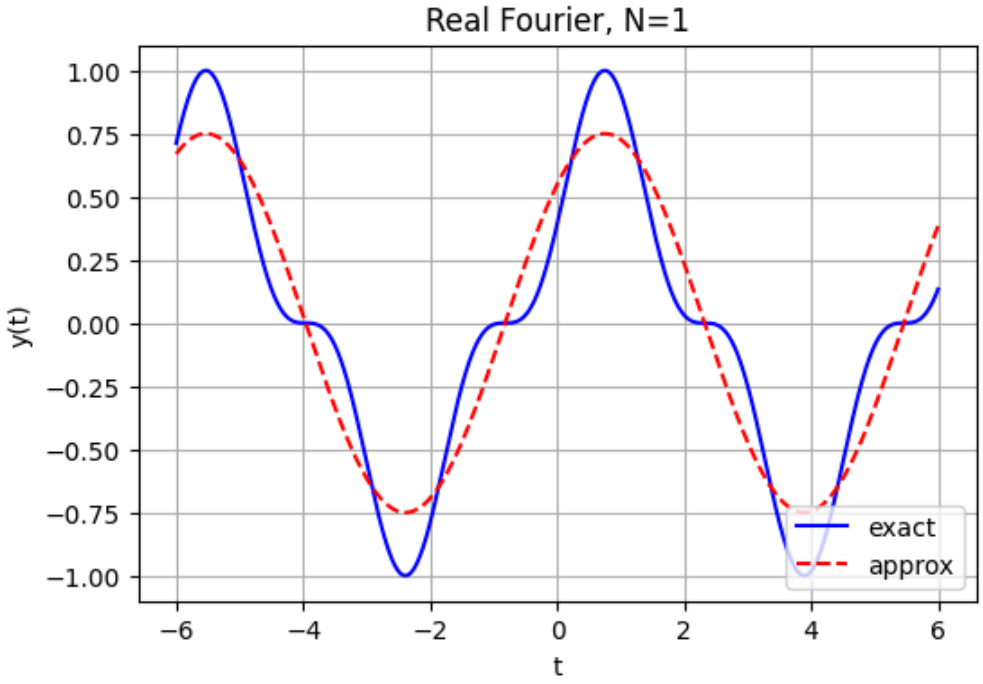
Получим коэффициенты Фурье с помощи программы при $n = \{0, 1, 2, 3\}$

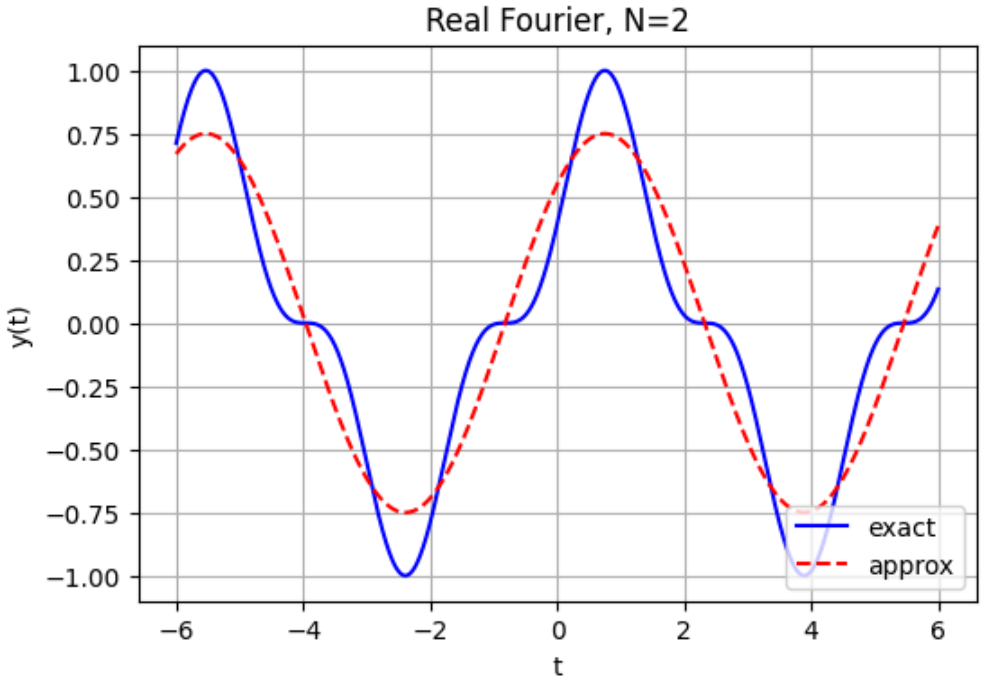
In [ ]:
```
printBeauty(compute_complex_fourier_coeffs(not_even_odd_func, 3, -T4, 2*T4))
printBeauty(compute_real_fourier_coeffs(not_even_odd_func, 3, -T4, 2*T4), float)
```

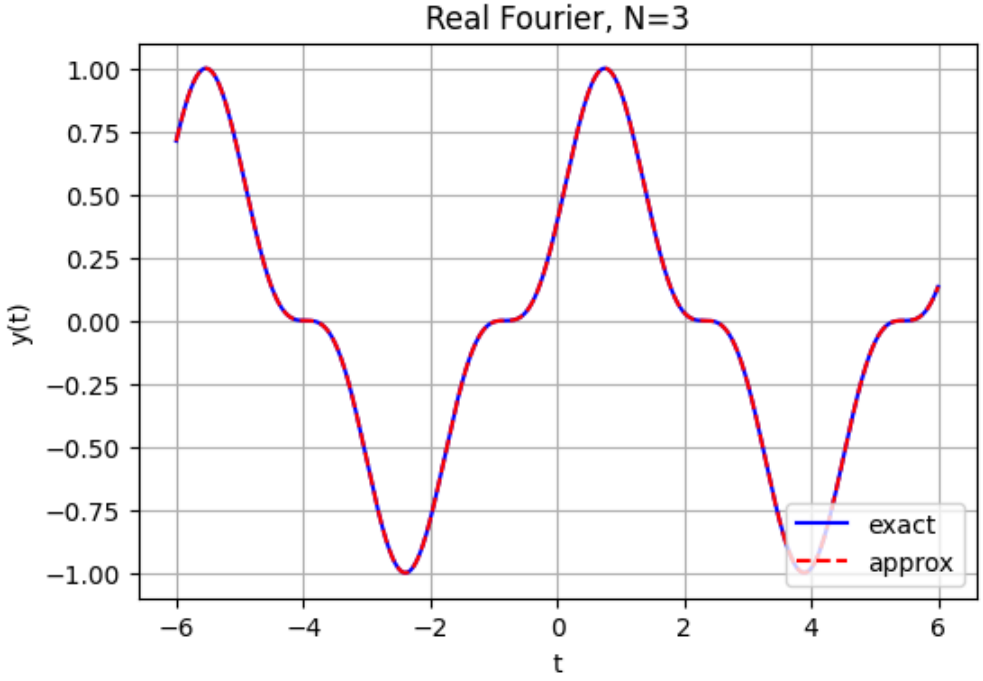
```
-0.0000+0.0000j
 0.2744+0.2556j
 0.0000+0.0000j
 0.0000+0.0000j
 0.0000-0.0000j
 0.2744-0.2556j
-0.0000-0.0000j

 0.0000 0.0000
 0.0000 0.0000
 0.5488 0.5112
-0.0000 0.0000
```

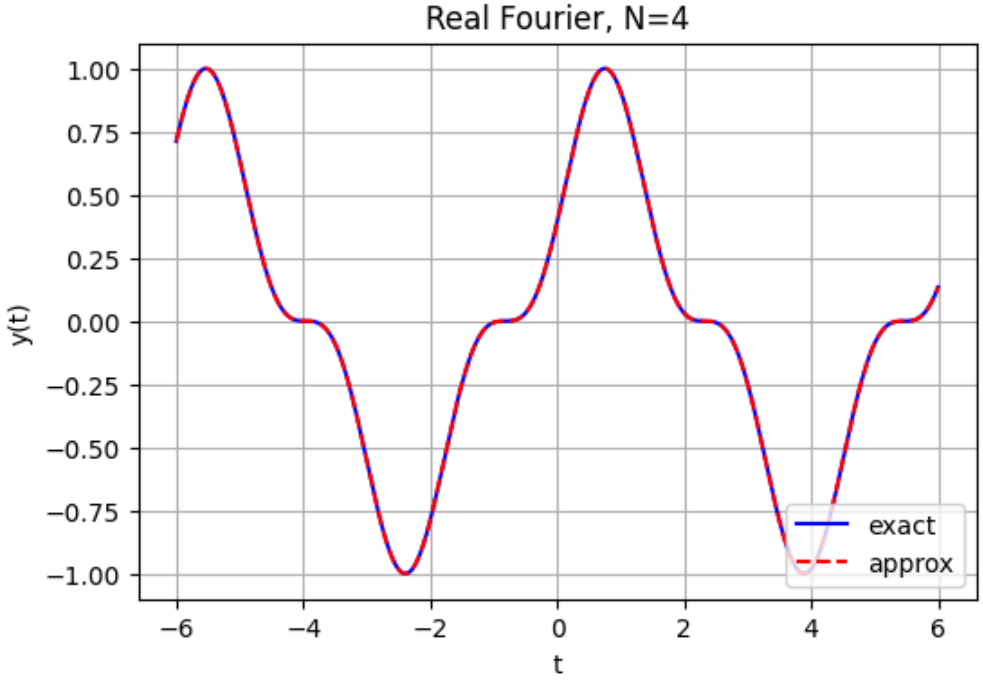
Построим графики частичных сумм при $n = \{1, 2, 3, 4, 5\}$

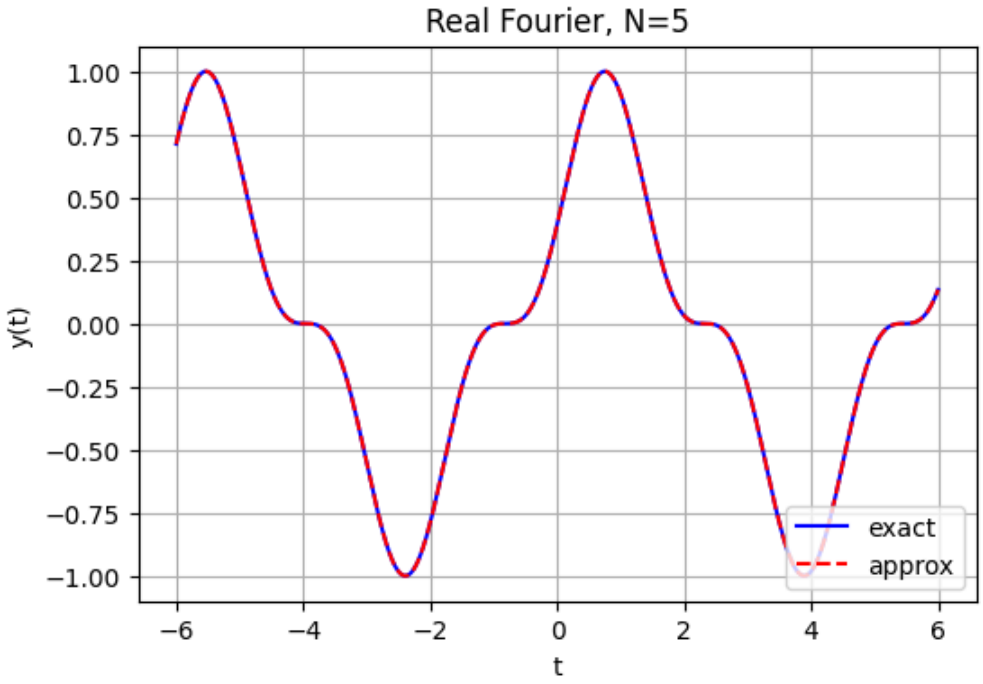
In [ ]:
```
plot_real_fourier(not_even_odd_func,t,[1,2,3,4,5],-np.pi, 2*np.pi)
```



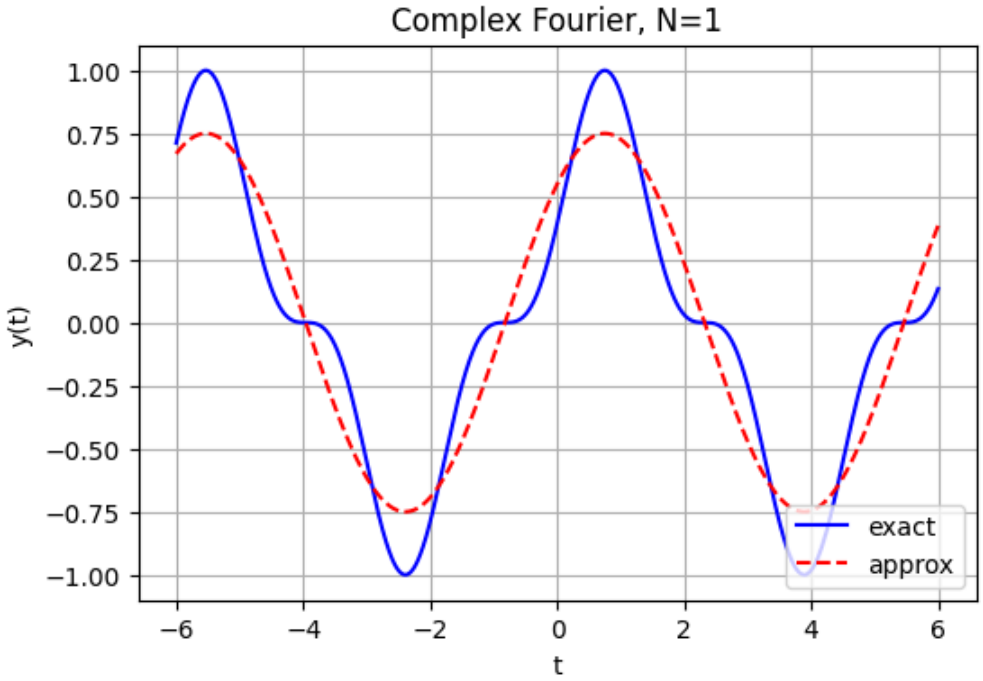


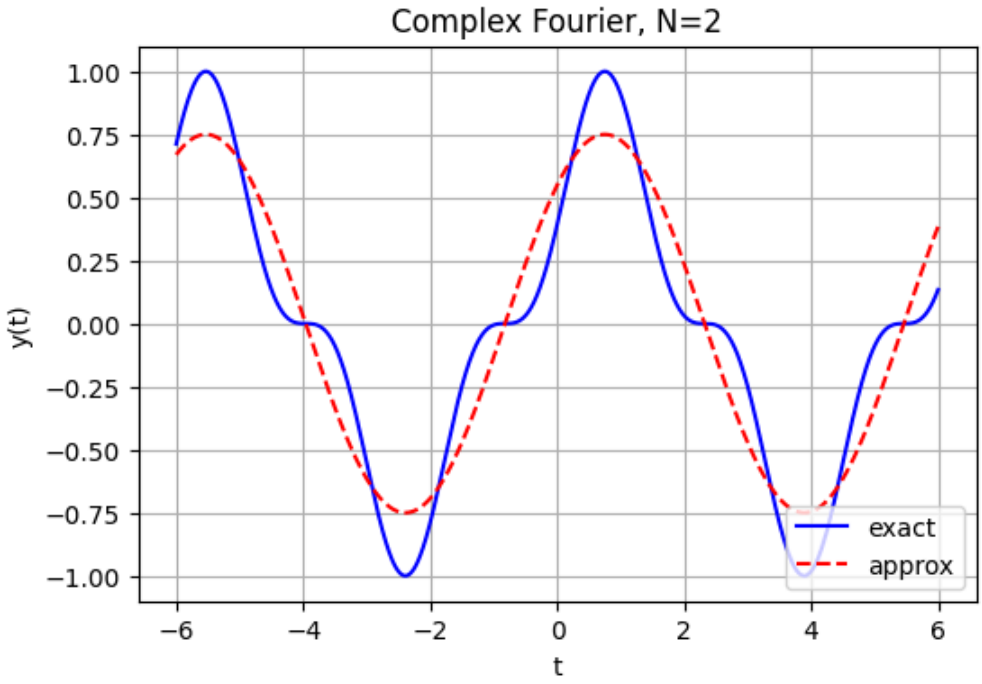

Real Fourier, N=3
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y(t)
−6
−4
−2
0
2
4
6
t
exact
approx

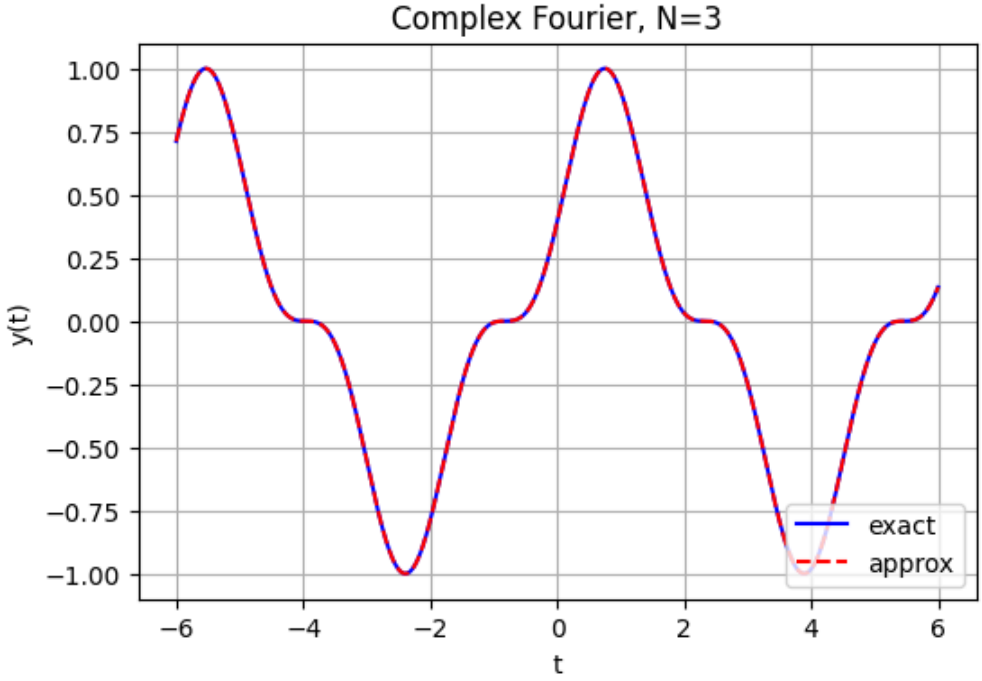

Real Fourier, N=4
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y(t)
−6
−4
−2
0
2
4
6
t
exact
approx

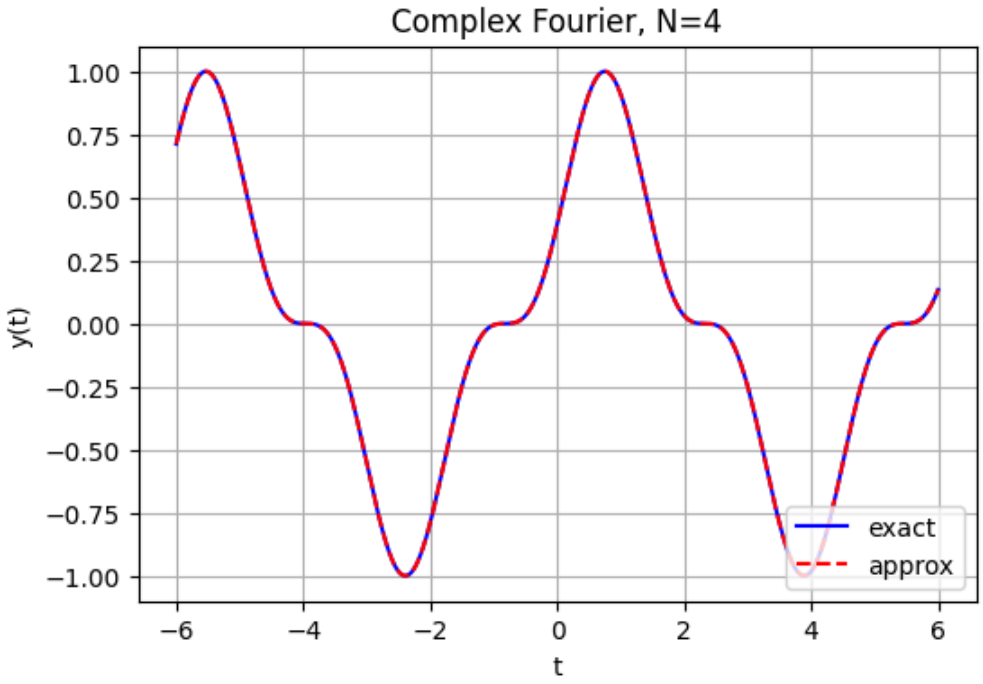


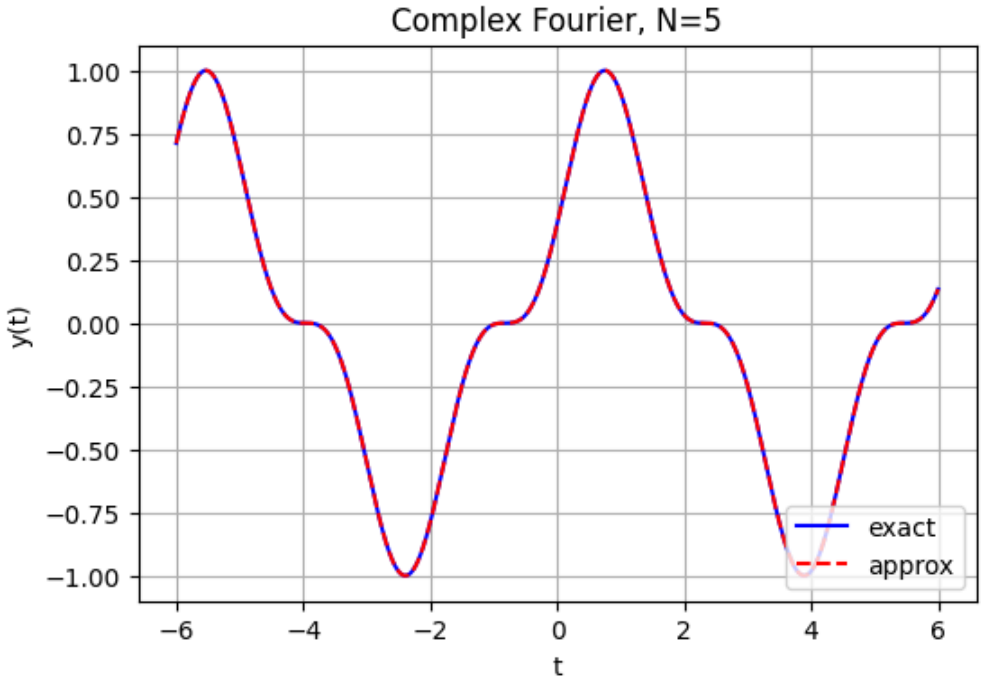
In [ ]:
```
plot_complex_fourier(not_even_odd_func,t,[1,2,3,4,5],-np.pi, 2*np.pi)
```




Complex Fourier, N=2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y(t)
−6
−4
−2
0
2
4
6
t
exact
approx


Complex Fourier, N=3
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y(t)
−6
−4
−2
0
2
4
6
t
exact
approx





Можно заметить, что уже при $N = 3$ мы хорошо (на глаз) смогли приблизиться к исходной функции, хотя вроде функция кажется сложно устроенной. Также снова подмечу, что приближение рядом Фурье в комплексной или вещественной форме здесь идентичны.

Как и с остальными функциями, давайте проверим равенство Парсеваля:

In [ ]:
```
T = 2*np.pi
h = -np.pi
for n in [1,2,3,7,8,10,20,50,100,200]:
    print(f"delta: {check_parseval(not_even_odd_func, t, h, T,n):.5f}")
```

```
N = 1 --> coeff 1.76715 and int 1.96350
delta: 0.19635
N = 2 --> coeff 1.76715 and int 1.96350
delta: 0.19635
N = 3 --> coeff 1.96350 and int 1.96350
delta: 0.00000
N = 7 --> coeff 1.96350 and int 1.96350
delta: 0.00000
N = 8 --> coeff 1.96350 and int 1.96350
delta: 0.00000
N = 10 --> coeff 1.96350 and int 1.96350
delta: 0.00000
N = 20 --> coeff 1.96350 and int 1.96350
delta: 0.00000
N = 50 --> coeff 1.96350 and int 1.96350
delta: 0.00000
N = 100 --> coeff 1.96350 and int 1.96350
delta: 0.00000
N = 200 --> coeff 1.98278 and int 1.96350
delta: 0.01928
```

Очень неожиданно, что при малом количестве слагаемых сумма довольно быстро сошлась до пятого знака, поэтому взял много N, чтобы изучить зависимость подробнее. Основные изменения по N свёл в таблицу ниже...

N	Delta	Норма	Сумма коэффициентов
50	0.00110	837.75804	837.75694
100	0.00014	837.75804	837.75790
200	3.38894	837.75804	841.14698

Задание 2. Комплексная функция

Зададимся числами $R, T > 0$ и рассмотрим функцию $f : \mathbb{R} \to \mathbb{C}$ с периодом T, что...

$$Re(f(t)) = \begin{cases} R, & t \in [-\frac{T}{8}, \frac{T}{8}), \\ 2R - 8Rt/T, & t \in [\frac{T}{8}, \frac{3T}{8}), \\ -R, & t \in [\frac{3T}{8}, \frac{5T}{8}), \\ -6R + 8Rt/T, & t \in [\frac{5T}{8}, \frac{7T}{8}), \end{cases} \qquad Im(f(t)) = \begin{cases} 8Rt/T, & t \in [-\frac{T}{8}, \frac{T}{8}), \\ R, & t \in [\frac{T}{8}, \frac{3T}{8}), \\ 4R - 8Rt/T, & t \in [\frac{3T}{8}, \frac{5T}{8}), \\ -R, & t \in [\frac{5T}{8}, \frac{7T}{8}), \end{cases}$$

In [ ]:
```
R = 10 ; T = 3
```

Тогда в нашем случае...

$$Re(f(t)) = \begin{cases} 10, & t \in [-\frac{T}{8}, \frac{T}{8}), \\ 20 - 80t/3, & t \in [\frac{T}{8}, \frac{3T}{8}), \\ -10, & t \in [\frac{3T}{8}, \frac{5T}{8}), \\ -60 + 80t/3, & t \in [\frac{5T}{8}, \frac{7T}{8}), \end{cases} \qquad Im(f(t)) = \begin{cases} 80t/3, & t \in [-\frac{T}{8}, \frac{T}{8}), \\ 10, & t \in [\frac{T}{8}, \frac{3T}{8}), \\ 40 - 80t/3, & t \in [\frac{3T}{8}, \frac{5T}{8}), \\ -10, & t \in [\frac{5T}{8}, \frac{7T}{8}), \end{cases}$$

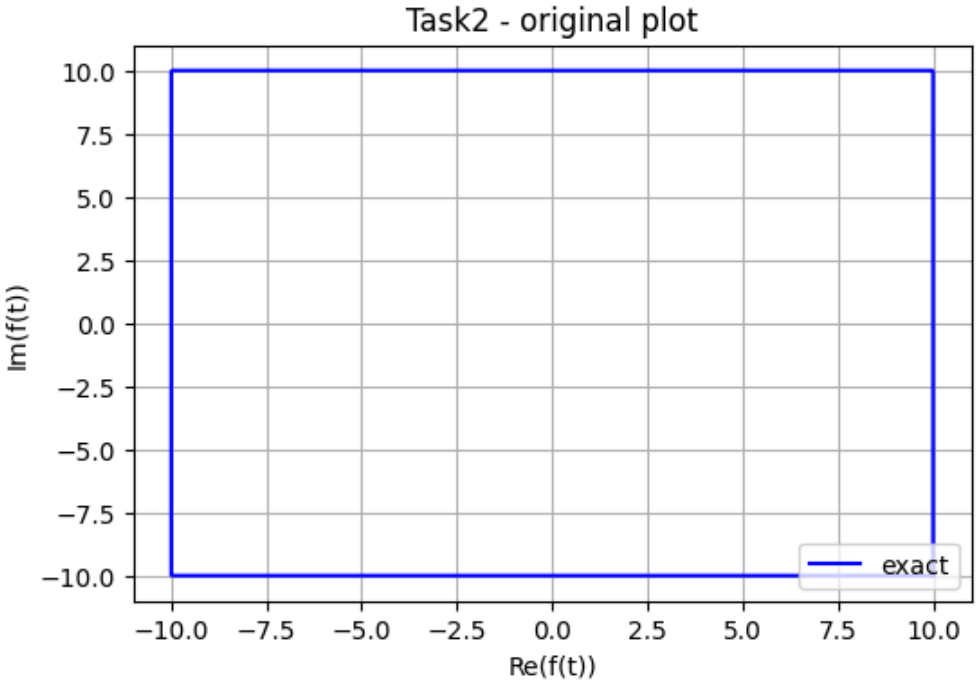
Зададим функцию программно через if-else

In [ ]:
```
def f(t):
    t = (t + T/8)%T - T/8
```

```
    if -T/8 <= t < T/8:
        real = R
        imag = 8*R*t/T
    elif T/8 <= t < 3*T/8:
        real = 2*R - 8*R*t/T
        imag = R
    elif 3*T/8 <= t < 5*T/8:
        real = -R
        imag = 4*R - 8*R*t/T
    elif 5*T/8 <= t < 7*T/8:
        real = -6*R + 8*R*t/T
        imag = -R

    elif 5*T/8 <= t < 7*T/8:
        real = -6*R + 8*R*t/T
        imag = -R
    return real + 1j*imag
# "векторизуем" кусочно-заданную функцию, чтобы потом использовать в других методах
# библиотеки numpy
ff = np.vectorize(f)
```

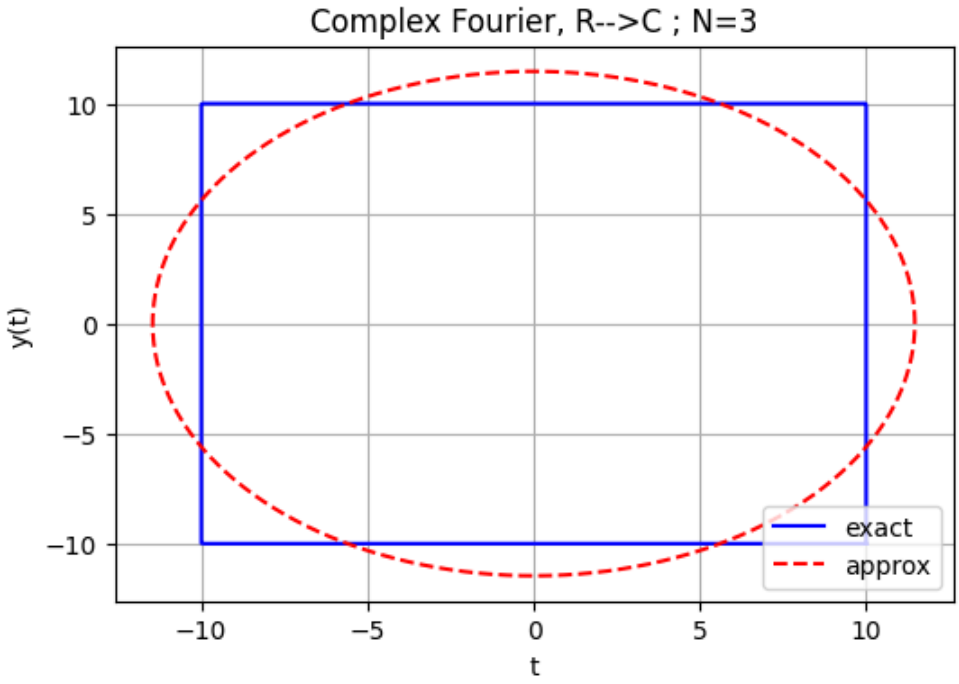
Построим параметрический график $f(t)$ в виде кривой на комплексной плоскости

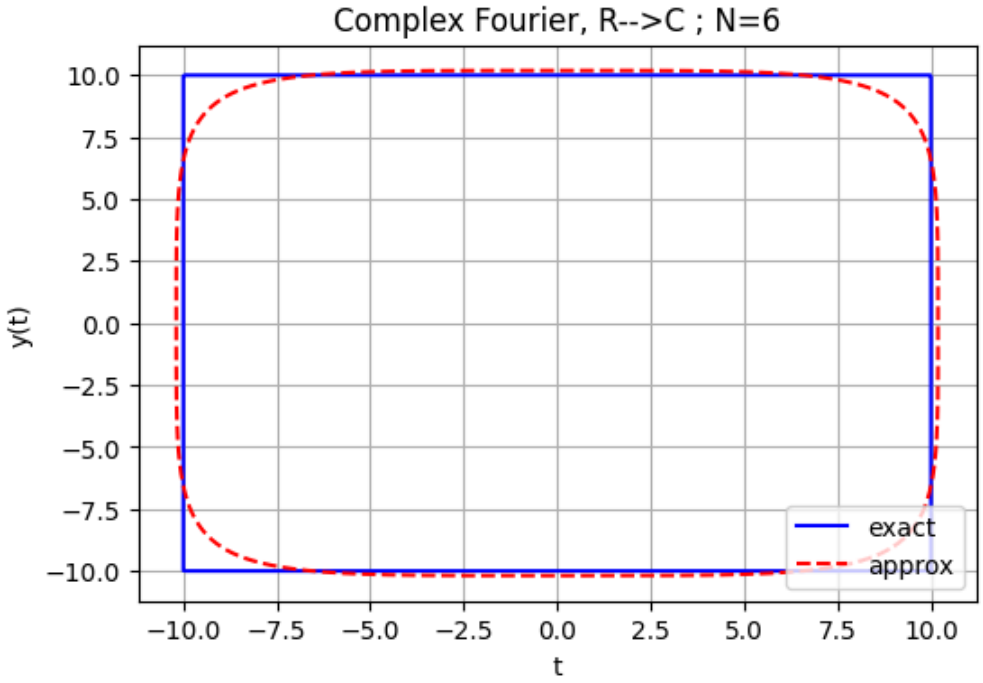
In [ ]:
```
# возьмём в качестве координаты X все возможные значения на периоде
X = np.linspace(-T/8, 7*T/8, 1000)
plot_task2_original(ff,X)
```

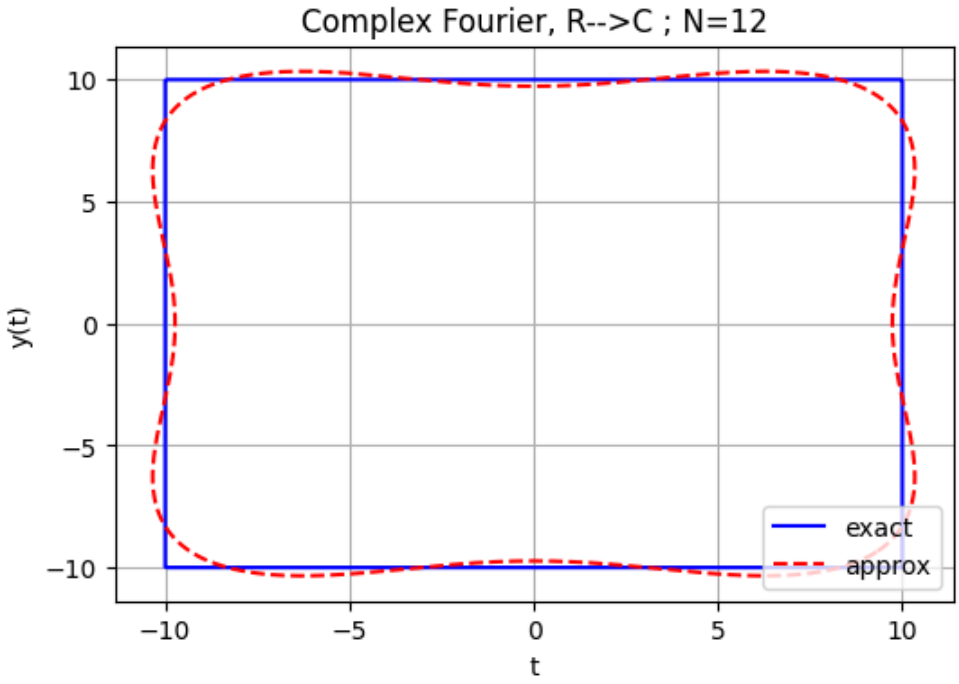


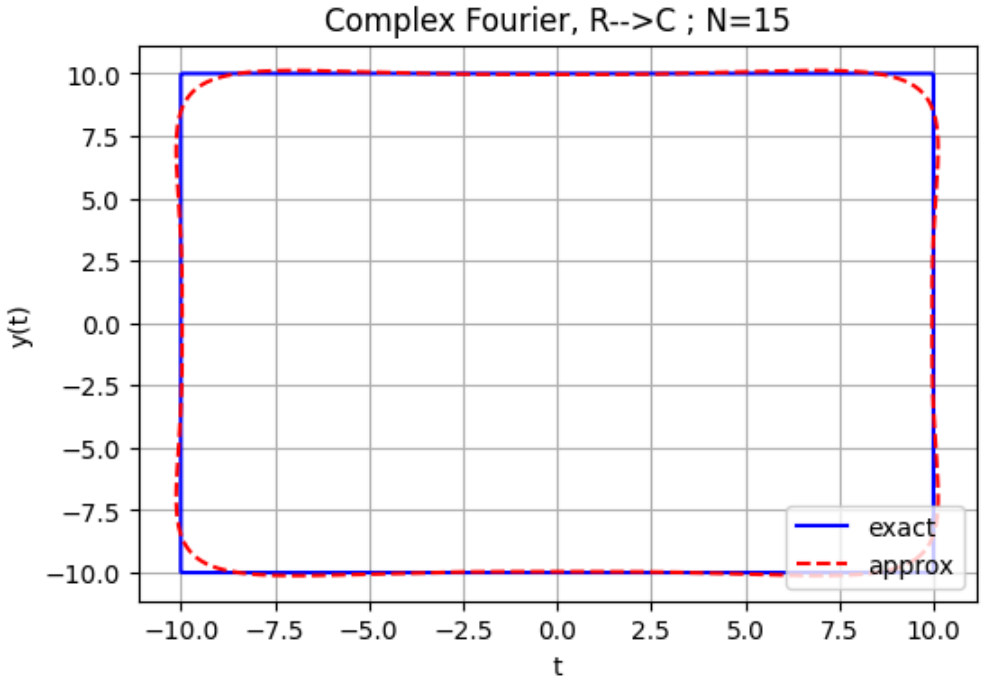
Посмотрим на частичные суммы Фурье для этого графика, $G_N(t) = \sum_{n=-N}^{N} c_n e^{i\omega_n t}$, где $\omega_n = 2\pi n/T$

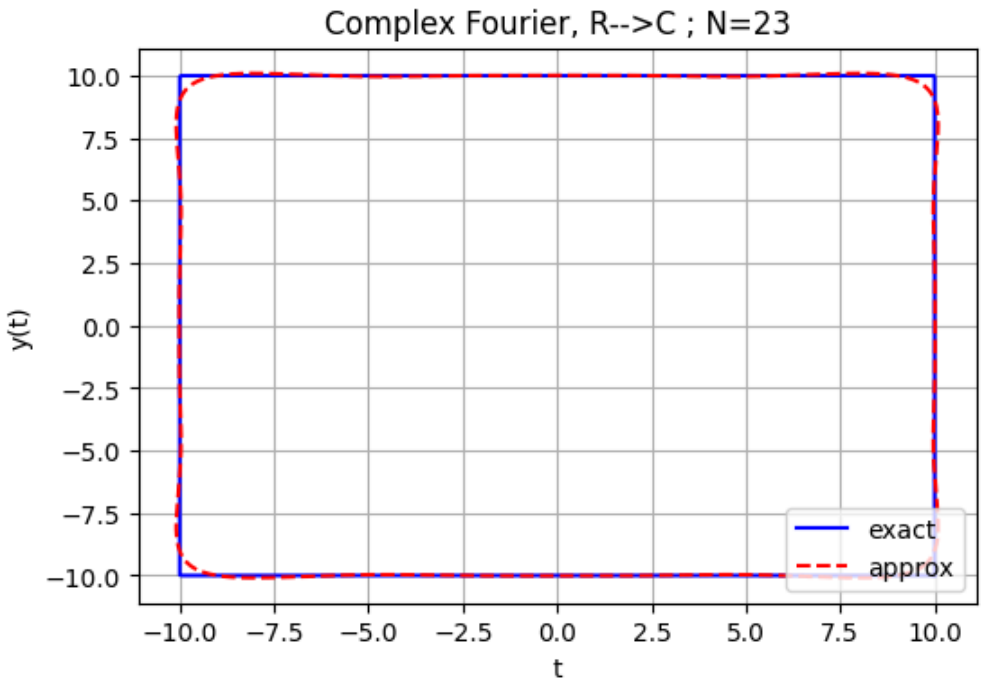
In [ ]:
```
plot_complex_plane_fourier(ff,X,[3,6,12,15,23,],-T, 2*T)
```


Complex Fourier, R-->C ; N=3
y(t)
t
exact
approx
10
5
0
−5
−10
−10
−5
0
5
10


Complex Fourier, R-->C ; N=6
y(t)
t
exact
approx
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Complex Fourier, R-->C ; N=12
10
5
0
−5
−10
y(t)
−10
−5
0
5
10
t
exact
approx


Complex Fourier, R-->C ; N=15
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
y(t)
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
t
exact
approx



Вычислим для $n = \{0, 1, 2\}$ вручную;

На всех кусках функция состоит из линейной и константной части, которая чередутся в зависимости от t, это немного упростит задачу вычисления "руками". $f(t)$ - кусочек функции

$$c_n = \frac{1}{T}\int_{n}^{n+T}\left[f(t)e^{-i\omega_n t}\right]dt$$

$$c_n = \frac{1}{3}\int\left[f(t)e^{-i\frac{2\pi t}{3}}\right]dt = \frac{1}{3}\left[\int_{15/8}^{21/8}(-60+80t/3-10i)e^{-i\frac{2\pi t}{3}}dt + \int_{9/8}^{15/8}(-10+i(40-80t/3))e^{-i\frac{2\pi t}{3}}dt + \int_{3/8}^{9/8}(20-80t/3+10i)e^{-i\frac{2\pi t}{3}}dt + \int_{-3/8}^{3/8}(10+i80t/3)e^{-i\frac{2\pi t}{3}}dt\right] = \int_{3/8}^{15/8}(-20/3+7t-6i)e^{-i\frac{2\pi t}{3}}dt$$

$$c_{-3} \approx -0.109$$

$$c_{-2} \approx 0.181$$

$$c_{-1} \approx -0.278$$

$$c_0 \approx 0.451$$

$$c_1 \approx -0.921$$

$$c_2 \approx 11.246$$

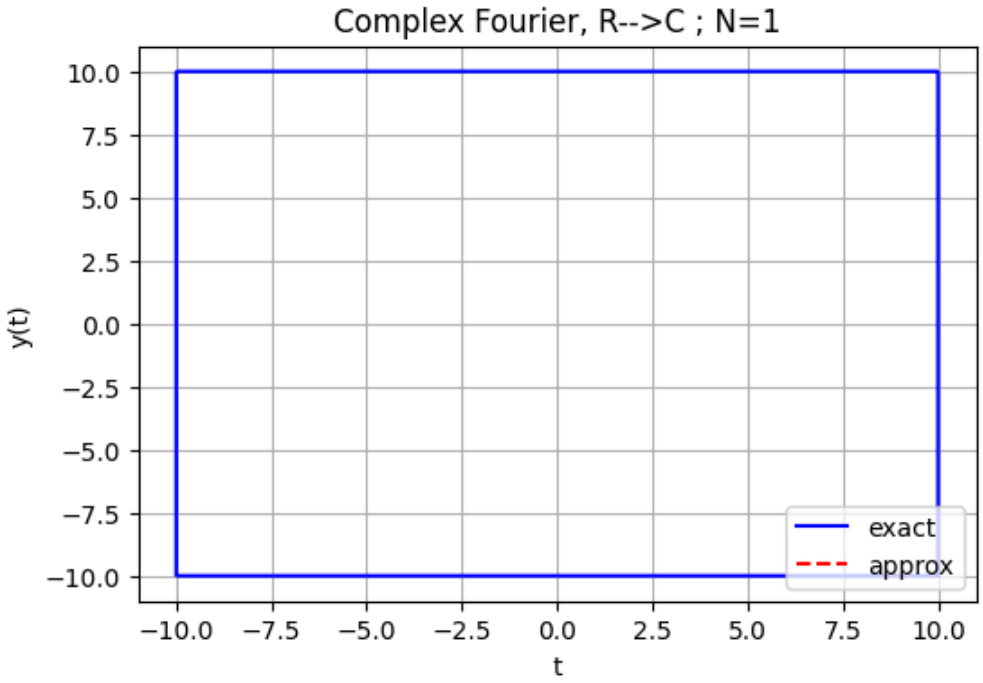
$$c_3 \approx 1.225$$

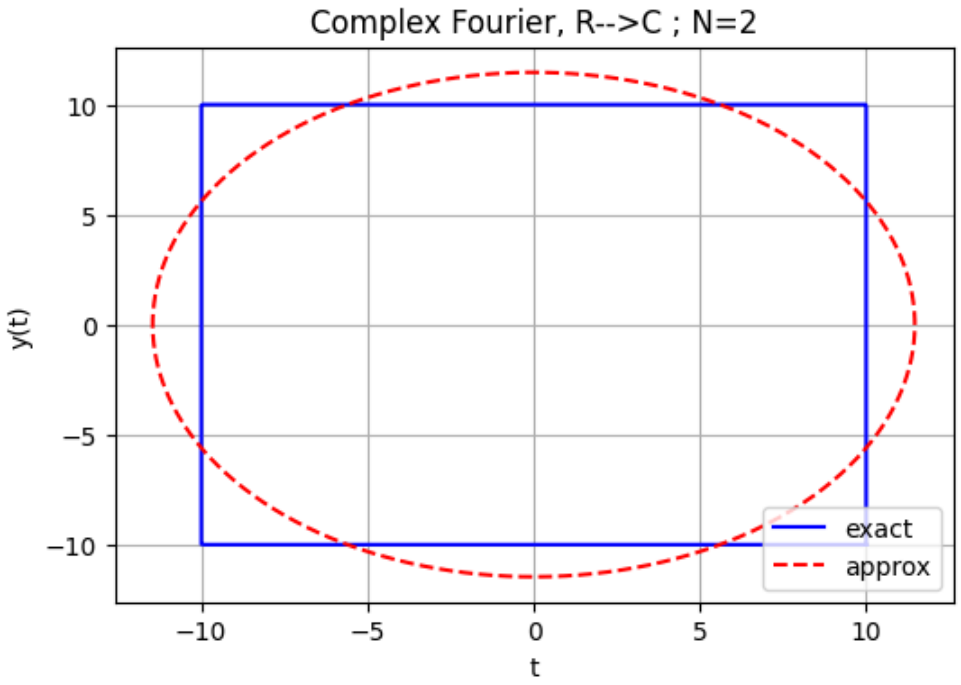
Сравним посчитанные коэффициенты вручную с программой, $N = 3$, и они хорошо сходятся

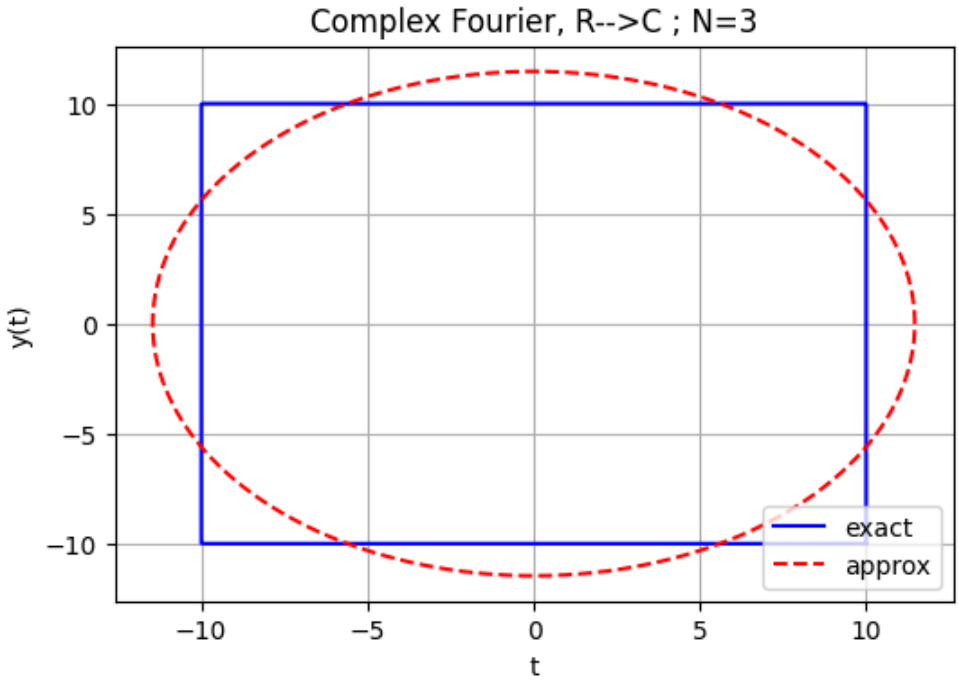
In [ ]:
```
printBeauty(compute_complex_fourier_coeffs(ff, 3,-np.pi, 2*np.pi))
```

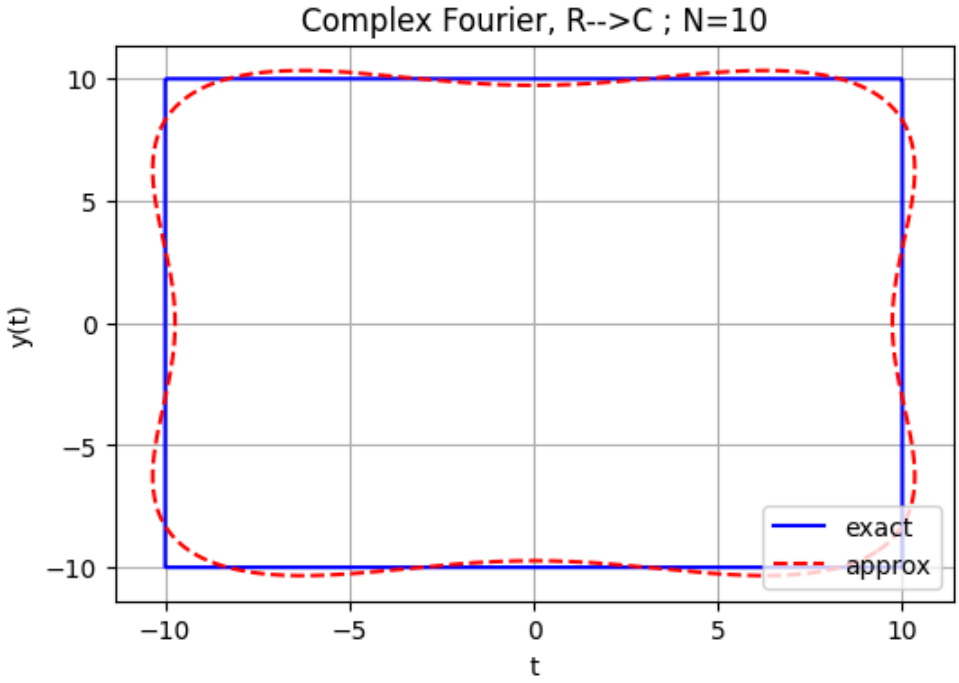
```
-0.1091-0.0000j
 0.1814-0.0000j
-0.2781+0.0000j
 0.4507+0.0000j
-0.9211-0.0000j
11.2460+0.0000j
 1.2257-0.0000j
```

In [ ]:
```
plot_complex_plane_fourier(ff,X,[1,2,3,10], -T, 2*T)
```





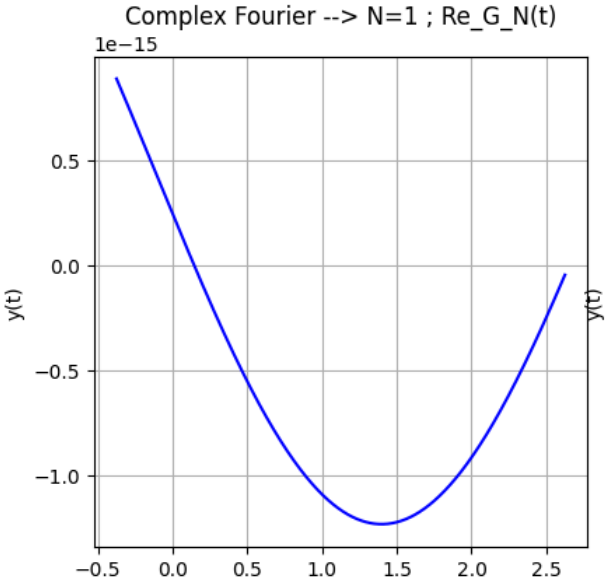


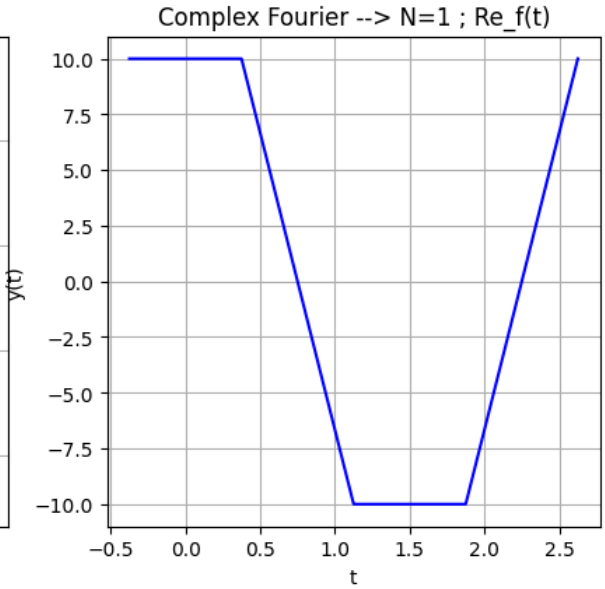


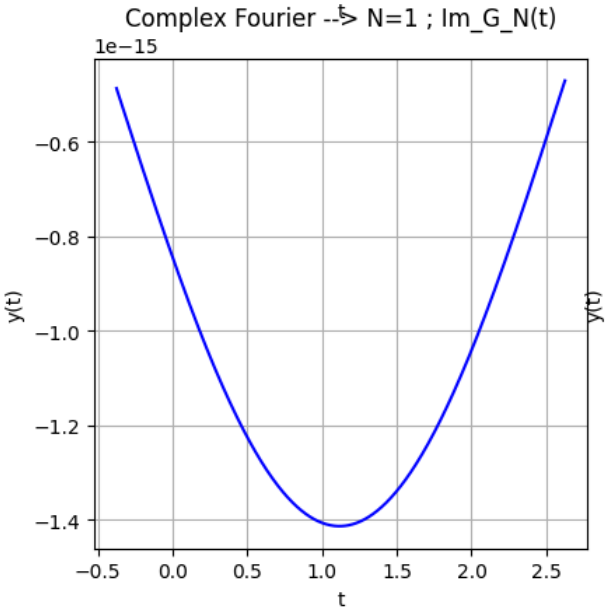
Приближаемся гармониками, как и раньше, чётными и нечётными, но теперь они стали круговыми, вау!

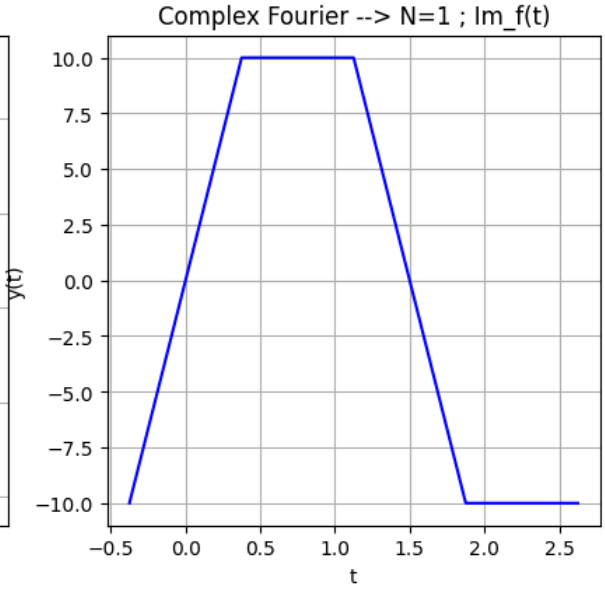
Построим следующие графики в следующем порядке: $Ref(t)$, $Imf(t)$, $ReG_N(t)$, $ImG_N(t)$ для $N = \Big\{1, 2, 3, 10\Big\}$

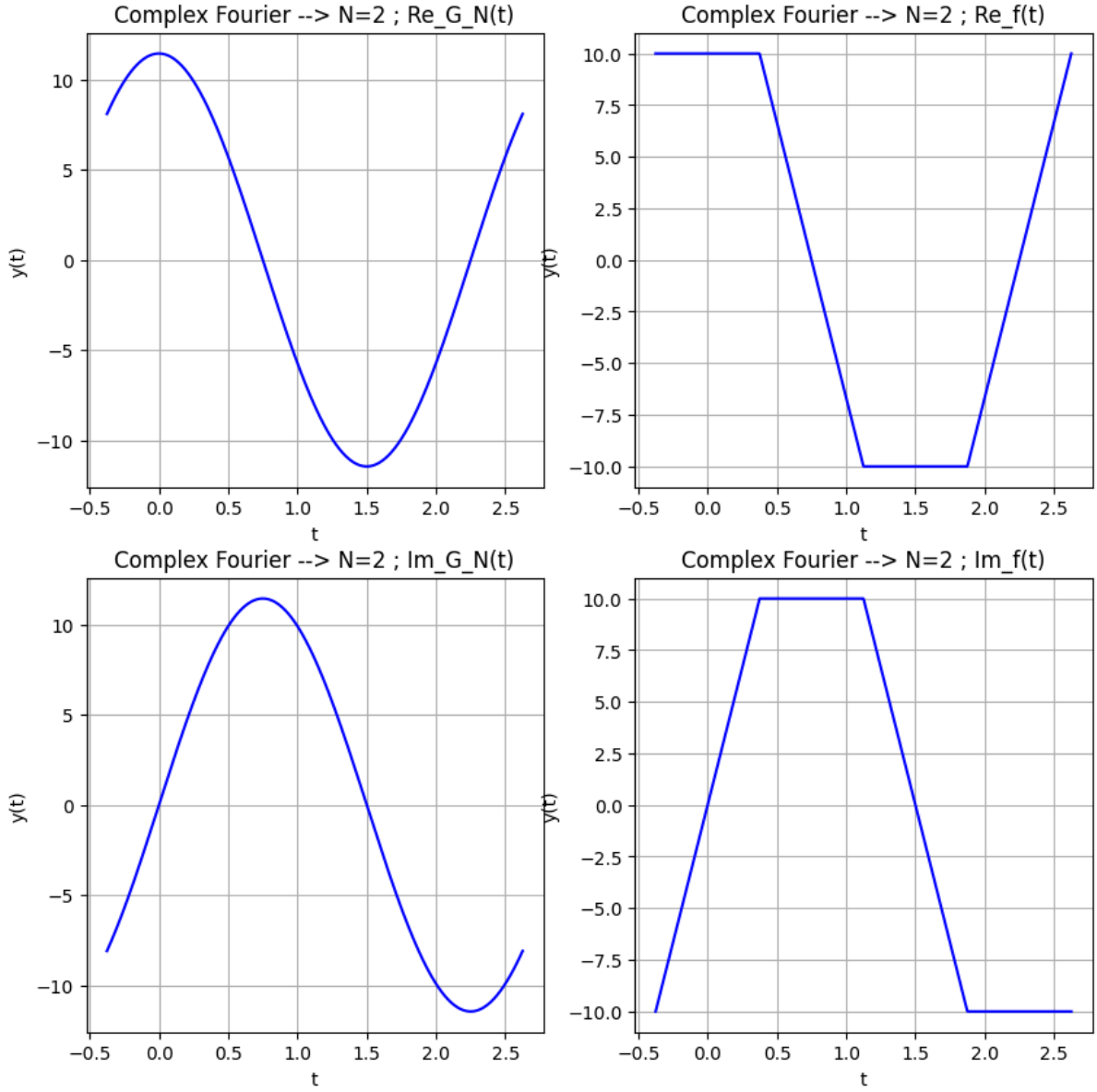
In [ ]:
```
plot_complex_plane_fourier_part(ff, X, [1,2,3,10], -T,2*T)
```

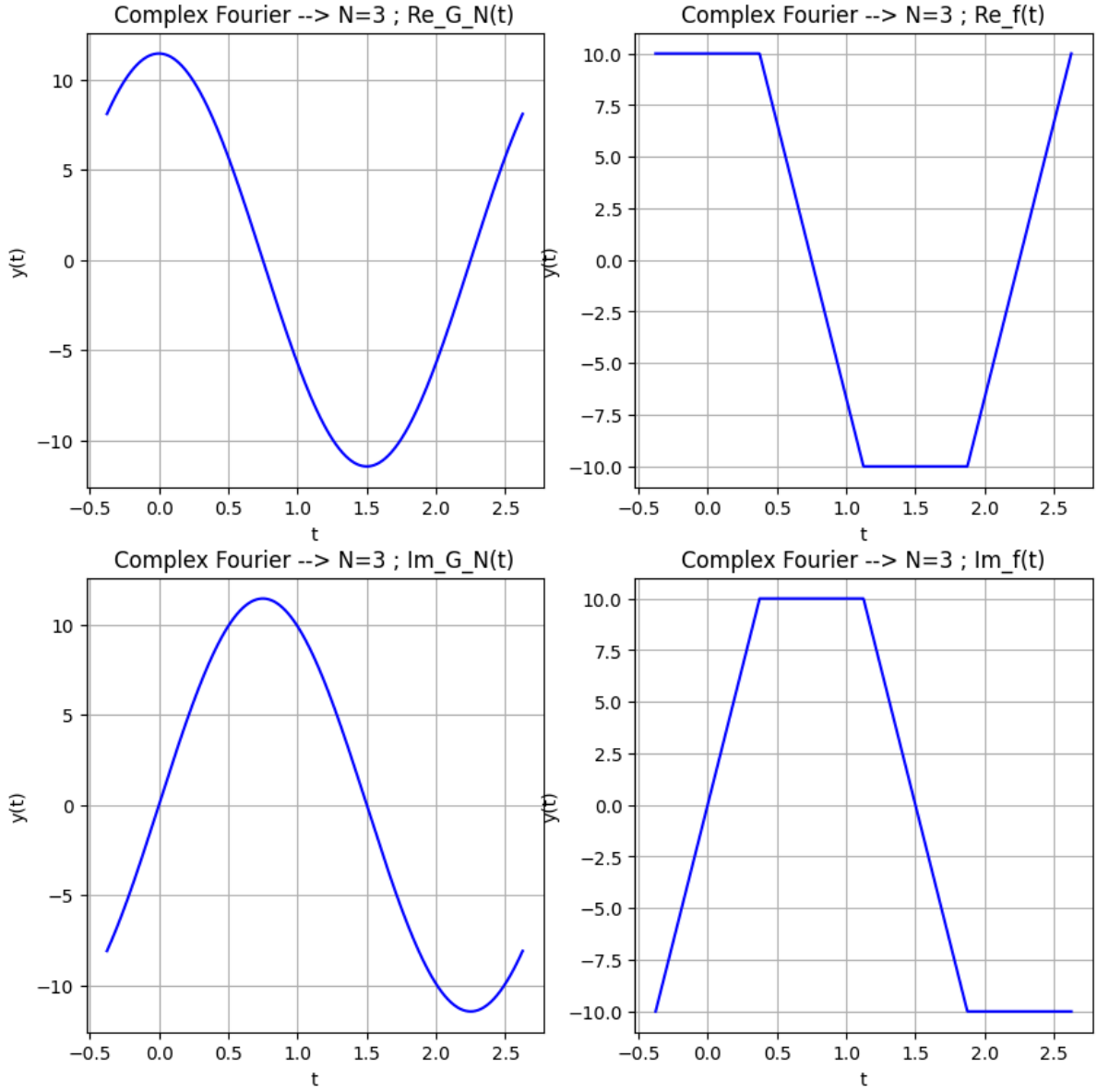

Complex Fourier --> N=1 ; Re_G_N(t)
1e−15
0.5
0.0
−0.5
−1.0
y(t)
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
t

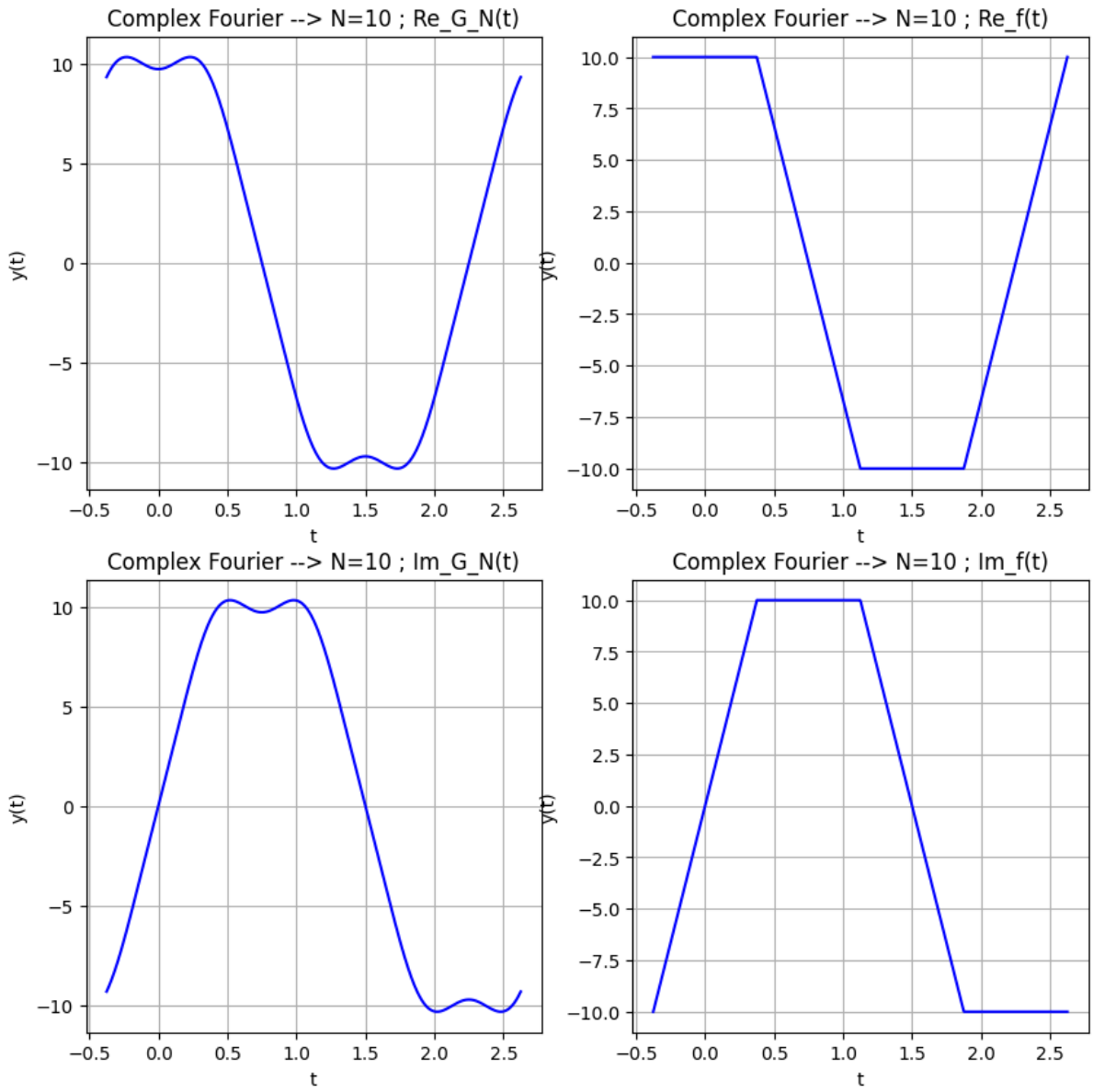

Complex Fourier --> N=1 ; Re_f(t)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
y(t)
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
t


Complex Fourier --> N=1 ; Im_G_N(t)
1e−15
−0.6
−0.8
−1.0
−1.2
−1.4
y(t)
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
t


Complex Fourier --> N=1 ; Im_f(t)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
y(t)
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
t


Complex Fourier --> N=2 ; Re_G_N(t)
Complex Fourier --> N=2 ; Re_f(t)
Complex Fourier --> N=2 ; Im_G_N(t)
Complex Fourier --> N=2 ; Im_f(t)
y(t)
t


Complex Fourier --> N=3 ; Re_G_N(t)
Complex Fourier --> N=3 ; Re_f(t)
Complex Fourier --> N=3 ; Im_G_N(t)
Complex Fourier --> N=3 ; Im_f(t)
y(t)
t



Проверим равенство парсеваля, в данном случае будем руководствоваться первой формулой, но придётся чуть поменять численное интегрирование под $\mathbb{C}$ случай

In [ ]:
```python
def check_parseval_complex(func, x, h, T, N):
    # norm of the function
    INT = complex_quad(lambda t: abs(func(t))**2, -np.pi, np.pi)
    # squared coeffs
    COEFFS =  2*np.pi * compute_complex_fourier_coeffs_sum(func, N, h, T)
    print(f"N = {N} --> coeff {COEFFS:.5f} and int {INT:.5f}")
    return abs(INT - COEFFS)


T = 2*np.pi
h = -np.pi
for n in [10,20,50,100,200]:
    print(f"delta: {check_parseval_complex(ff, t, h, T,n):.5f}")
```

```
N = 10 --> coeff 837.62310 and int 837.75804+0.00000j
delta: 0.13495
N = 20 --> coeff 837.74093 and int 837.75804+0.00000j
delta: 0.01712
N = 50 --> coeff 837.75694 and int 837.75804+0.00000j
delta: 0.00110
N = 100 --> coeff 837.75790 and int 837.75804+0.00000j
delta: 0.00014
N = 200 --> coeff 841.14698 and int 837.75804+0.00000j
delta: 3.38894
```

N	Delta	Норма	Сумма коэффициентов
50	0.00110	837.75804	837.75694
100	0.00014	837.75804	837.75790
200	3.38894	837.75804	841.14698

Как можно заметить, при лучшем коэффициенте нам удаётся добится равенства до третьего знака

Задание 3. Произвольный рисунок

Этот таймкод из видео 3Blue1Brown даёт нам понять, что для начала нужно нарисовать в каком-нибудь редакторе замкнутую, допускающую самопересечения, кривую, а после найти способ сохранить её в хорошем .svg формате. Внутри такого файла будет xml разметка и можно с помощью питона спарсить её в path.

А что будет парсится то? и что такое Path? Эти вопросы уже больше относятся к формату **S**calable **V**ector **G**raphics, но если отвечать кратко, то внутри такого файла зашита обычная xml разметка, один из блоков которой содержит в себе информацию о примитивах, из которых "собирается" изображение. В нашем случае у нас такой частный случай изображения, что будут использоваться только кривые или линейные перемещения, поэтому файл можно спарсить в одну продолжительную кривую

Способов создать такое изображение, чтобы с него можно было спарсить - много, но я лично не хотел устанавливать новые программы, поэтому приведу основные шаги как это всё сделать в фотошопе.

1. Находим картинку, а после обводим её инструментом Pen в произвольном формате, главное, чтобы это была одна Фигура.
2. Делаем эту фигуру "смарт-объектом", после заходим внутрь неё и уже там пытаемся сделать экспорт проекта
3. Если у нас новая версия фотошопа, то возможно не будет формата SVG как выбора, тогда следуем этой инструкции и возвращаем себе её, экспортируем уже окончательное

Вдохновитель инструкции

N.B - я не нашёл конвертеров $.PNG \rightarrow .SVG$, которые понимают линии как сверхтонкие и, как следствие ,не делают двойную кривую, поэтому если не понимаете что такое "сверхтонкий абрис в coreldraw толщина в мм", то лучше просто сделайте всё в фотошопе, а не ищите конвертер

Также выяснил, что спарсить кривые, по которым строится исходное изображение в .svg можно с помощью питоновской библиотеки *svg.path*, однако с ней дальнейшая дружба у меня не

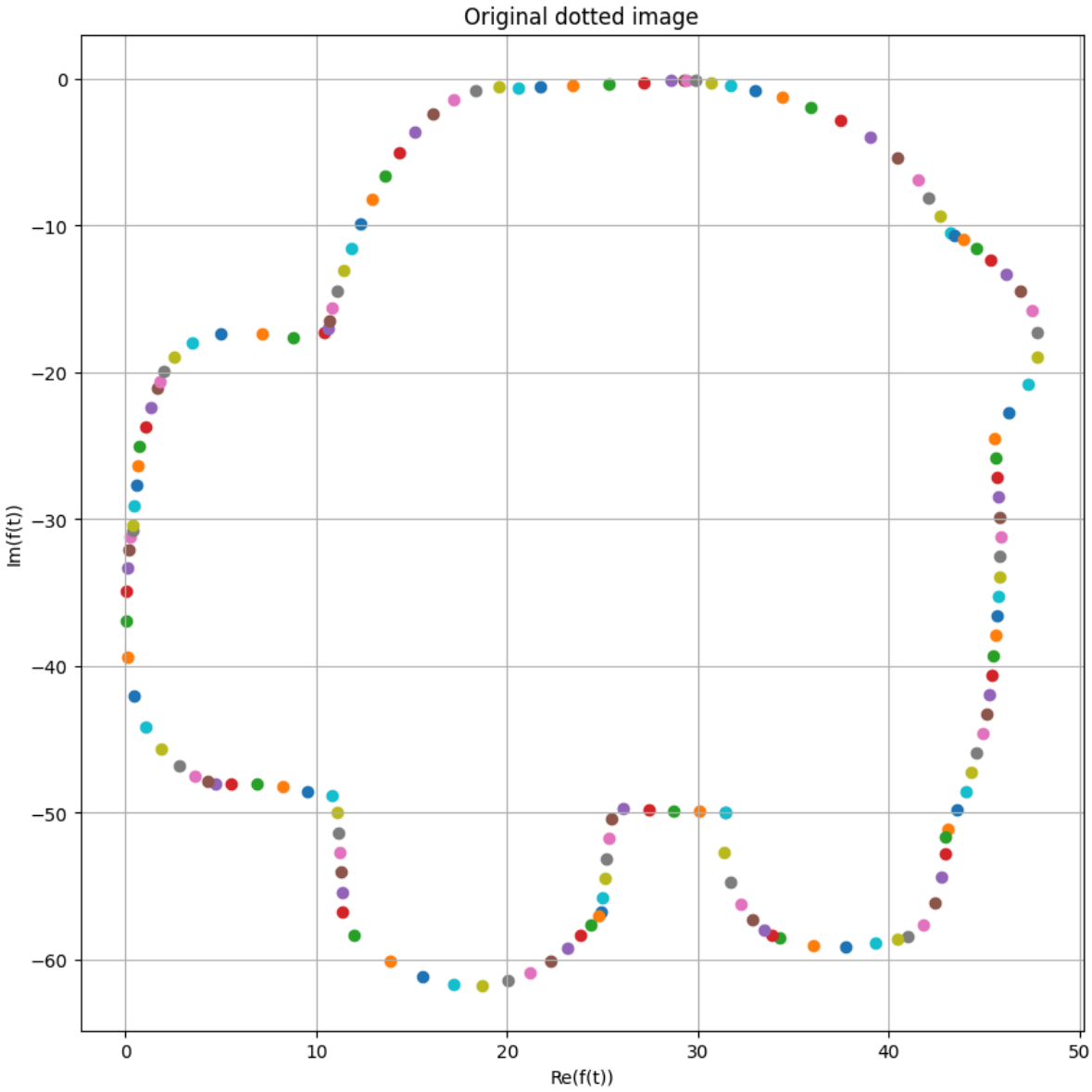
сложилась и я воспользовался модулем *svgpathtools*

Теперь добавим немного смысловой нагрузки, ведь любое изображение из одной кривой можно представить в виде большой, но конечной(в смысле количества кусочков) кусочно-заданной функции, что сейчас мы и сделаем с помощью метода vectorize

In [ ]:
```
# файл можно найти в исходном коде
paths, attributes = svg2paths("Fig.svg")
path = paths[0]
# Create a lambda function for the path
path_func = np.vectorize(lambda t: path.point(t) * (1/16))
```

Убедимся, что мы смогли корректно импортировать кривые с файла, и наша функция действительно работаем сделаем график из точек на равномерном расстоянии друг от друга, так как пространство оси X здесь нормировано $[0; 1]$, то его можно задать удобным образом. В качестве примера возьмём 150 точек

In [ ]:
```
x = np.linspace(0,1,150)

def plot_scatter(path):
  plt.figure(figsize=(10, 10))
  for t in x:
    path = path_func(t)
    plt.scatter(path.real, -path.imag)
  plt.xlabel('Re(f(t))')
  plt.ylabel('Im(f(t))')
  plt.title("Original dotted image")
  plt.grid()
  plt.show()

plot_scatter(path_func)
```



Очевидным становится, что каждая точка - просто комплексное число, а значит для такой функции можно применить идею из второго задания - ряд Фурье для комплексной функции...

Получим коэффициенты для частичное суммы при $N = 3$

Отдельное внимание хочу уделить тому, что коэффициенты ищутся на промежутке интеграла от 0 до 1, до конца не выяснил почему, но возможно особенность импортированной функции, главное, что работает :)

In [ ]:
```
printBeauty(compute_complex_fourier_coeffs(path_func, 3, 0, 1))
```

```
 0.0800- 0.8339j
 1.3141+ 0.1137j
-1.0340+ 2.7876j
24.7979+32.6910j
 8.1614+24.5065j
-1.7982- 2.2044j
-0.1388- 1.3203j
```

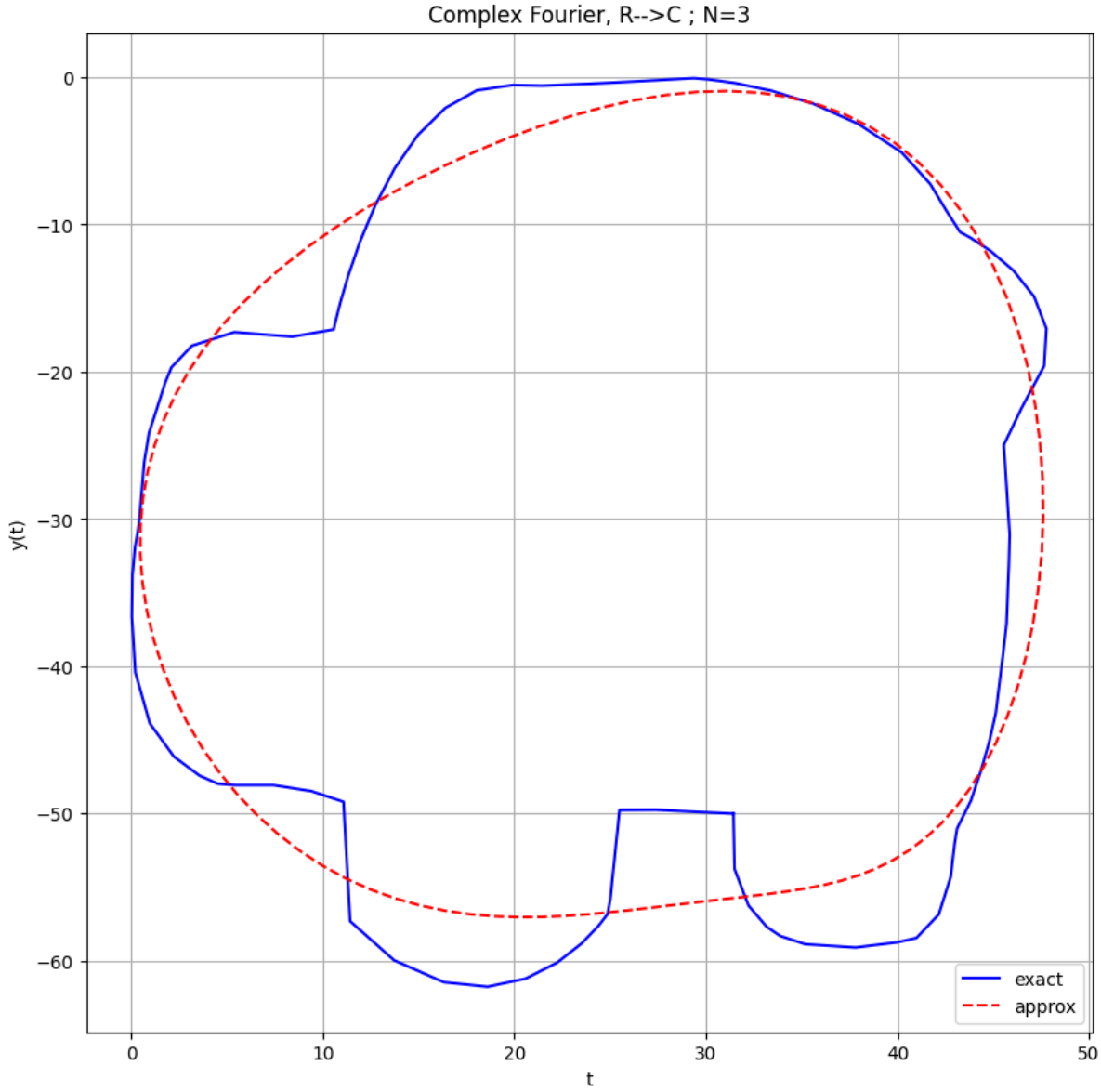
Также придётся переопределить функции вычисления коэффициентов, потому что иначе график становится перевёрнутым по оси Y, вторая, так называемая особенность импорта... (просто два минуса добавить)

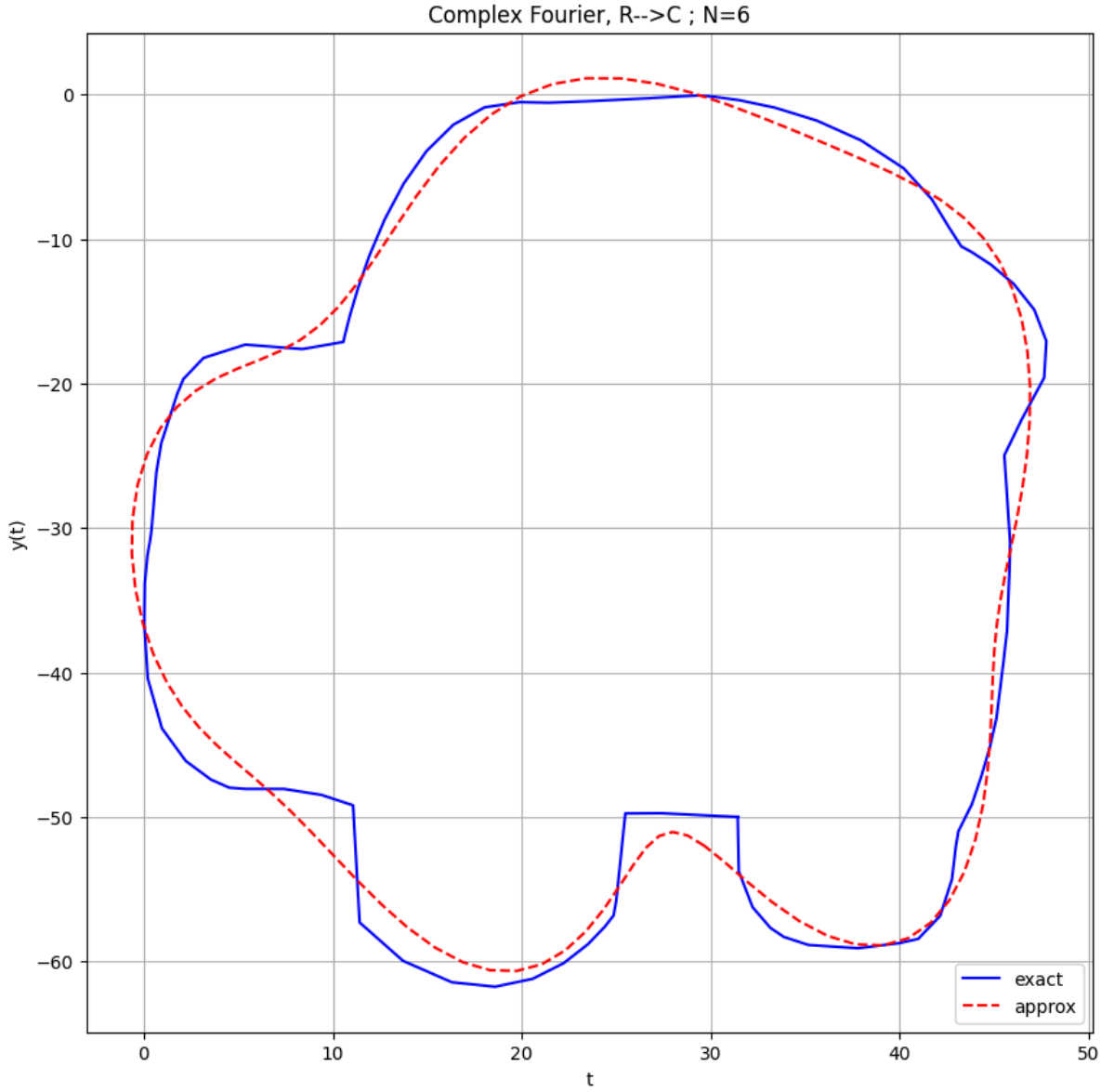
In [ ]:
```
def compute_complex_plane_fourier_coeffs(func, N, h,T):
    result = []
    for n in range(-N, N+1):
        cn = (1./T) * complex_quad(lambda t: func(t) * np.exp(-1j * 2 * np.pi * n * t / T),
        result.append(cn)
    return np.array(result)

def fit_func_by_fourier_series_with_complex_plane_coeffs(t, C, T):
    result = 0. + 0.j
    L = int((len(C) - 1) / 2)
    for n in range(-L, L+1):
        c = C[n+L]
        result +=  c * np.exp(1j * 2. * np.pi * n * t / T)
    return result

def G_N_complex(f, x,N, h,T):
    C = compute_complex_plane_fourier_coeffs(f, N, h,T)
    y_approx = fit_func_by_fourier_series_with_complex_plane_coeffs(x, C, T)

    return [[y.real for y in y_approx], [-y.imag for y in y_approx]]
```

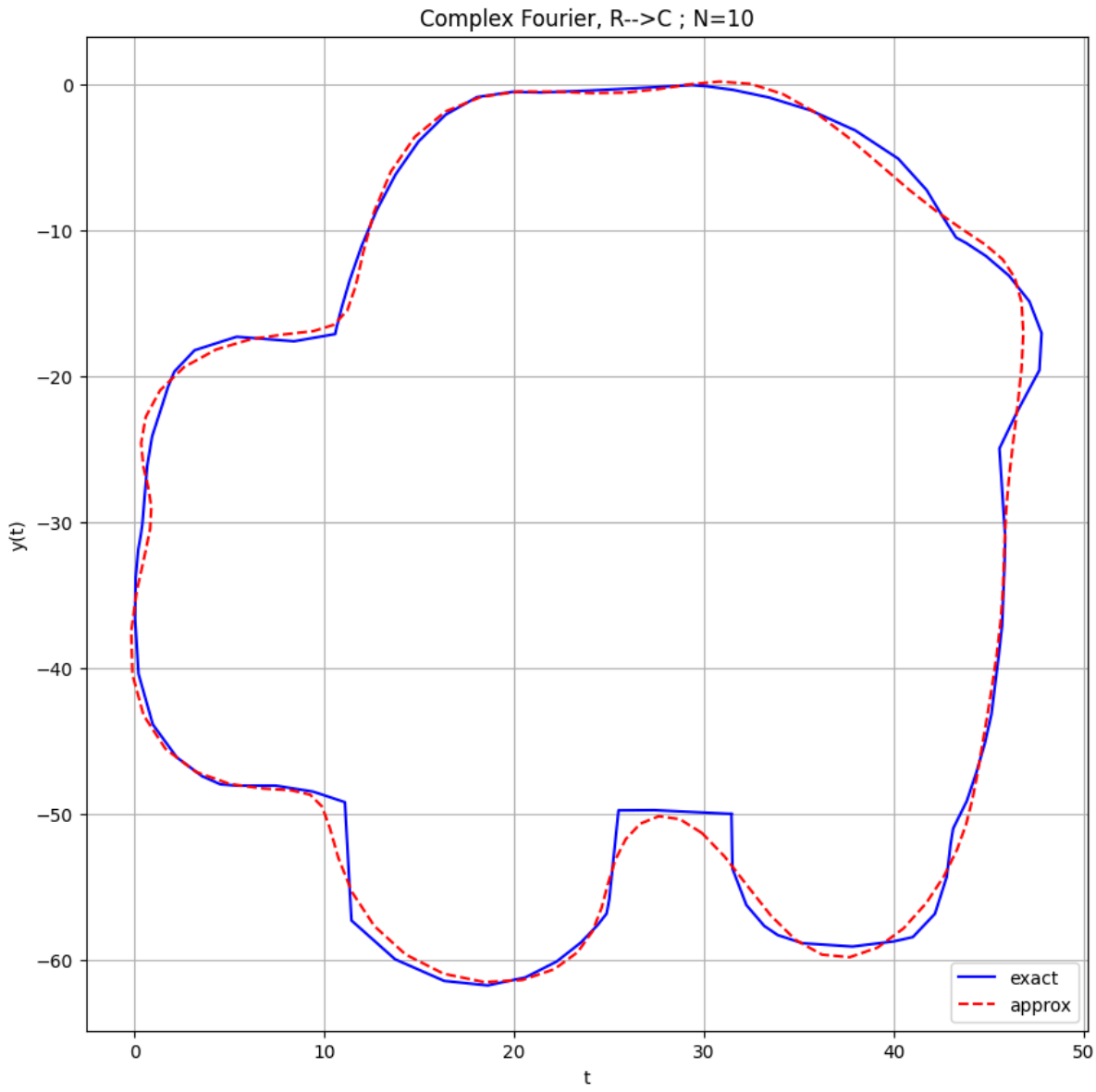
In [ ]:
```
def plot_complex_plane_fourier(func, x, N, h, T):
    for i in N:
        G_N = G_N_complex(func, x, i, h,T)
        plt.figure(figsize = (10, 10))
        plt.plot(func(x).real, -func(x).imag, label='exact',color='blue')
        plt.plot(G_N[0], G_N[1], color='red', linestyle='dashed',label='approx')
        plt.title('Complex Fourier, R-->C ; ' + f"N={i}")
        plt.xlabel('t')
        plt.ylabel('y(t)')
        plt.legend(loc='lower right')
        plt.grid()
        plt.show()
```

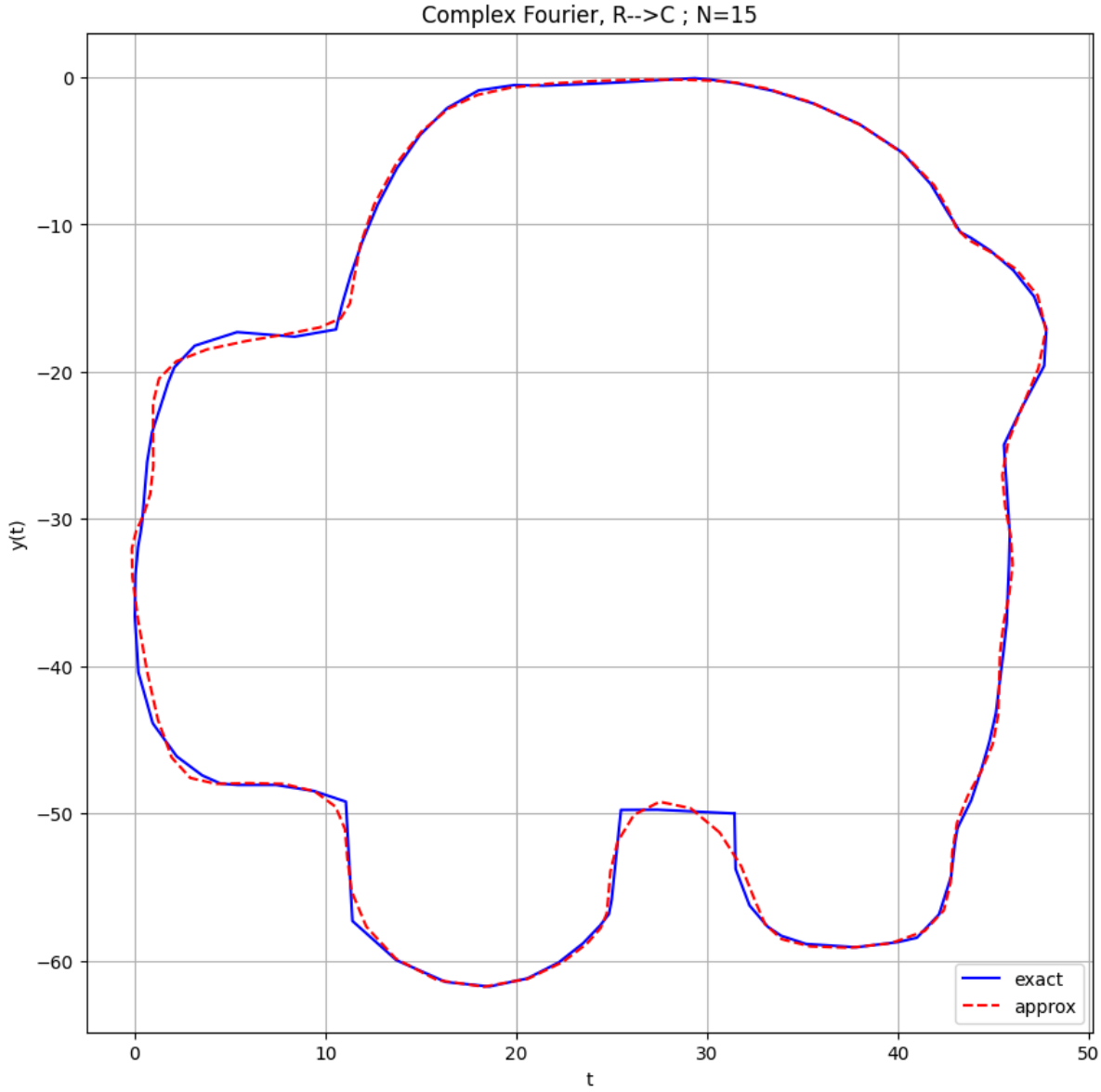
Построим некоторые частичные суммы.... да, график действительно сходится!

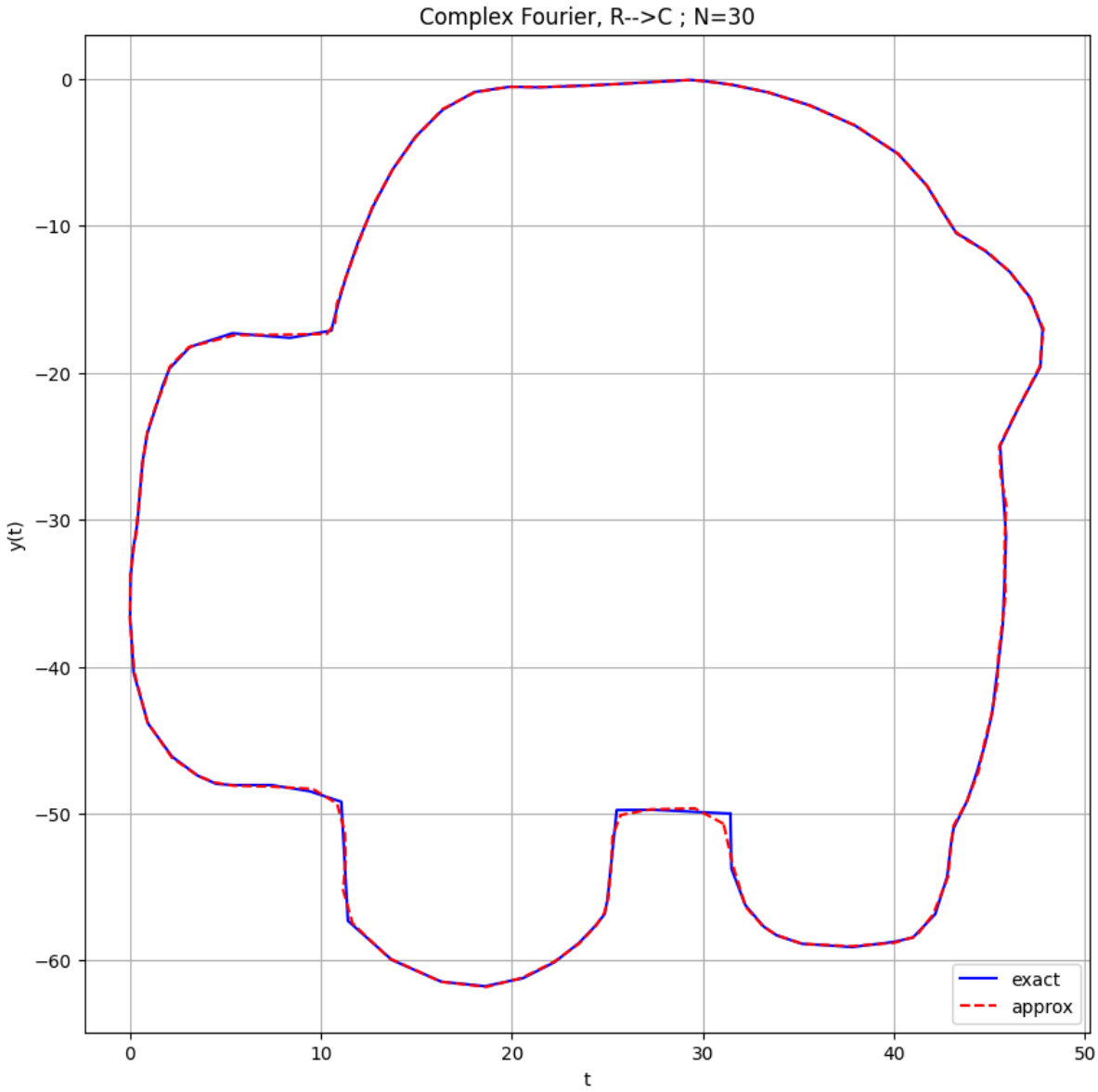
In [ ]:
```
x = np.linspace(0,1,100)
plot_complex_plane_fourier(path_func,x,[3,6, 10, 15,30],0, 1)
```


Complex Fourier, R-->C ; N=3
0
−10
−20
−30
−40
−50
−60
y(t)
0
10
20
30
40
50
t
exact
approx


Complex Fourier, R-->C ; N=6
0
−10
−20
−30
−40
−50
−60
y(t)
0
10
20
30
40
50
t
exact
approx


Complex Fourier, R-->C ; N=10
0
−10
−20
−30
−40
−50
−60
y(t)
0
10
20
30
40
50
t
exact
approx


Complex Fourier, R-->C ; N=15
0
−10
−20
−30
−40
−50
−60
y(t)
0
10
20
30
40
50
t
exact
approx



Сегодня я не нашёл в себе силы победить Manim и сделать красивую анимацию, но уж в следующий раз....

Полезные ссылки

Здесь решил оставить некоторую часть всего добра, что помогало мне на протяжении лабы, по возможности к некоторым оставил свои комментарии

1. Просто крутой интерактивный сайт
2. По поводу визуализации рядов Фурье
3. Много про разные виды Фурье, в том числе многомерные, с примерами кода вычисления через scipy
4. Ряды Фурье в блендере
5. Ряды Фурье DFT отрисовка
6. Ряды Фурье и 3B1B - про идею появления
7. Ряды Фурье и 3B1B - про рисование