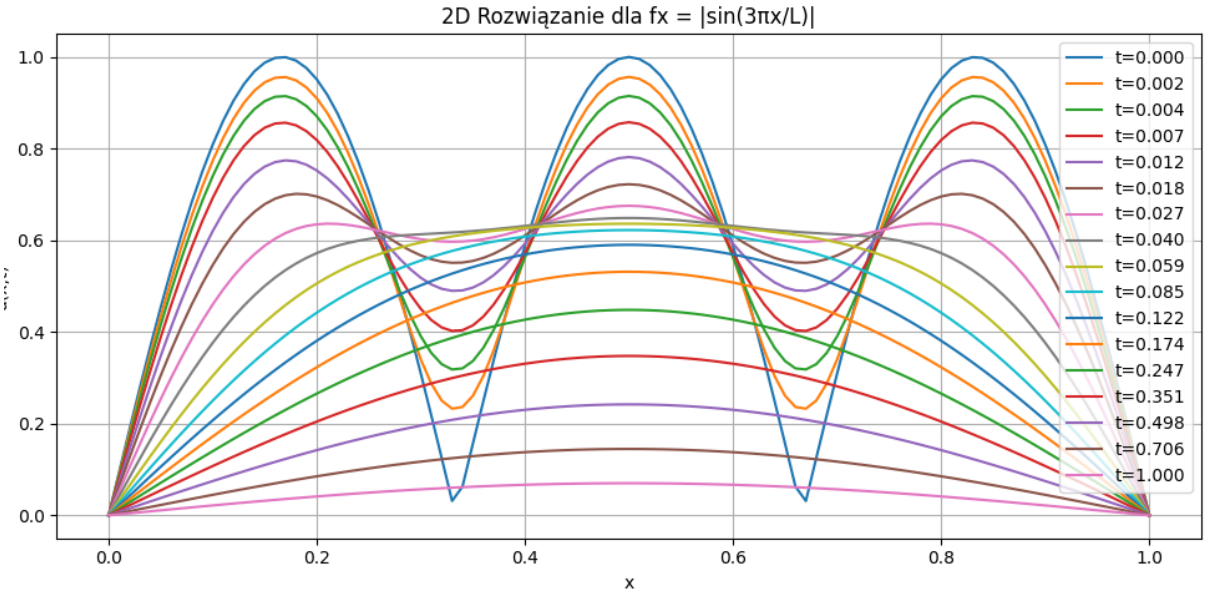
Lab 3 – Równania różniczkowe cząstkowe.

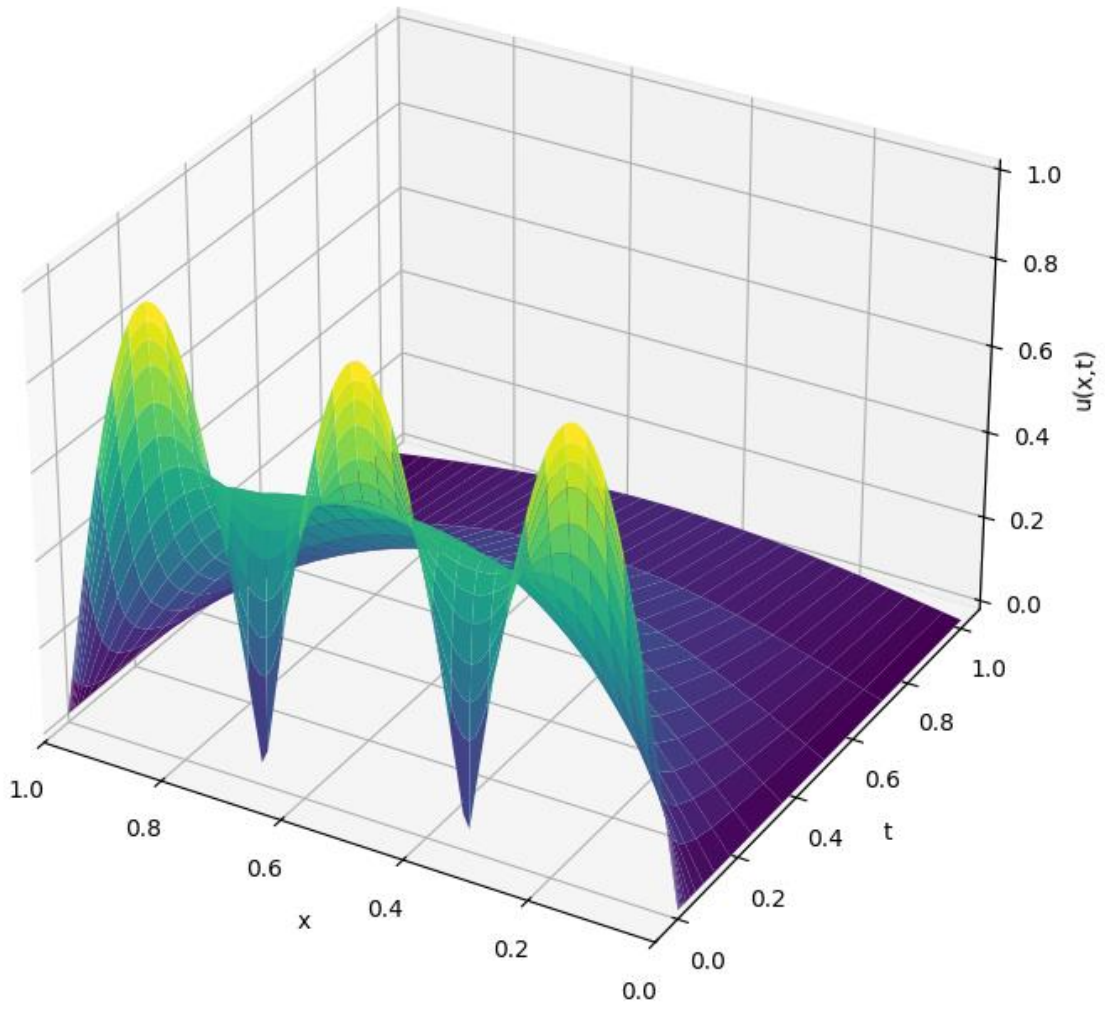
Arkadiusz Kurnik, Jan Cichoń

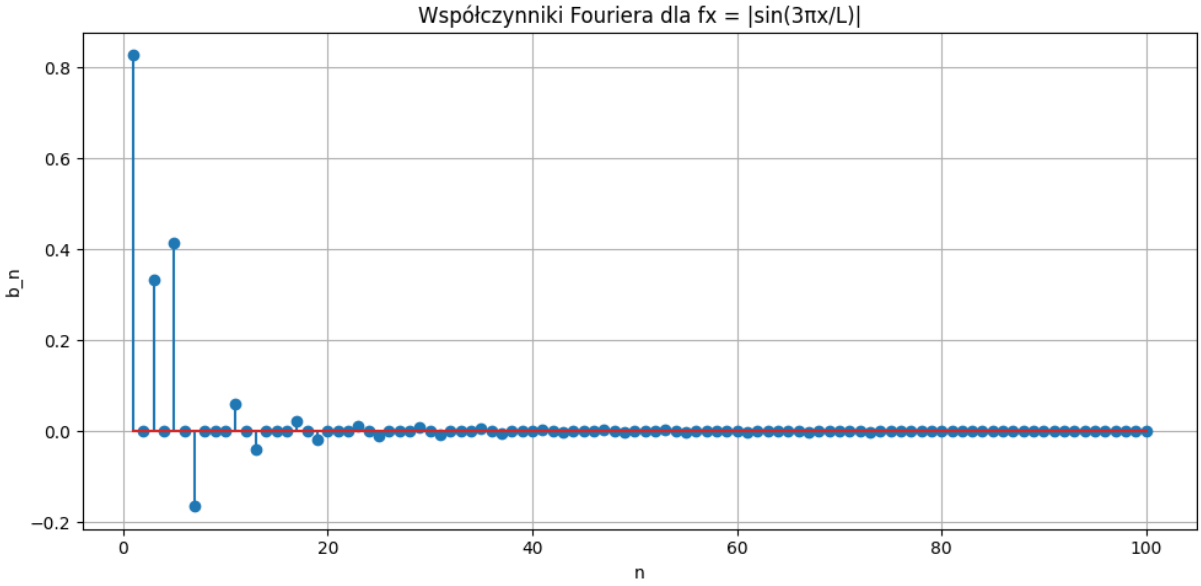
Zadanie 1:

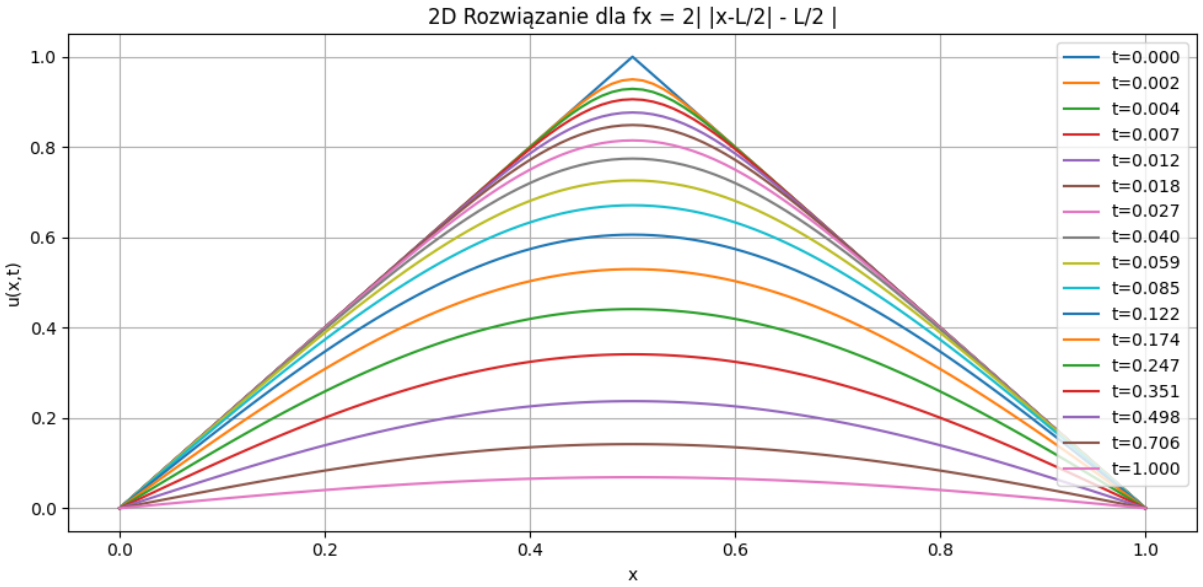
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.integrate import trapezoid

#ZAD1

# Definicja parametrów
L = 1.0
D = 0.25
N = 100
x = np.arange(0, L+0.01, 0.01)
t_values = [0.000, 0.002, 0.004, 0.007, 0.012, 0.018, 0.027, 0.040, 0.059, 0.085, 0.122, 0.174, 0.247, 0.351, 0.498, 0.706, 1.000]

def f1(x):
    return np.abs(np.sin(3 * np.pi * x / L))

def f2(x):
    return 2 * np.abs(np.abs(x - L/2) - L/2)

def f3(x):
    return np.where((x >= 0.4) & (x <= 0.6), 1, 0)

def compute_bn(n, f):
    return (2 / L) * trapezoid(f(x) * np.sin(n * np.pi * x / L), x)

def u_xt(x, t, f):
    sum_series = np.zeros_like(x)
    for n in range(1, N+1):
        bn = compute_bn(n, f)
        sum_series += bn * np.sin(n * np.pi * x / L) * np.exp(-n**2 * np.pi**2 * D * t / L**2)
    return sum_series

def plot_results(f, title):
    # Wykres 2D
    fig, ax = plt.subplots(figsize=(10, 5))
    for t in t_values:
        ax.plot(x, u_xt(x, t, f), label=f't={t:.3f}')
    ax.set_xlabel('x')
    ax.set_ylabel('u(x,t)')
    ax.set_title(f'2D Rozwiązanie {title}')
    ax.legend()
    ax.grid()
    plt.tight_layout()
    plt.show()
```

```python
    # Wykres 3D
    fig = plt.figure(figsize=(10, 7))
    ax = fig.add_subplot(111, projection='3d')
    X, T = np.meshgrid(x, t_values)
    U = np.array([u_xt(x, t, f) for t in t_values])
    ax.plot_surface(X, T, U, cmap='viridis')
    ax.set_xlabel('x')
    ax.set_ylabel('t')
    ax.set_zlabel('u(x,t)')
    ax.set_title(f'3D Rozwiązanie {title}')
    ax.set_xlim(np.max(x), np.min(x))
    plt.tight_layout()
    plt.show()

    # Wykres współczynników Fouriera
    fig, ax = plt.subplots(figsize=(10, 5))
    n_values = np.arange(1, N+1)
    bn_values = np.array([compute_bn(n, f) for n in n_values])
    ax.stem(n_values, bn_values)
    ax.set_xlabel('n')
    ax.set_ylabel('b_n')
    ax.set_title(f'Współczynniki Fouriera {title}')
    ax.grid()
    plt.tight_layout()
    plt.show()

# Generowanie wykresów
plot_results(f1, 'dla fx = |sin(3πx/L)|')
plot_results(f2, 'dla fx = 2| |x-L/2| - L/2 |')
plot_results(f3, 'dla fx = Prostokąt (0.4 ≤ x ≤ 0.6)')
```

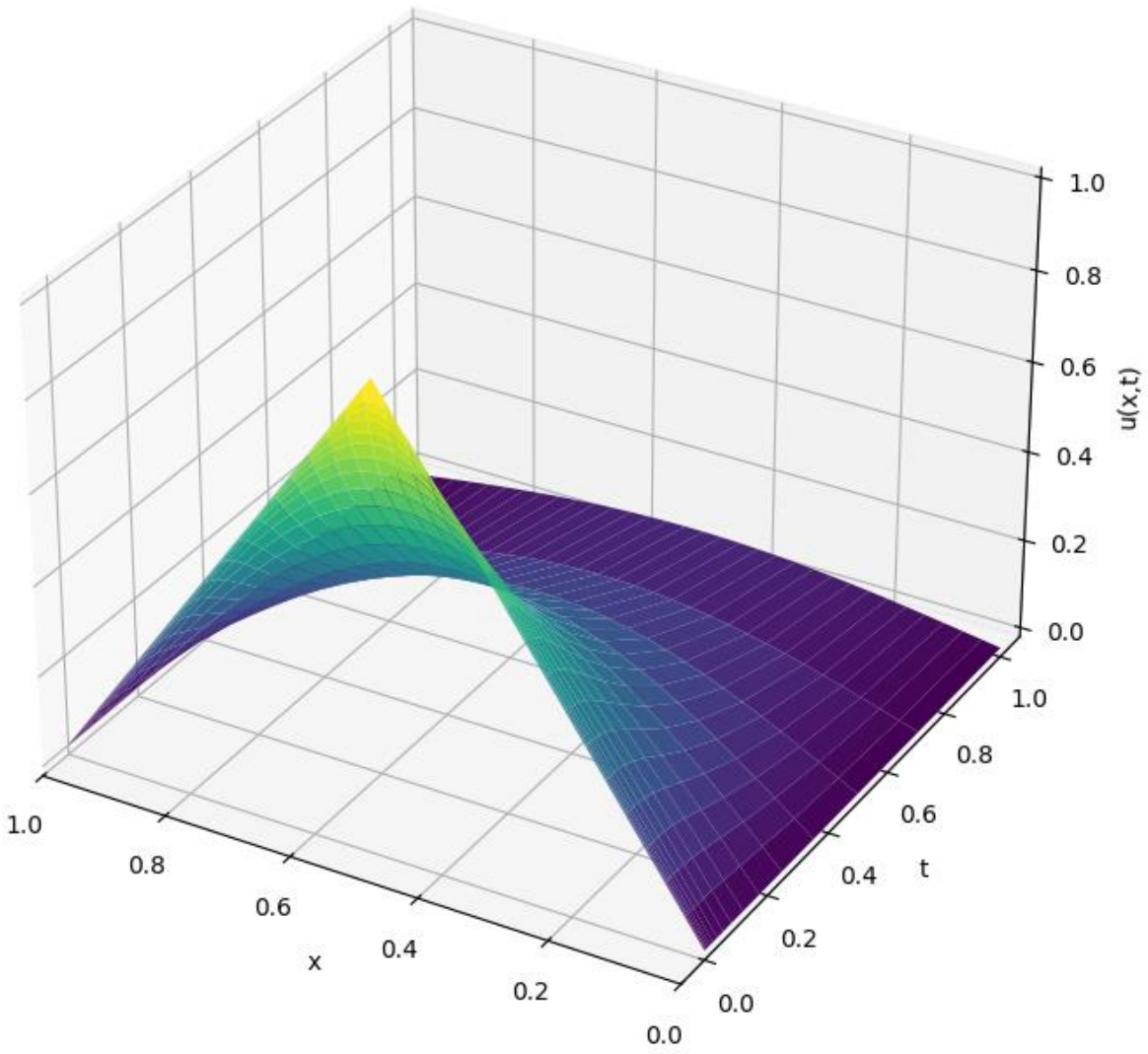


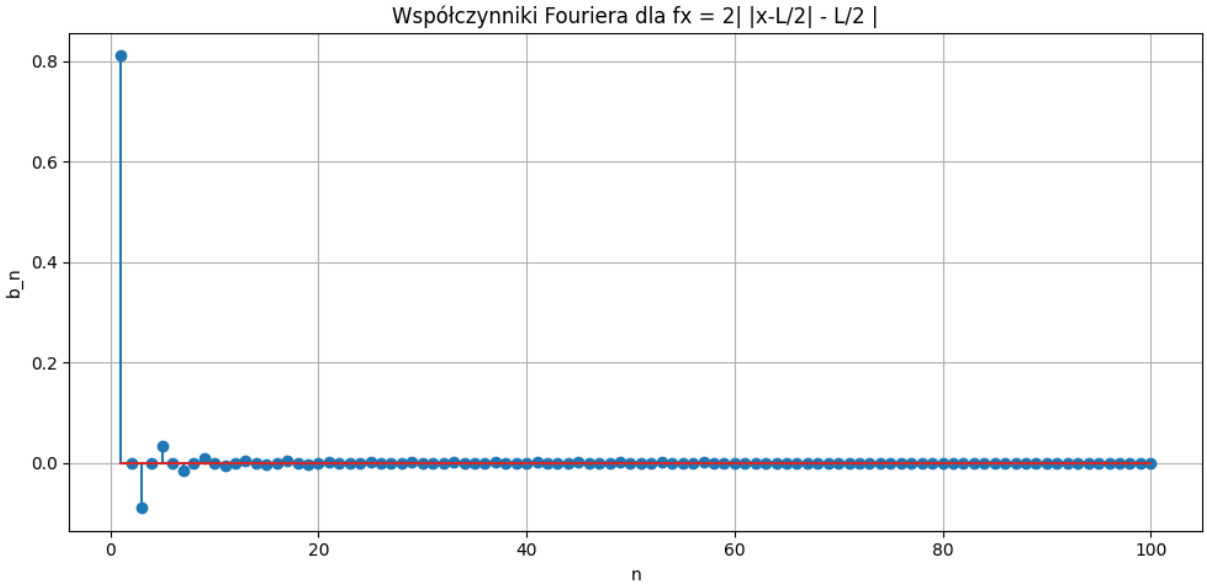

3D Rozwiązanie dla fx = |sin(3πx/L)|
u(x,t)
x
t

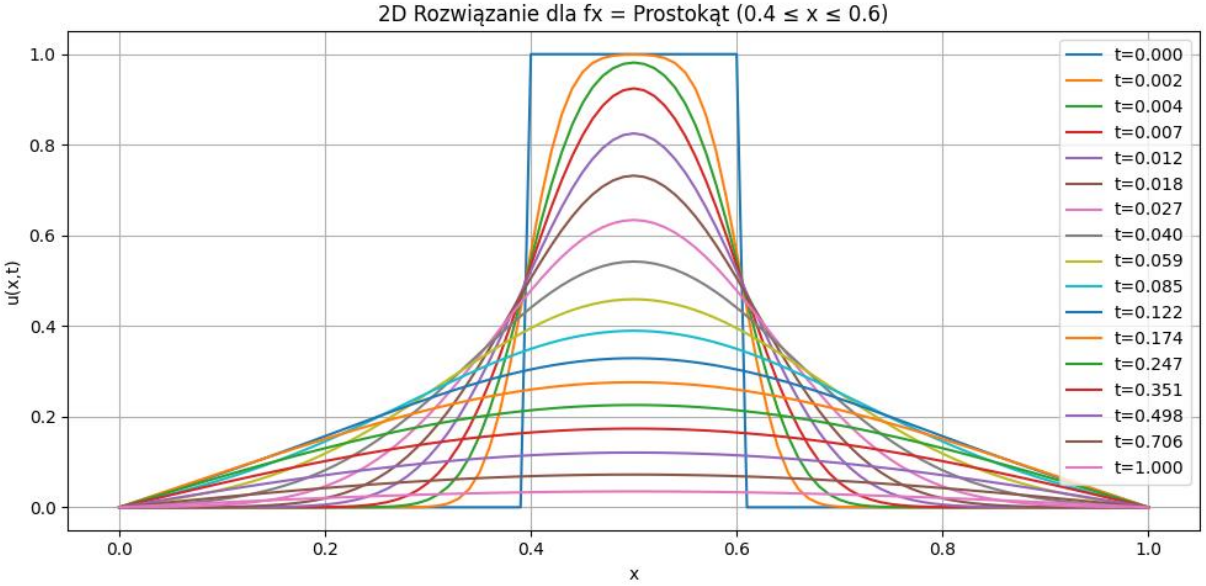

Współczynniki Fouriera dla fx = |sin(3πx/L)|
b_n
n

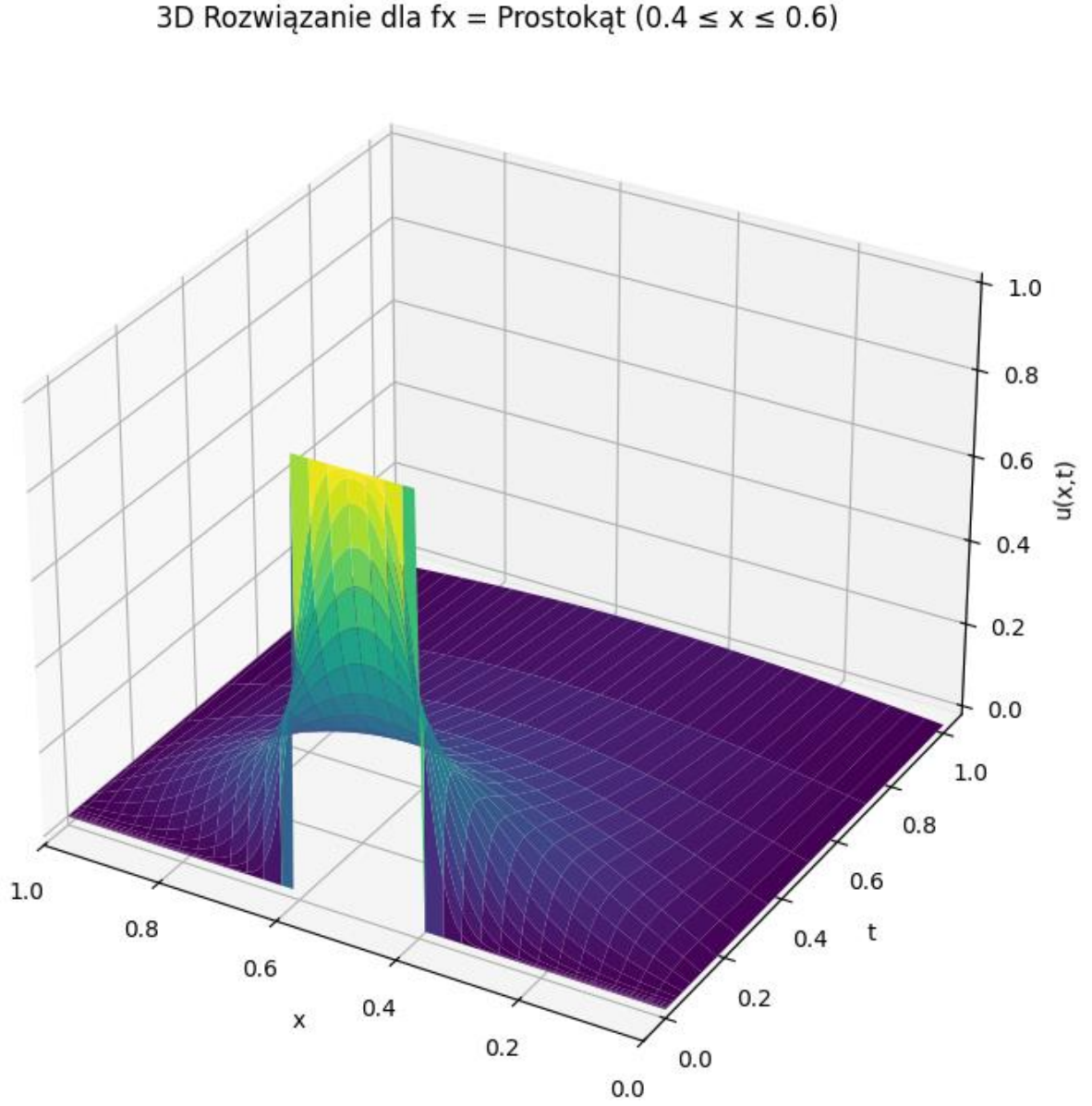


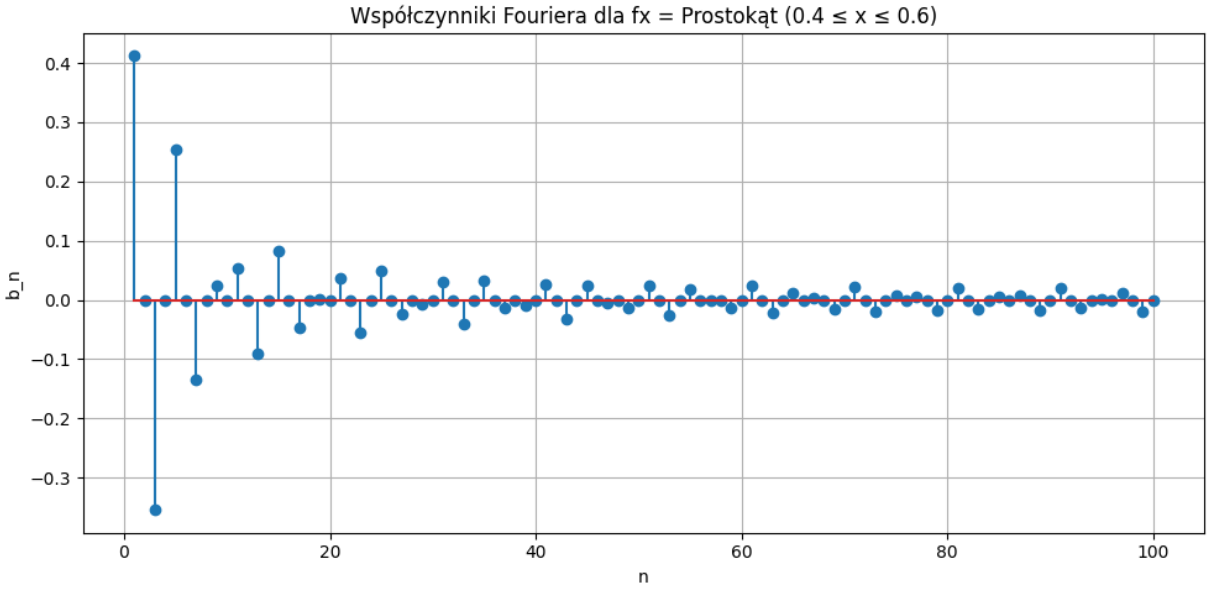
3D Rozwiązanie dla fx = 2| |x-L/2| - L/2 |




Współczynniki Fouriera dla fx = 2| |x-L/2| - L/2 |
b_n
n
0.0
0.2
0.4
0.6
0.8
0
20
40
60
80
100


2D Rozwiązanie dla fx = Prostokąt (0.4 ≤ x ≤ 0.6)
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000

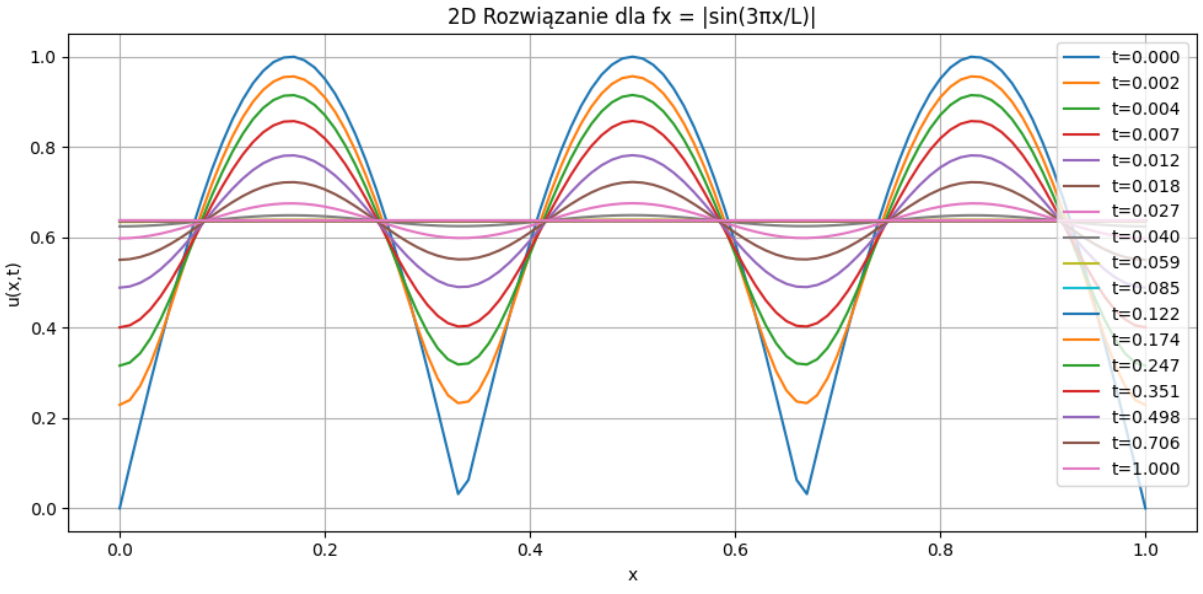

3D Rozwiązanie dla fx = Prostokąt (0.4 ≤ x ≤ 0.6)
u(x,t)
x
t
1.0
0.8
0.6
0.4
0.2
0.0

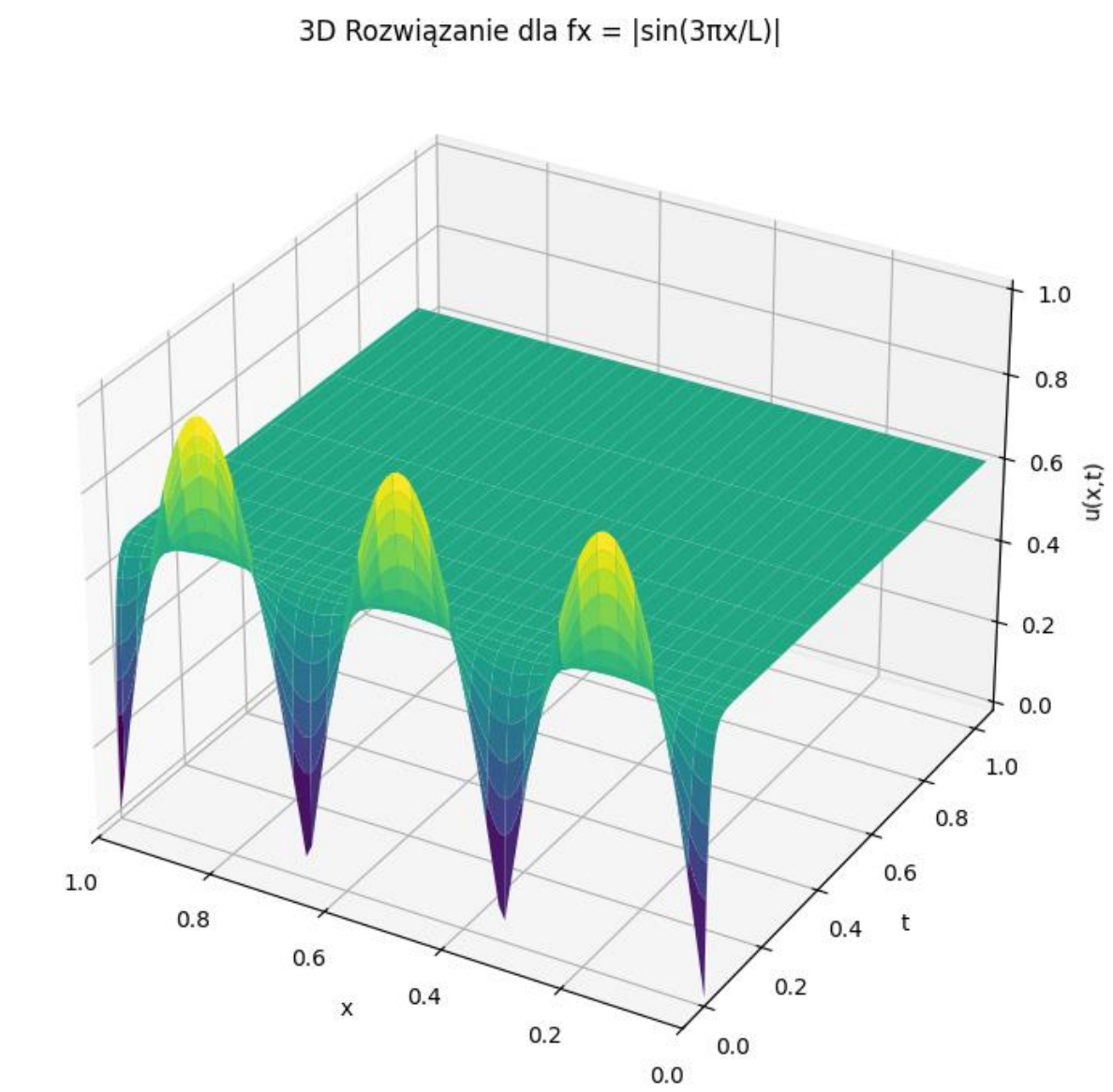

Współczynniki Fouriera dla fx = Prostokąt (0.4 ≤ x ≤ 0.6)
b_n
n
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
20
40
60
80
100

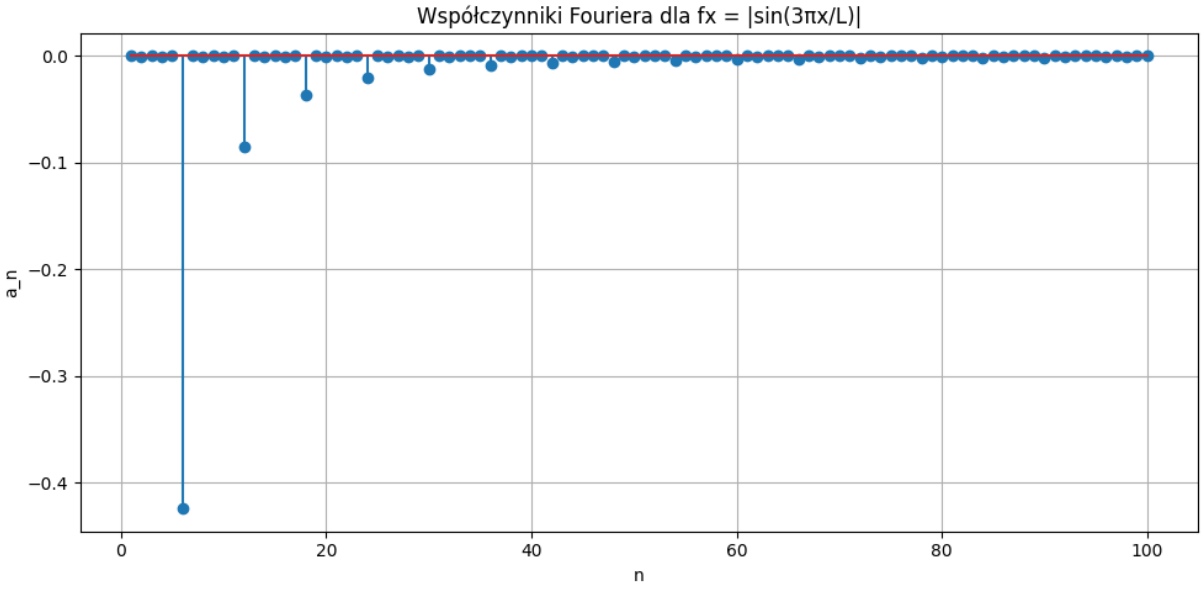
Zadanie 2:

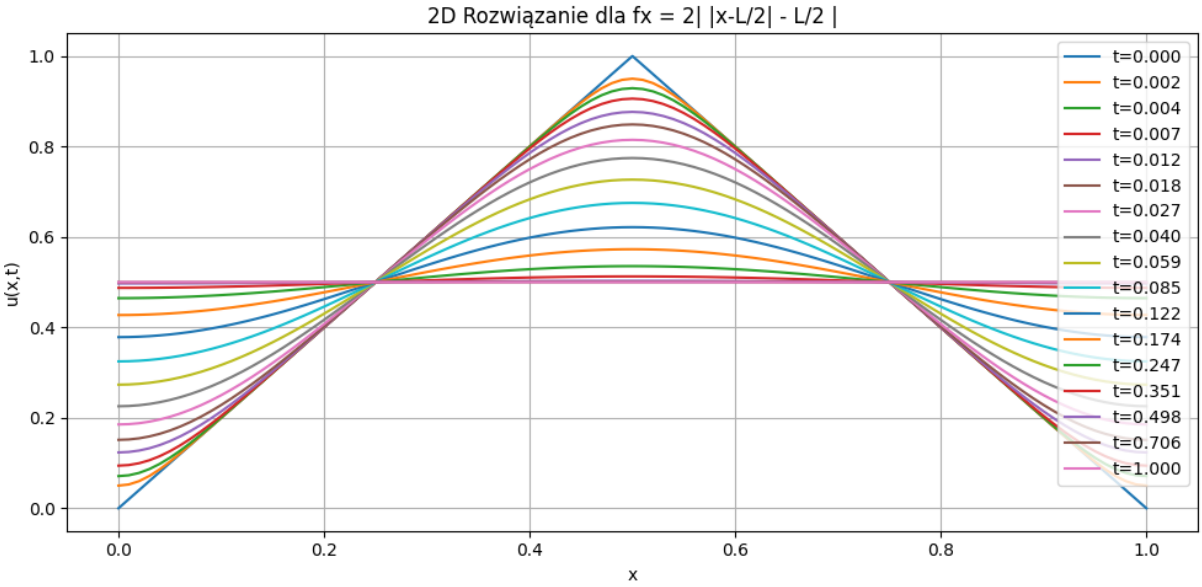
```python
#ZAD2

# Parametry
L = 1.0
D = 0.25
N = 100
x = np.arange(0, L+0.01, 0.01)
t_values = [0.000, 0.002, 0.004, 0.007, 0.012, 0.018, 0.027, 0.040, 0.059, 0.085, 0.122, 0.174, 0.247, 0.351, 0.498, 0.706, 1.000]

def compute_an(n, f):
    return (2 / L) * trapezoid(f(x) * np.cos(n * np.pi * x / L), x)

def u_xt(x, t, f):
    sum_series = np.zeros_like(x)

    a_0 = (2 / L) * trapezoid(f(x), x)
    sum_series += a_0 / 2

    for n in range(1, N+1):
        an = compute_an(n, f)
        sum_series += an * np.cos(n * np.pi * x / L) * np.exp(-n**2 * np.pi**2 * D * t / L**2)

    return sum_series

def plot_results(f, title):
    # Wykres 2D
    fig, ax = plt.subplots(figsize=(10, 5))
    for t in t_values:
        ax.plot(x, u_xt(x, t, f), label=f't={t:.3f}')
    ax.set_xlabel('x')
    ax.set_ylabel('u(x,t)')
    ax.set_title(f'2D Rozwiązanie {title}')
    ax.legend()
    ax.grid()
    plt.tight_layout()
    plt.show()
```

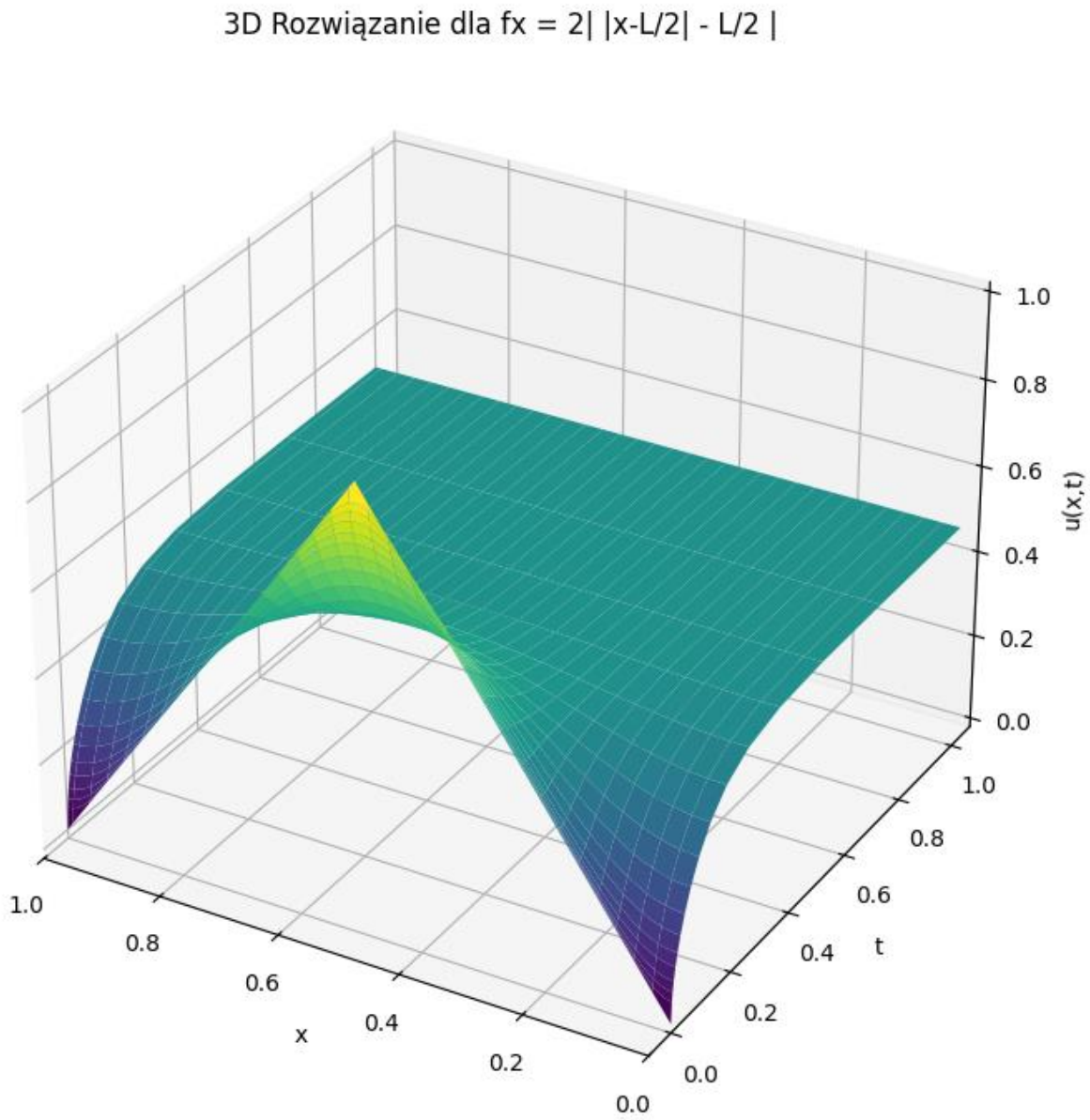
```python
    # Wykres 3D
    fig = plt.figure(figsize=(10, 7))
    ax = fig.add_subplot(111, projection='3d')
    X, T = np.meshgrid(x, t_values)
    U = np.array([u_xt(x, t, f) for t in t_values])
    ax.plot_surface(X, T, U, cmap='viridis')
    ax.set_xlabel('x')
    ax.set_ylabel('t')
    ax.set_zlabel('u(x,t)')
    ax.set_title(f'3D Rozwiązanie {title}')
    ax.set_xlim(np.max(x), np.min(x))
    plt.tight_layout()
    plt.show()

    # Wykres współczynników Fouriera
    fig, ax = plt.subplots(figsize=(10, 5))
    n_values = np.arange(1, N+1)
    an_values = np.array([compute_an(n, f) for n in n_values])
    ax.stem(n_values, an_values)
    ax.set_xlabel('n')
    ax.set_ylabel('a_n')
    ax.set_title(f'Współczynniki Fouriera {title}')
    ax.grid()
    plt.tight_layout()
    plt.show()

# Generowanie wykresów
plot_results(f1, 'dla fx = |sin(3πx/L)|')
plot_results(f2, 'dla fx = 2| |x-L/2| - L/2 |')
plot_results(f3, 'dla fx = Prostokąt (0.4 ≤ x ≤ 0.6)')
```

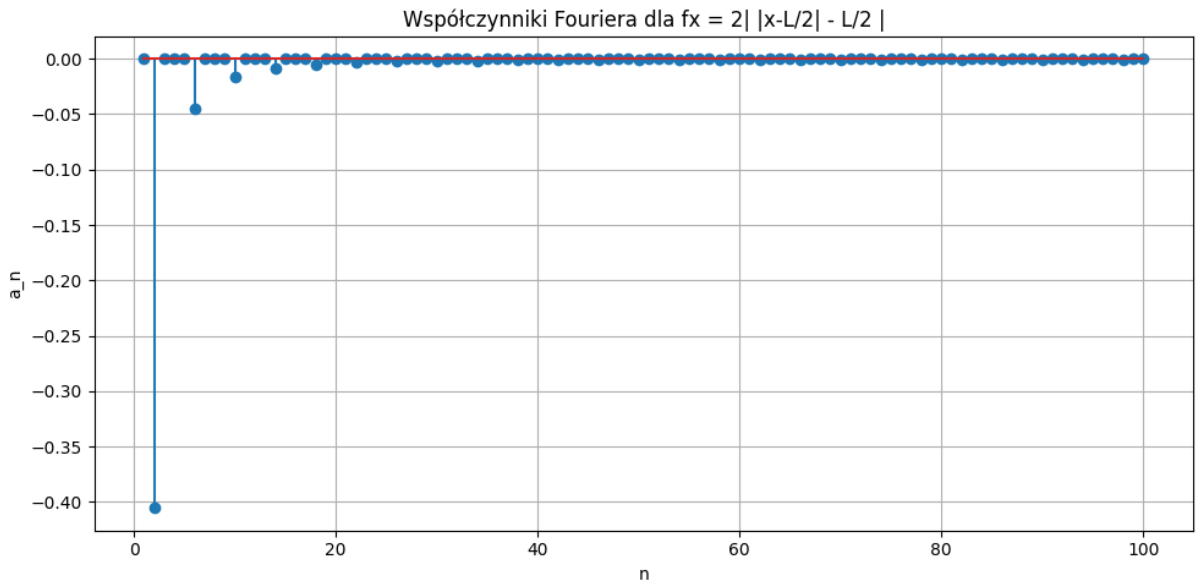


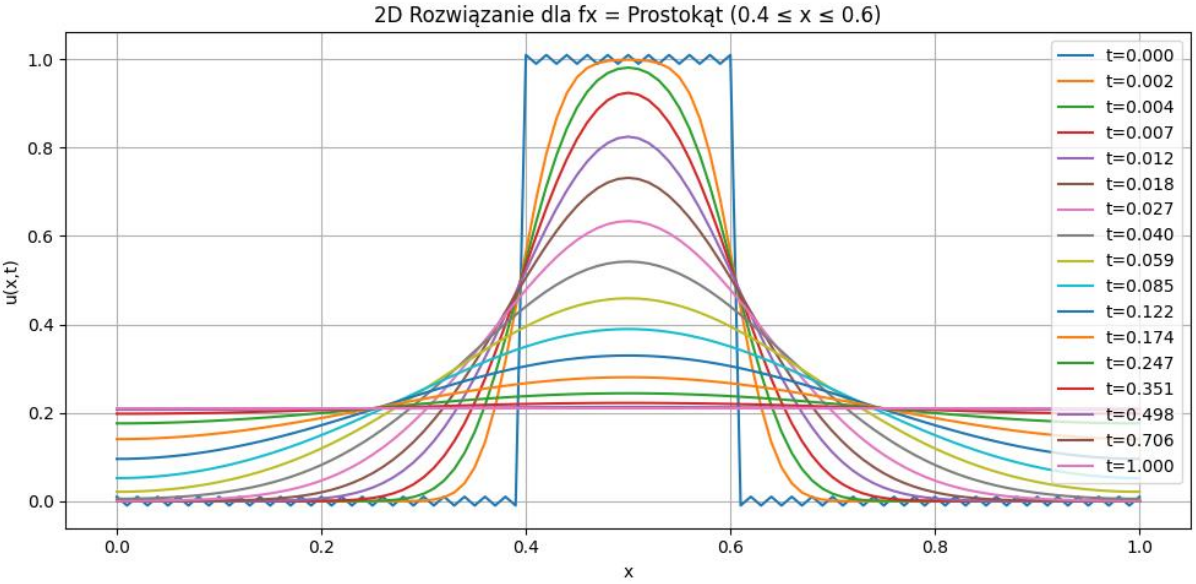

3D Rozwiązanie dla fx = |sin(3πx/L)|
u(x,t)
x
t

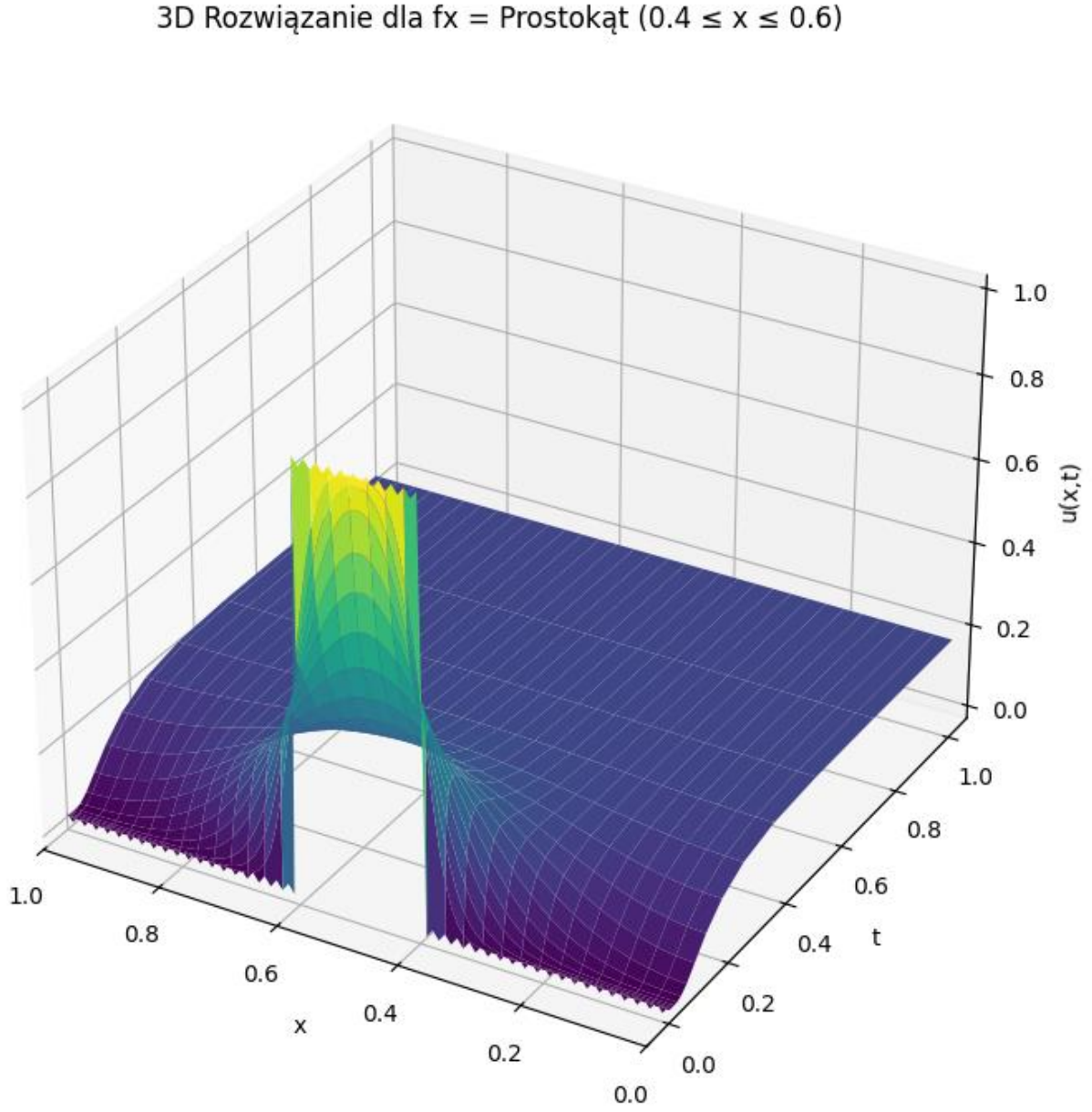

Współczynniki Fouriera dla fx = |sin(3πx/L)|
a_n
n

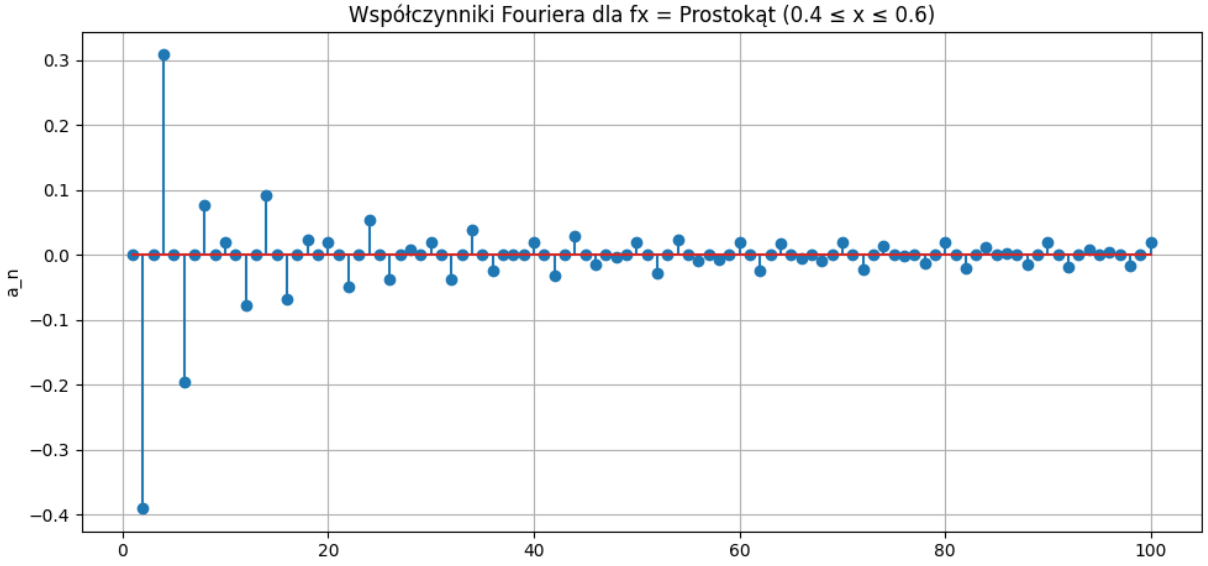

2D Rozwiązanie dla fx = 2| |x-L/2| - L/2 |
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000


3D Rozwiązanie dla fx = 2| |x-L/2| - L/2 |
u(x,t)
x
t


Współczynniki Fouriera dla fx = 2| |x-L/2| - L/2 |
a_n
n
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
−0.35
−0.40
0
20
40
60
80
100


2D Rozwiązanie dla fx = Prostokąt (0.4 ≤ x ≤ 0.6)
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000
0.0
0.2
0.4
0.6
0.8
1.0

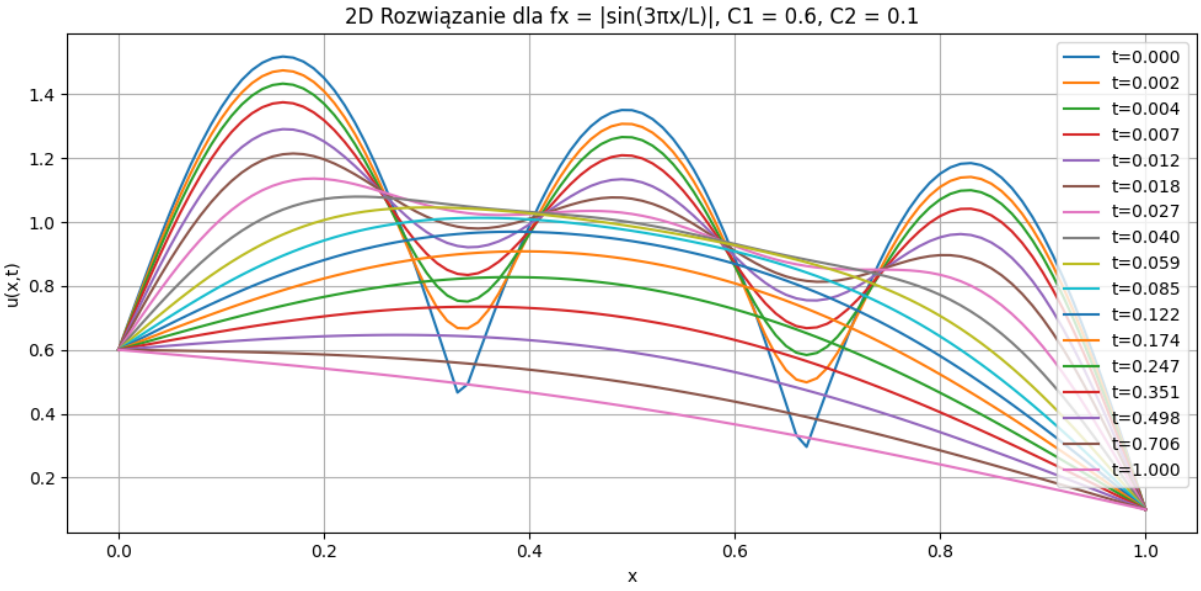

3D Rozwiązanie dla fx = Prostokąt (0.4 ≤ x ≤ 0.6)
u(x,t)
x
t
1.0
0.8
0.6
0.4
0.2
0.0


Współczynniki Fouriera dla fx = Prostokąt (0.4 ≤ x ≤ 0.6)
a_n
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0
20
40
60
80
100

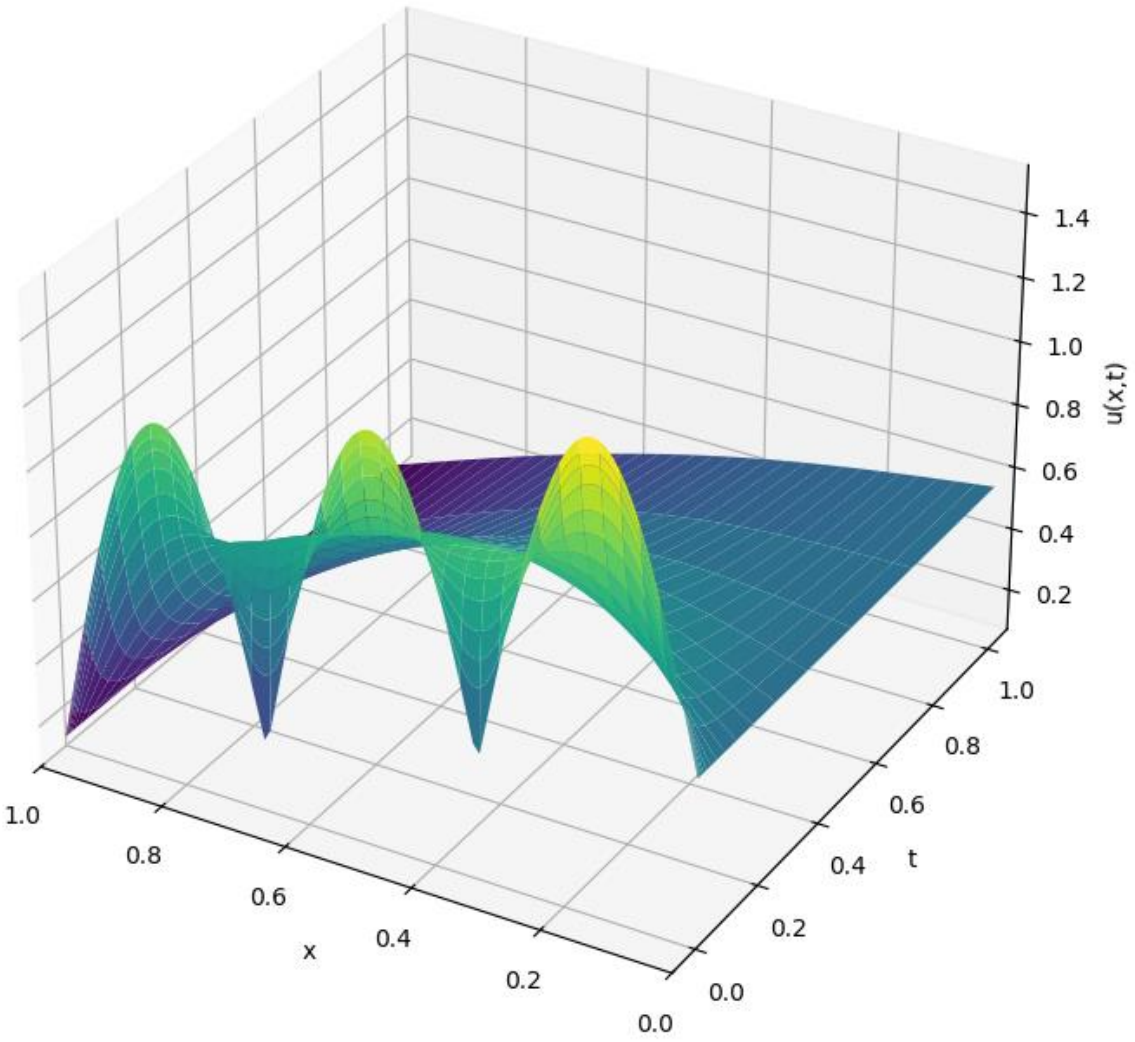
Zadanie 3:

```python
#ZAD3

def f1(x, L):
    return np.abs(np.sin(3 * np.pi * x / L))

def f2(x, L):
    return 2 * np.abs(np.abs(x - L/2) - L/2)

def f3(x, L):
    return np.where((x >= 0.4 * L) & (x <= 0.6 * L), 1, 0)

L = 1
D = 0.25
dx = 0.01
dt = 0.0001
x = np.arange(0, L + dx, dx)
t_values = [0.000, 0.002, 0.004, 0.007, 0.012, 0.018, 0.027, 0.040, 0.059,
            0.085, 0.122, 0.174, 0.247, 0.351, 0.498, 0.706, 1.000]
C1_C2_cases = [(0.6, 0.1), (0.2, 0.7), (0.1, 0.4)]

def solve_diffusion(f, C1, C2, L, D, dx, dt, N):
    u = f(x, L)
    v = C1 + (C2 - C1) * x / L
    u = u + v
    r = D * dt / dx**2

    for _ in range(N):
        u_new = u.copy()
        u_new[1:-1] = u[1:-1] + r * (u[2:] - 2*u[1:-1] + u[:-2])
        u_new[0] = C1
        u_new[-1] = C2
        u = u_new

    return u
```

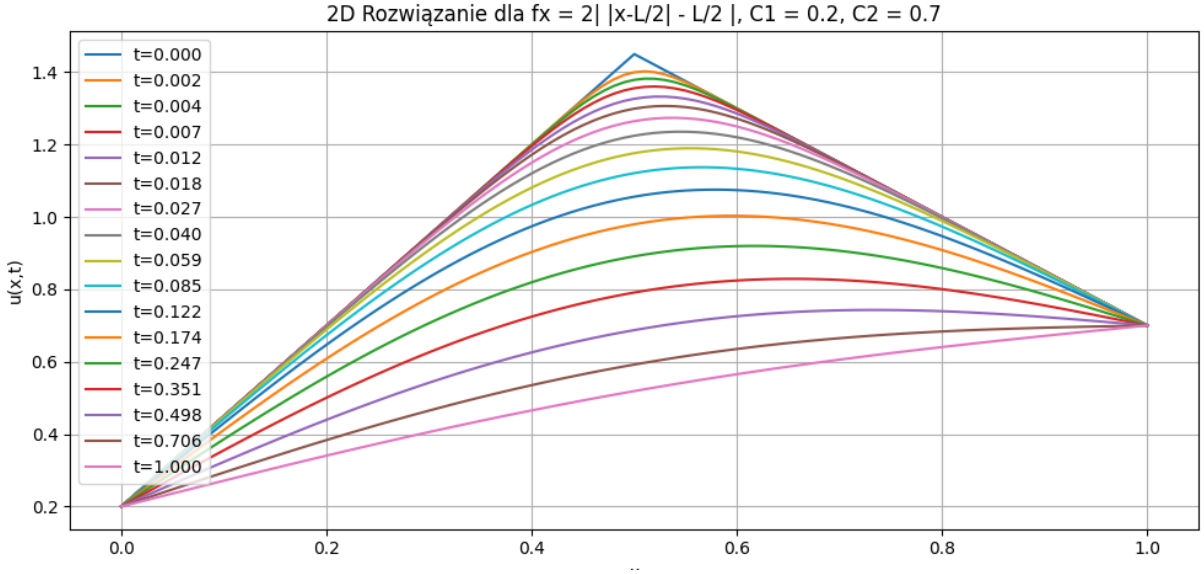
```python
# Funkcja do rysowania wykresów 2D i 3D
def plot_results(f, title, C1, C2, L, D, dx, dt, t_values, x):
    # Wykres 2D
    fig, ax = plt.subplots(figsize=(10, 5))
    for t in t_values:
        N_t = int(t / dt)
        u_sol = solve_diffusion(f, C1, C2, L, D, dx, dt, N_t)
        ax.plot(x, u_sol, label=f't={t:.3f}')
    ax.set_xlabel('x')
    ax.set_ylabel('u(x,t)')
    ax.set_title(f'2D Rozwiązanie {title}')
    ax.legend()
    ax.grid()
    plt.tight_layout()
    plt.show()  # Wyświetlenie wykresu 2D

    # Wykres 3D
    fig = plt.figure(figsize=(10, 7))
    ax = fig.add_subplot(111, projection='3d')
    X, T = np.meshgrid(x, t_values)
    U = np.array([solve_diffusion(f, C1, C2, L, D, dx, dt, int(t/dt)) for t in t_values])
    ax.plot_surface(X, T, U, cmap='viridis')
    ax.set_xlabel('x')
    ax.set_ylabel('t')
    ax.set_zlabel('u(x,t)')
    ax.set_title(f'3D Rozwiązanie {title}')
    ax.set_xlim(np.max(x), np.min(x))  # Odwrócenie osi X
    plt.tight_layout()
    plt.show()  # Wyświetlenie wykresu 3D

# Generowanie wykresów
plot_results(f1, 'dla fx = |sin(3πx/L)|, C1 = 0.6, C2 = 0.1', 0.6, 0.1, L, D, dx, dt, t_values, x)
plot_results(f2, 'dla fx = 2| |x-L/2| - L/2 |, C1 = 0.2, C2 = 0.7', 0.2, 0.7, L, D, dx, dt, t_values, x)
plot_results(f3, 'dla fx = Prostokąt (0.4 ≤ x ≤ 0.6), C1 = 0.1, C2 = 0.4', 0.1, 0.4, L, D, dx, dt, t_values, x)
```



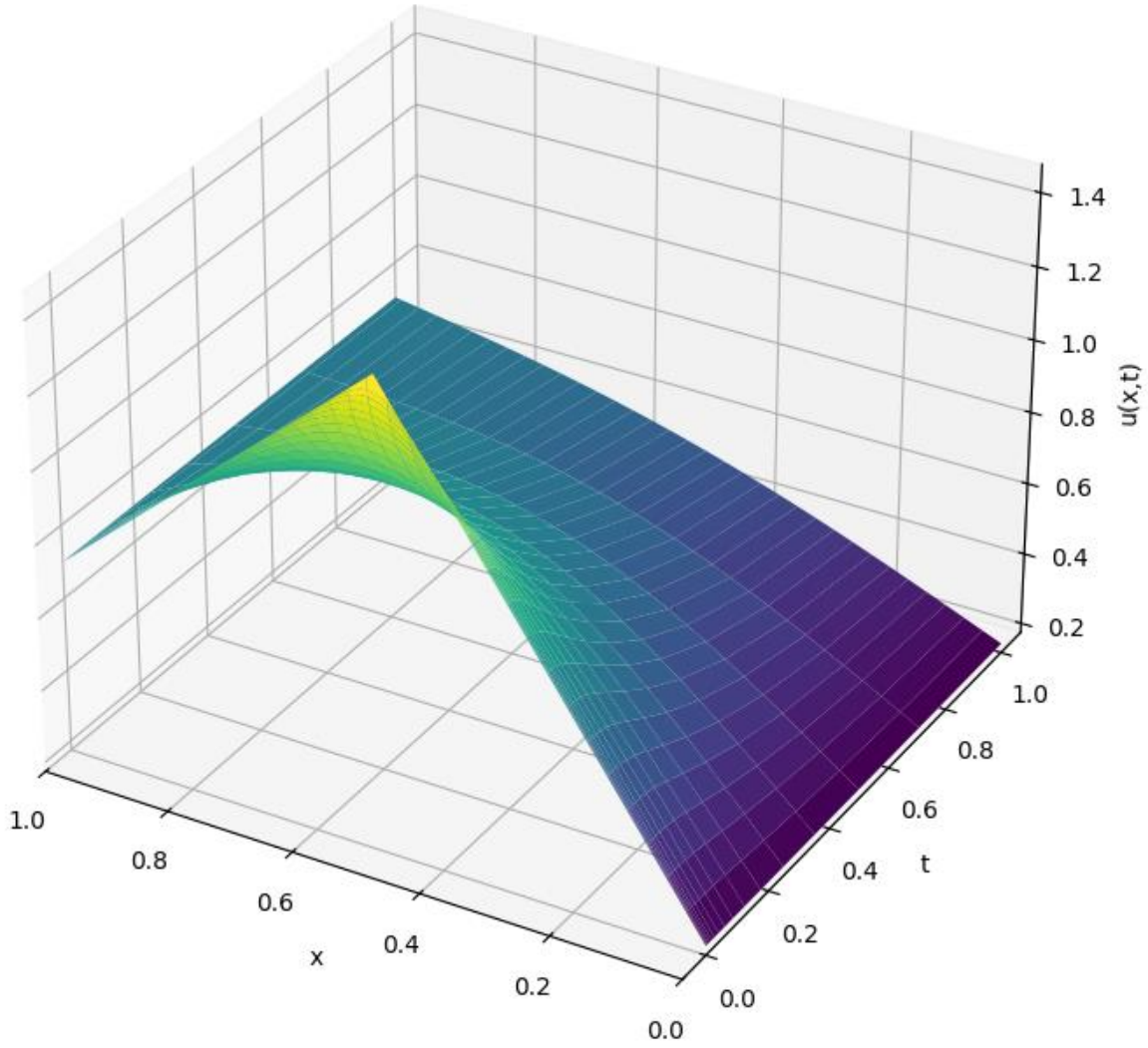
3D Rozwiązanie dla fx = |sin(3πx/L)|, C1 = 0.6, C2 = 0.1



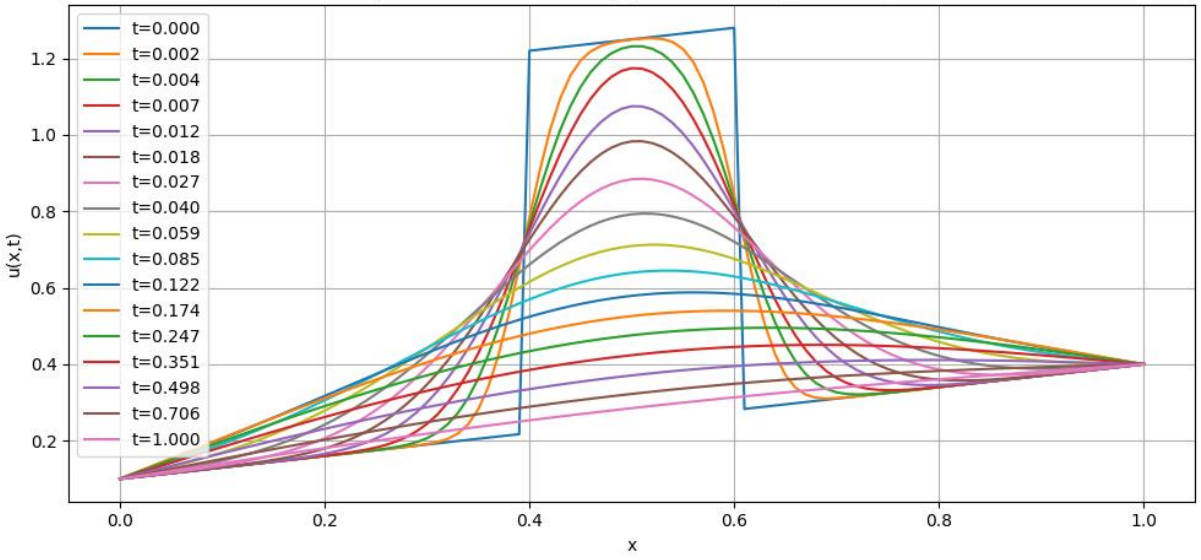
2D Rozwiązanie dla fx = 2| |x-L/2| - L/2 |, C1 = 0.2, C2 = 0.7

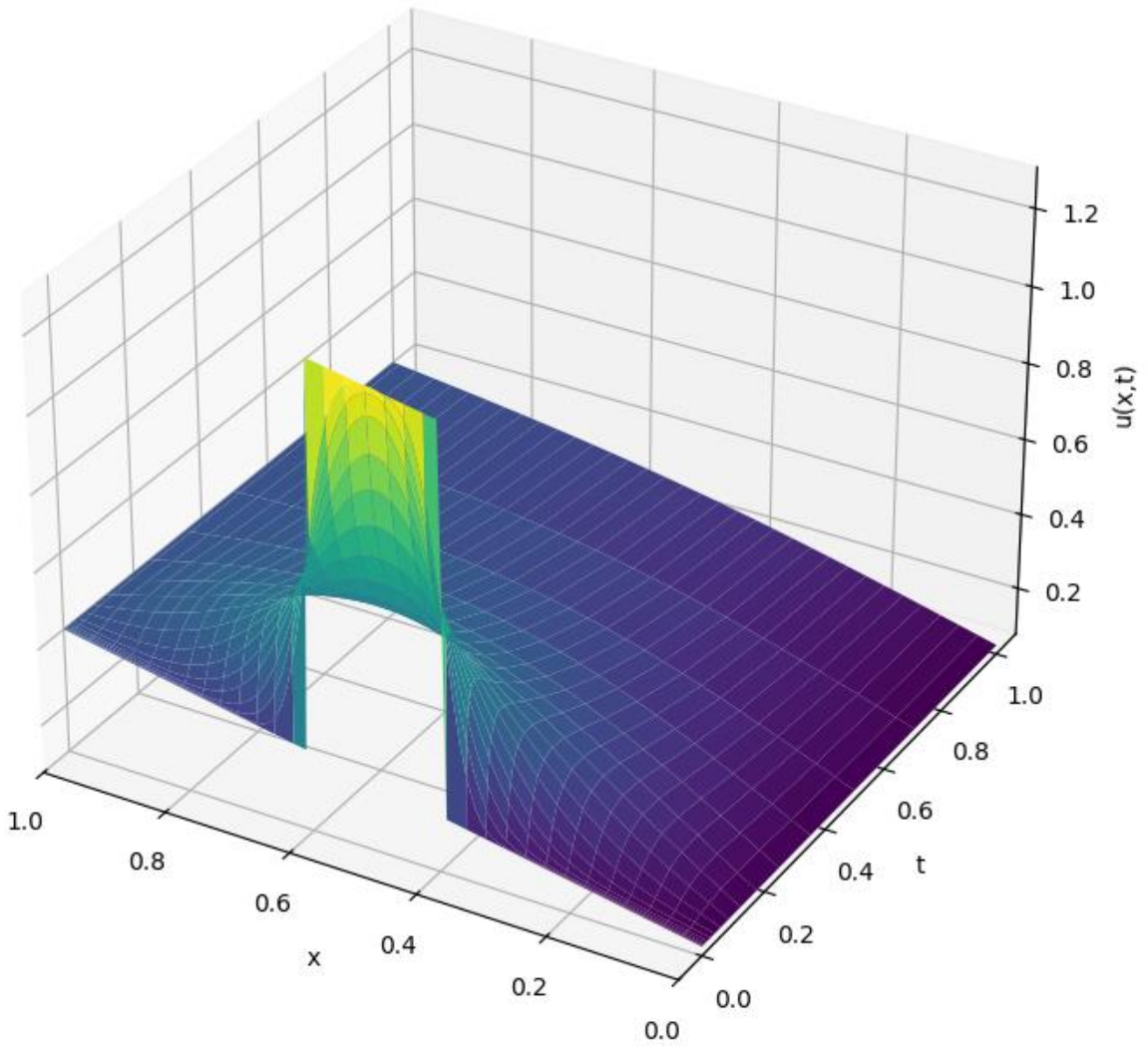


3D Rozwiązanie dla fx = 2| |x-L/2| - L/2 |, C1 = 0.2, C2 = 0.7



2D Rozwiązanie dla fx = Prostokąt (0.4 ≤ x ≤ 0.6), C1 = 0.1, C2 = 0.4



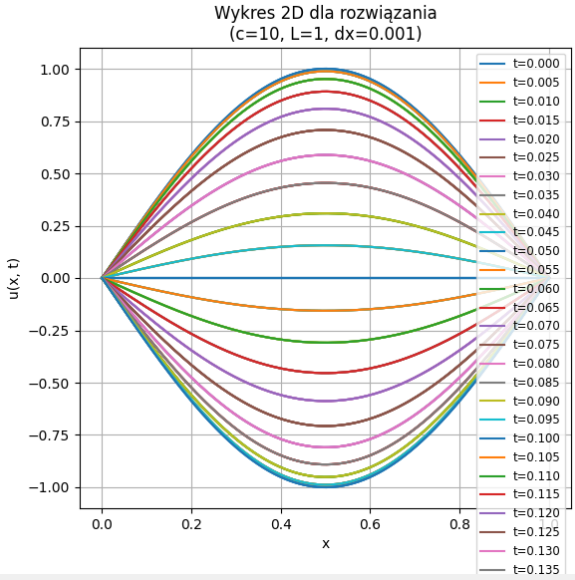

3D Rozwiązanie dla fx = Prostokąt (0.4 ≤ x ≤ 0.6), C1 = 0.1, C2 = 0.4
u(x,t)
1.2
1.0
0.8
0.6
0.4
0.2
x
1.0
0.8
0.6
0.4
0.2
0.0
t
0.0
0.2
0.4
0.6
0.8
1.0

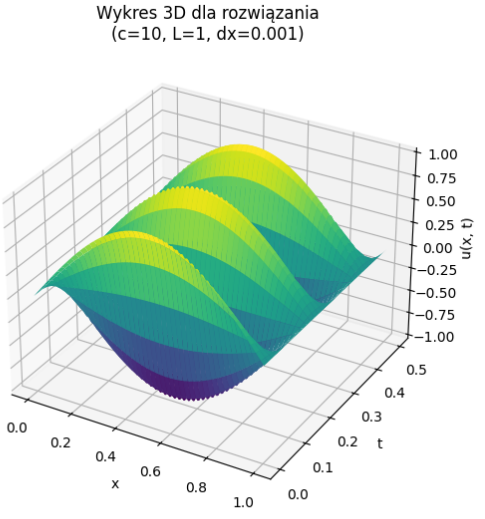
Zadanie 4:

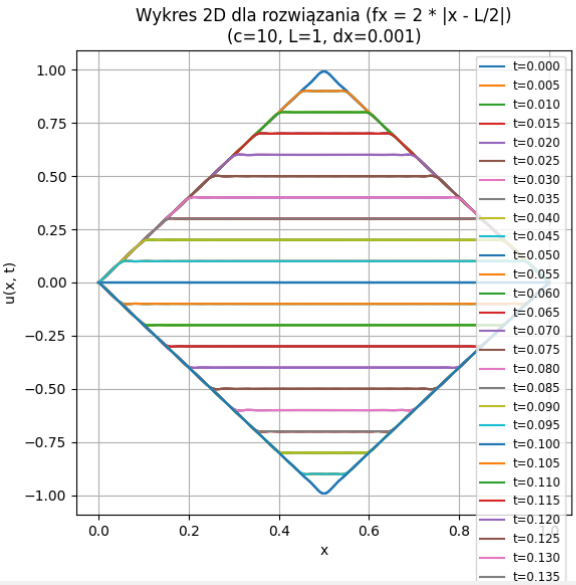
```python
#ZAD4

L = 1.0
c = 1.0
T = 2.0
dx = 0.001
dt = 0.001
x = np.arange(0, L + dx, dx)
t = np.arange(0, T + dt, dt)

def f1(x, L):
    return np.abs(np.sin(np.pi * x / L))

def f2(x, L):
    return 2 * np.abs(np.abs(x - L / 2) - L / 2)

def compute_bn(fx, n, L):
    integrand = lambda x: fx(x, L) * np.sin(n * np.pi * x / L)
    return (2 / L) * np.trapezoid(integrand(x), x)

def compute_u(x, t, L, c, fx, num_terms=50):
    u = np.zeros((len(t), len(x)))
    for n in range(1, num_terms + 1):
        bn = compute_bn(fx, n, L)
        u += bn * np.sin(n * np.pi * x / L)[None, :] * np.cos(n * np.pi * c * t / L)[:, None]
    return u

fx = f1

u = compute_u(x, t, L, c, fx)

c = 10.0
dx = 0.001
x = np.arange(0, L + dx, dx)
t = np.arange(0, 0.5 + 0.005, 0.005)

u = compute_u(x, t, L, c, fx)

fig = plt.figure(figsize=(14, 6))
```

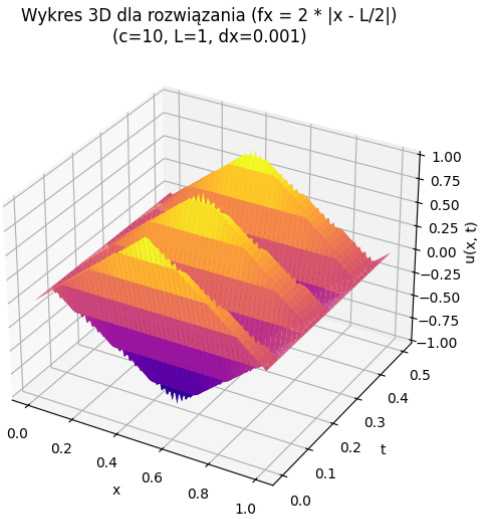
```python
# Wykres 2D
ax1 = fig.add_subplot(1, 2, 1)
step = 1  # Display all lines
for i, ti in enumerate(t[::step]):
    ax1.plot(x, u[i * step, :], label=f't={ti:.3f}')
ax1.set_xlabel('x')
ax1.set_ylabel('u(x, t)')
ax1.set_title('Wykres 2D dla rozwiązania\n(c=10, L=1, dx=0.001)')
ax1.legend(loc='upper right', fontsize='small', ncol=1)
ax1.grid()

# Wykres 3D
ax2 = fig.add_subplot(1, 2, 2, projection='3d')
X, T = np.meshgrid(x, t)
ax2.plot_surface(X, T, u, cmap='viridis', edgecolor='none')
ax2.set_xlabel('x')
ax2.set_ylabel('t')
ax2.set_zlabel('u(x, t)')
ax2.set_title('Wykres 3D dla rozwiązania\n(c=10, L=1, dx=0.001)')

plt.tight_layout()
plt.show()

fx = f2
u_f2 = compute_u(x, t, L, c, fx)

fig = plt.figure(figsize=(14, 6))

# Wykres 2D
ax1 = fig.add_subplot(1, 2, 1)
for i, ti in enumerate(t[::step]):
    ax1.plot(x, u_f2[i * step, :], label=f't={ti:.3f}')
ax1.set_xlabel('x')
ax1.set_ylabel('u(x, t)')
ax1.set_title('Wykres 2D dla rozwiązania (fx = 2 * |x - L/2|)\n(c=10, L=1, dx=0.001)')
ax1.legend(loc='upper right', fontsize='small', ncol=1)
ax1.grid()
```

```python
# Wykres 3D
ax2 = fig.add_subplot(1, 2, 2, projection='3d')
X, T = np.meshgrid(x, t)
ax2.plot_surface(X, T, u_f2, cmap='plasma', edgecolor='none')
ax2.set_xlabel('x')
ax2.set_ylabel('t')
ax2.set_zlabel('u(x, t)')
ax2.set_title('Wykres 3D dla rozwiązania (fx = 2 * |x - L/2|)\n(c=10, L=1, dx=0.001)')

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


Wykres 2D dla rozwiązania
(c=10, L=1, dx=0.001)
u(x, t)
x
t=0.000
t=0.005
t=0.010
t=0.015
t=0.020
t=0.025
t=0.030
t=0.035
t=0.040
t=0.045
t=0.050
t=0.055
t=0.060
t=0.065
t=0.070
t=0.075
t=0.080
t=0.085
t=0.090
t=0.095
t=0.100
t=0.105
t=0.110
t=0.115
t=0.120
t=0.125
t=0.130
t=0.135


Wykres 3D dla rozwiązania
(c=10, L=1, dx=0.001)
u(x, t)
x
t

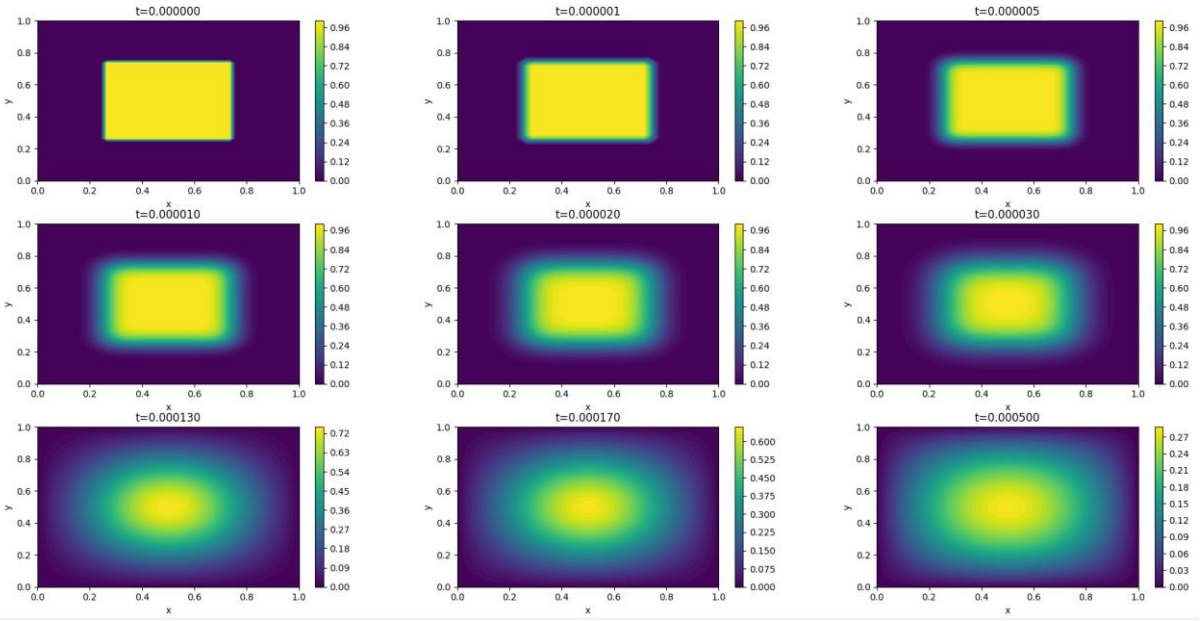

Wykres 2D dla rozwiązania (fx = 2 * |x - L/2|)
(c=10, L=1, dx=0.001)
u(x, t)
x
t=0.000
t=0.005
t=0.010
t=0.015
t=0.020
t=0.025
t=0.030
t=0.035
t=0.040
t=0.045
t=0.050
t=0.055
t=0.060
t=0.065
t=0.070
t=0.075
t=0.080
t=0.085
t=0.090
t=0.095
t=0.100
t=0.105
t=0.110
t=0.115
t=0.120
t=0.125
t=0.130
t=0.135


Wykres 3D dla rozwiązania (fx = 2 * |x - L/2|)
(c=10, L=1, dx=0.001)
u(x, t)
x
t

Zadanie 5:

```python
#ZAD5

nx, ny = 50, 50
lx, ly = 1.0, 1.0
dx, dy = lx / (nx - 1), ly / (ny - 1)
tolerance = 1e-6

u = np.zeros((ny, nx))
center_x, center_y = nx // 2, ny // 2
square_size = min(nx, ny) // 4
u[center_y - square_size:center_y + square_size, center_x - square_size:center_x + square_size] = 1

u[0, :] = 0
u[:, 0] = 0
u[:, -1] = 0


def solve_laplace_time(u, dx, dy, tolerance, time_steps):
    results = [(0, u.copy())]
    error = 1.0
    step = 0
    while error > tolerance and step < max(time_steps):
        u_new = u.copy()
        u_new[1:-1, 1:-1] = 0.25 * (u[1:-1, :-2] + u[1:-1, 2:] + u[:-2, 1:-1] + u[2:, 1:-1])
        error = np.max(np.abs(u_new - u))
        u = u_new
        step += 1
        if step in time_steps:
            results.append((step, u.copy()))
    return results

time_steps = [0, 1, 5, 10, 20, 30, 130, 170, 500]

results = solve_laplace_time(u, dx, dy, tolerance, time_steps)

while len(results) < 9:
    results.append((None, np.zeros_like(u)))

x = np.linspace(0, lx, nx)
y = np.linspace(0, ly, ny)
X, Y = np.meshgrid(x, y)

fig, axes = plt.subplots(3, 3, figsize=(12, 12))
```

```python
for ax, (step, u_snapshot) in zip(axes.flat, results):
    if step is not None:
        contour = ax.contourf(X, Y, u_snapshot, 50, cmap='viridis')
        ax.set_title(f't={step * tolerance:.6f}')
    else:
        ax.set_title('')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    cbar = plt.colorbar(contour, ax=ax, orientation='vertical')
    cbar.set_label('')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```


t=0.000000
t=0.000001
t=0.000005
t=0.000010
t=0.000020
t=0.000030
t=0.000130
t=0.000170
t=0.000500
x
y