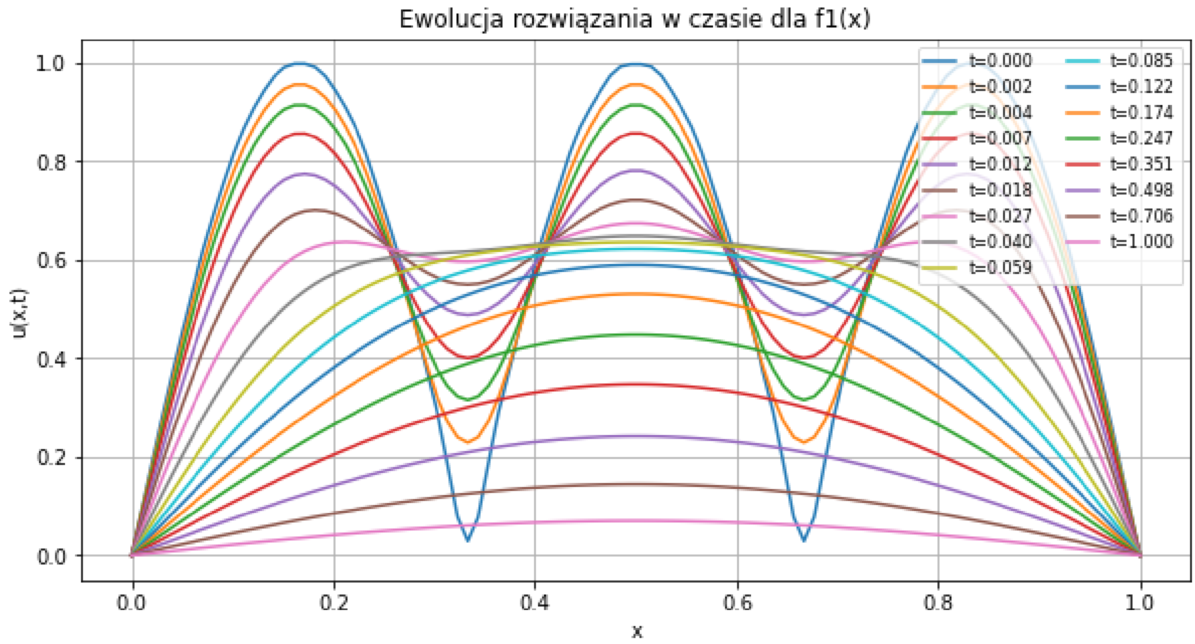
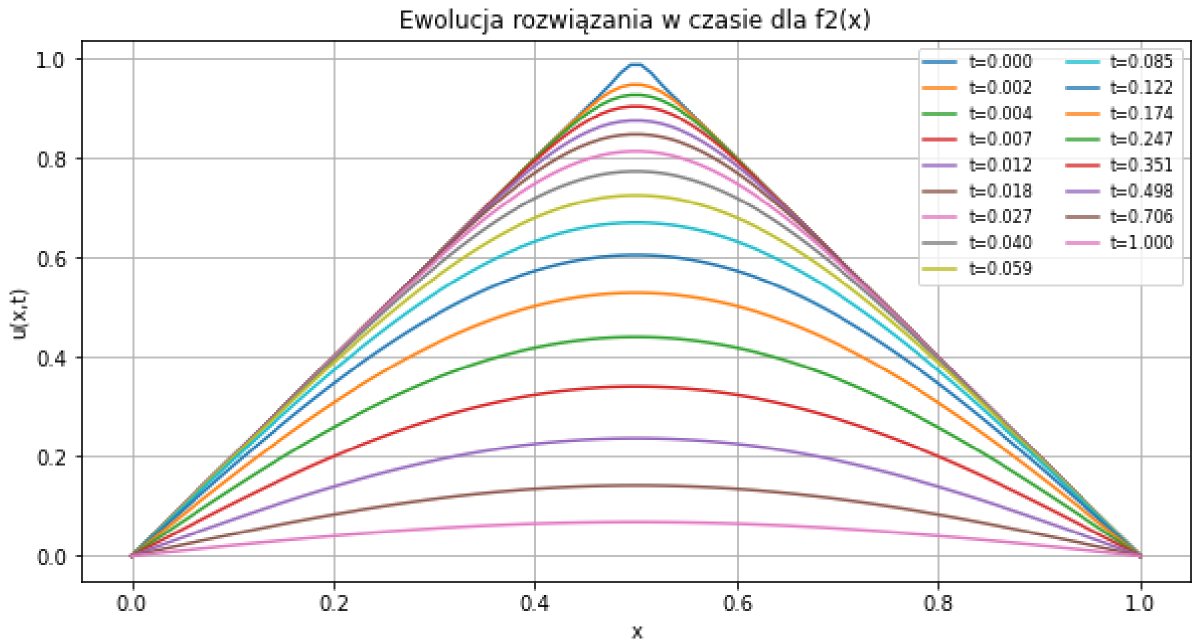
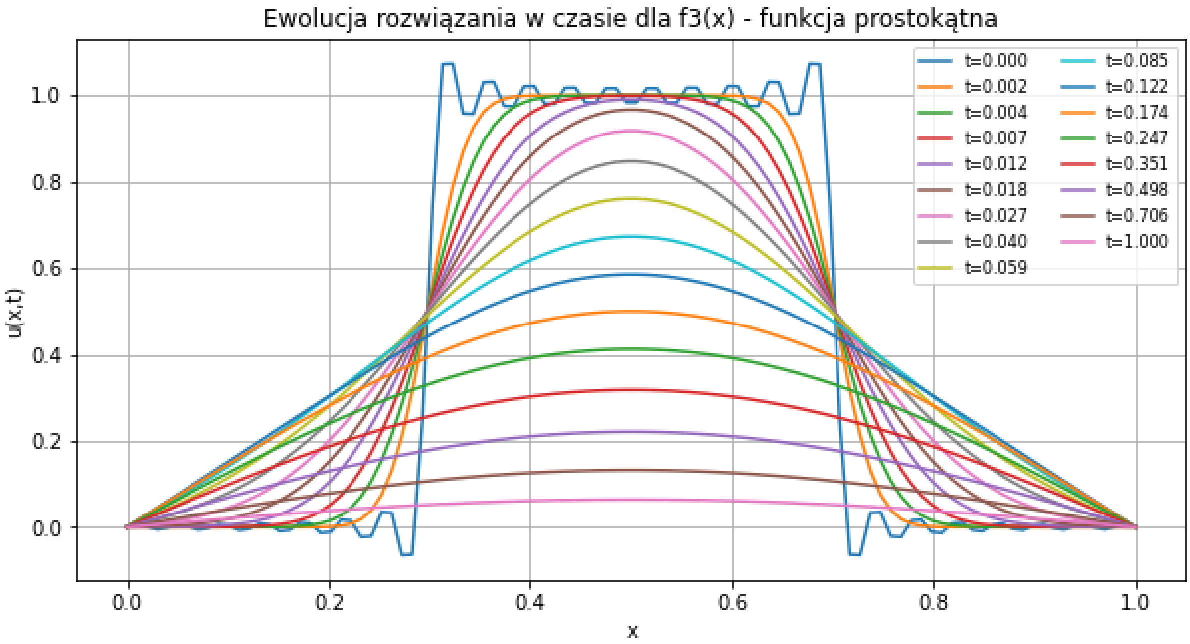
Sprawozdanie Lab3 Paweł Mozgowiec

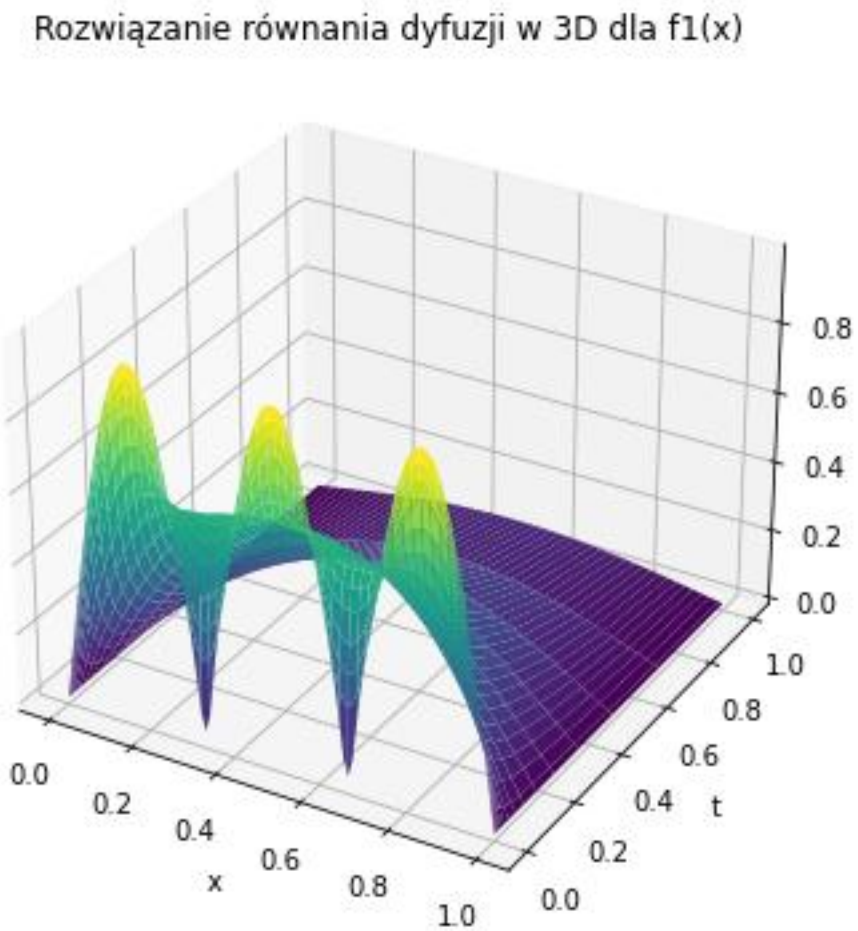
Zad1.

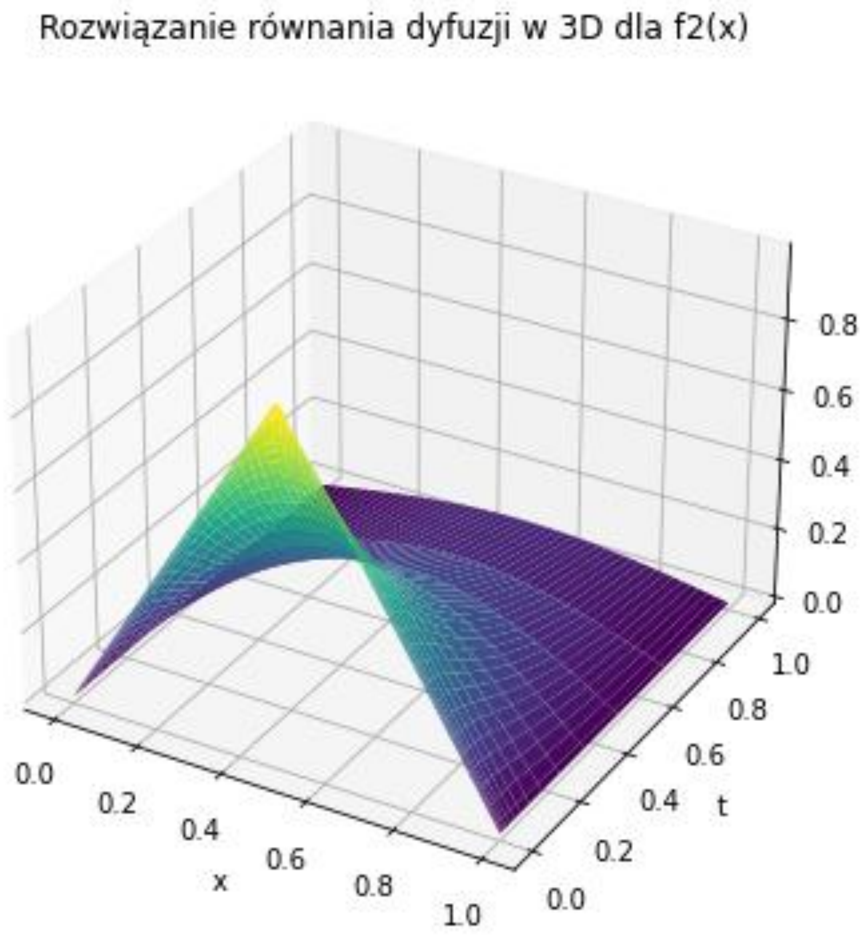
```python
# Parametry
L = 1.0  # Długość domeny
D = 0.25  # Współczynnik dyfuzji
dx = 0.01  # Krok przestrzenny
x = np.linspace(0, L, int(L/dx))  # Siatka przestrzenna
t_vals = np.array([0.000, 0.002, 0.004, 0.007, 0.012, 0.018, 0.027,
                   0.040, 0.059, 0.085, 0.122, 0.174, 0.247, 0.351,
                   0.498, 0.706, 1.000])  # Konkretne wartości czasu

# Warunki początkowe
def f1(x):  # Pierwsza funkcja początkowa
    return np.abs(np.sin(3 * np.pi * x / L))

def f2(x):  # Druga funkcja początkowa
    return 2 * np.abs(np.abs(x - L/2) - L/2)

def f3(x):  # Trzecia funkcja - prostokątna
    return np.where((x > 0.3) & (x < 0.7), 1, 0)

# Obliczanie współczynników Fouriera
def compute_bn(n, f, L):
    return (2 / L) * np.trapz(f(x) * np.sin(n * np.pi * x / L), x)

# Rozwiązanie równania
def u_xt(x, t, f, terms=50):
    u = np.zeros_like(x)
    for n in range(1, terms + 1):
        bn = compute_bn(n, f, L)
        u += bn * np.sin(n * np.pi * x / L) * np.exp(-n**2 * np.pi**2 * D * t / L**2)
    return u

# Obliczanie wartości u(x,t) dla każdej funkcji początkowej
solutions_f1 = np.array([u_xt(x, t, f1) for t in t_vals])
solutions_f2 = np.array([u_xt(x, t, f2) for t in t_vals])
solutions_f3 = np.array([u_xt(x, t, f3) for t in t_vals])
```

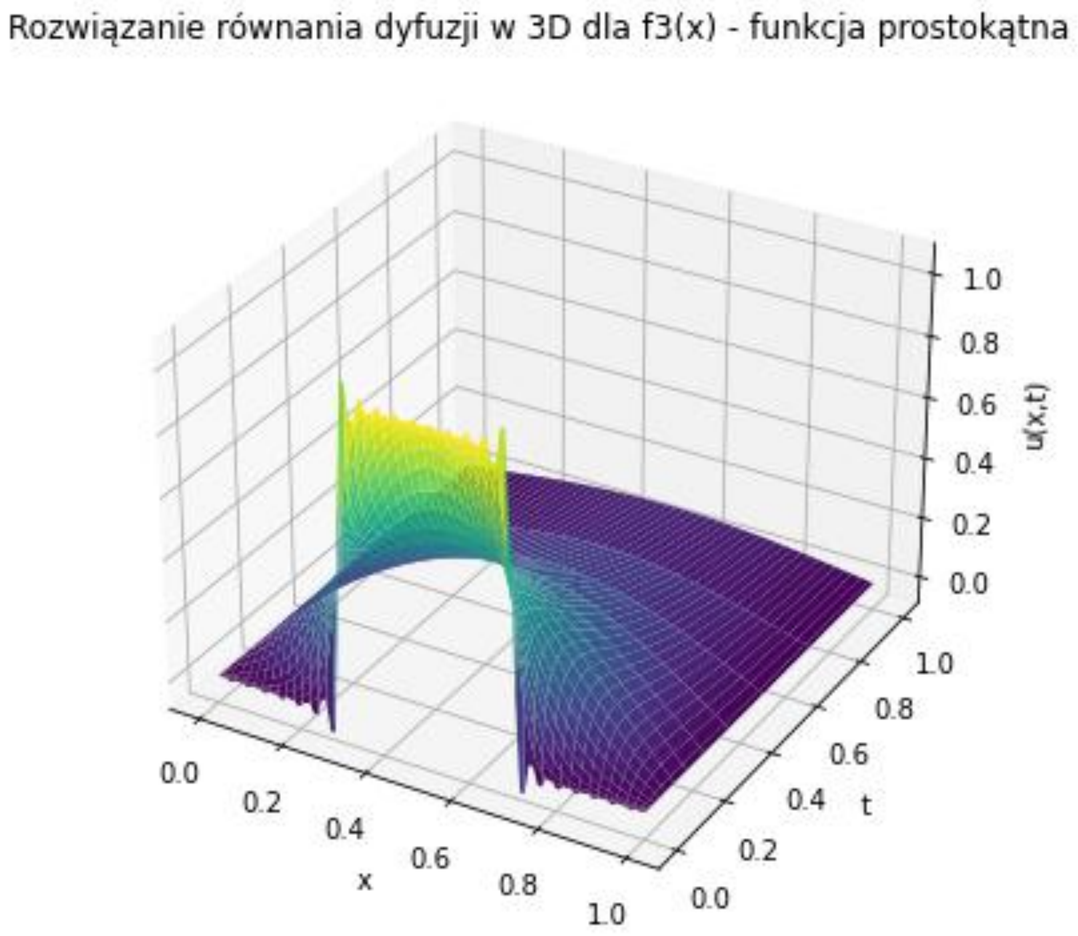

Ewolucja rozwiązania w czasie dla f1(x)
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000

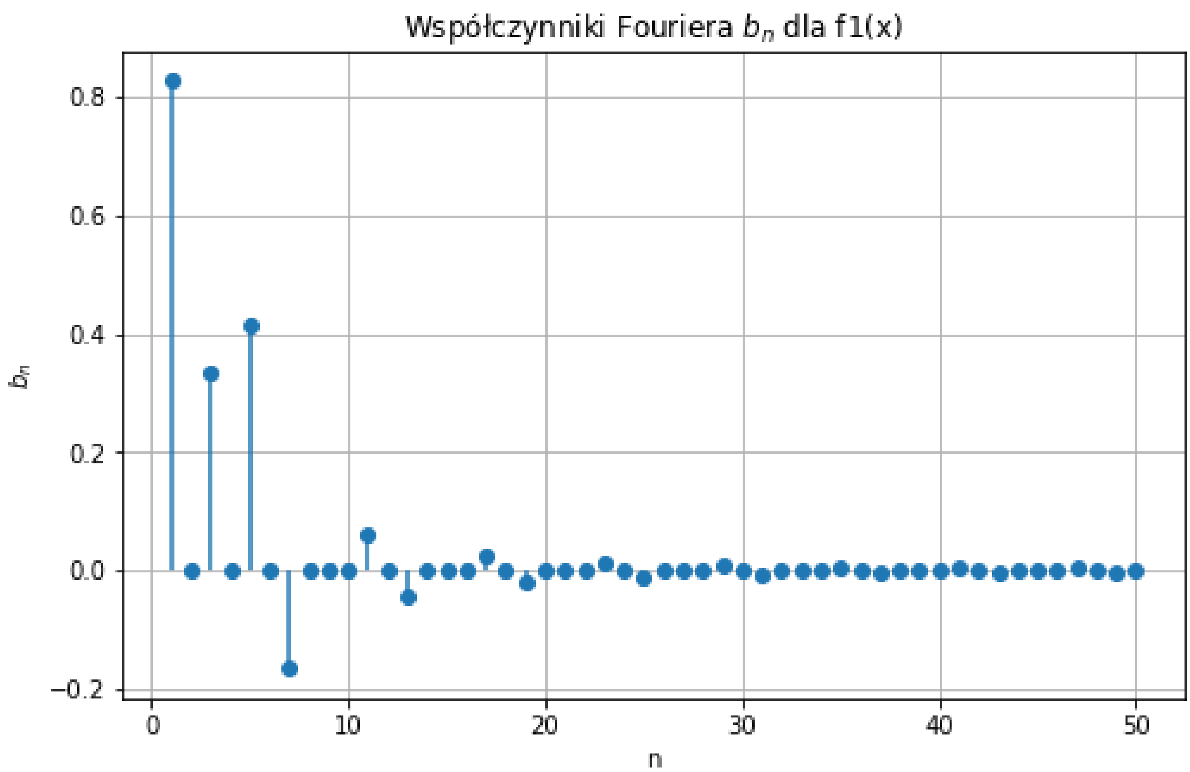

Ewolucja rozwiązania w czasie dla f2(x)
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000

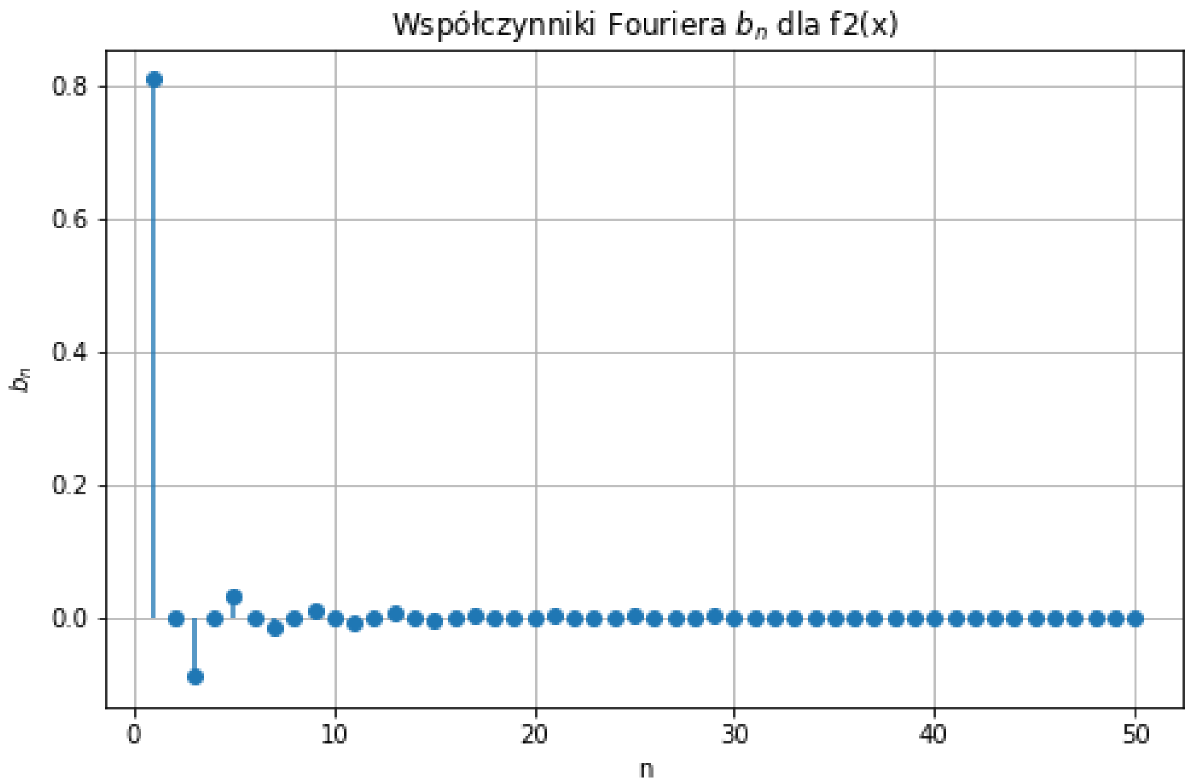

Ewolucja rozwiązania w czasie dla f3(x) - funkcja prostokątna
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000


Rozwiązanie równania dyfuzji w 3D dla f1(x)
x
t

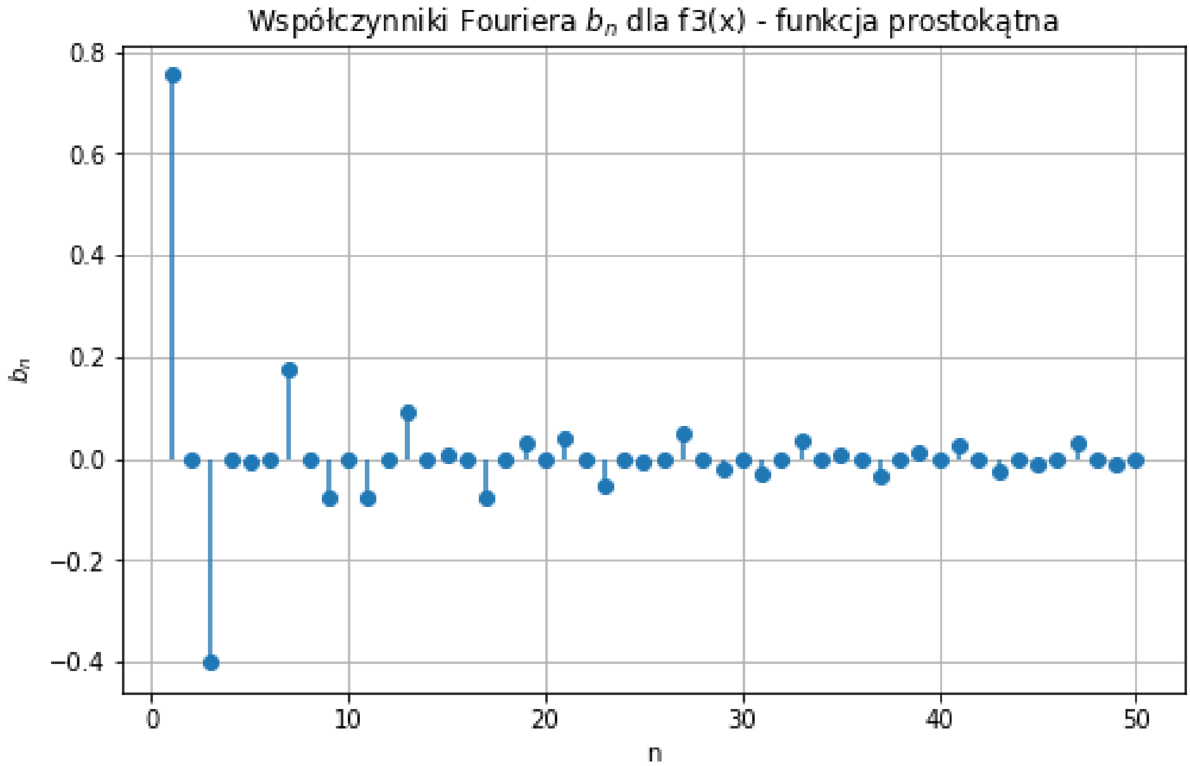

Rozwiązanie równania dyfuzji w 3D dla f2(x)
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x
0.0
0.2
0.4
0.6
0.8
1.0
t


Rozwiązanie równania dyfuzji w 3D dla f3(x) - funkcja prostokątna
u(x,t)
x
t


Współczynniki Fouriera b_n dla f1(x)
b_n
n



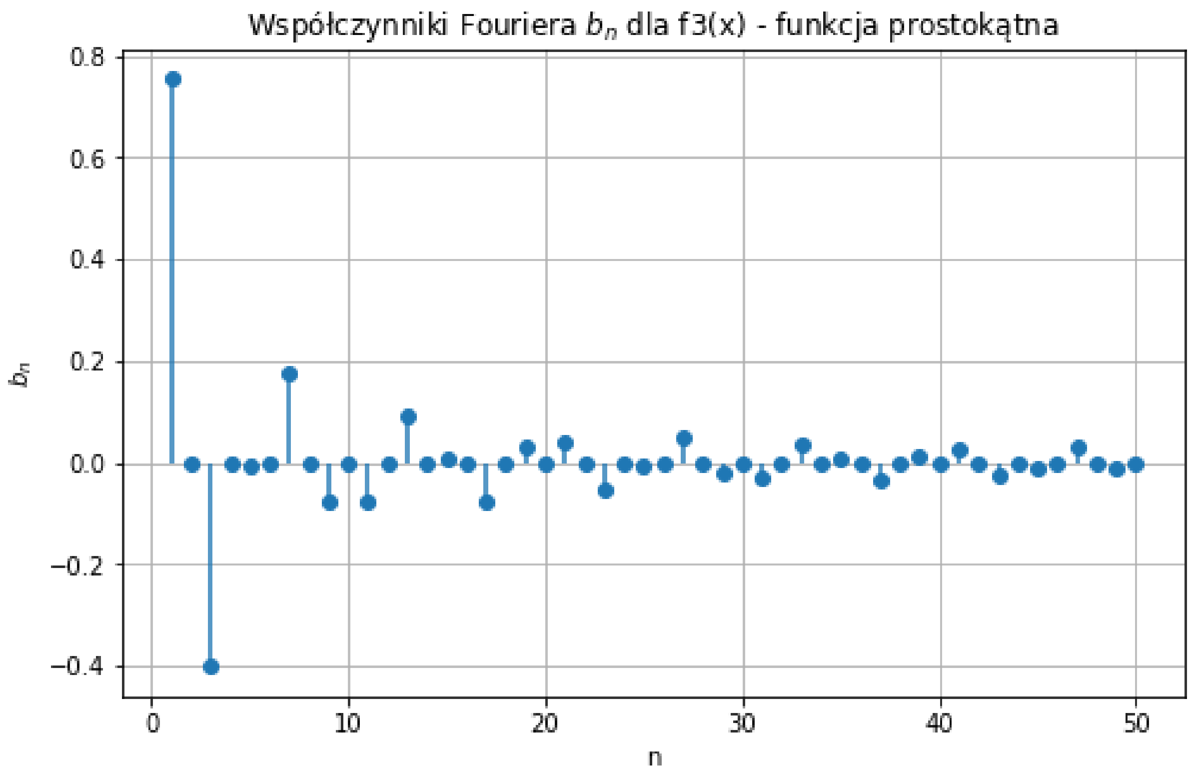
Współczynniki Fouriera b_n dla f2(x)
b_n
n

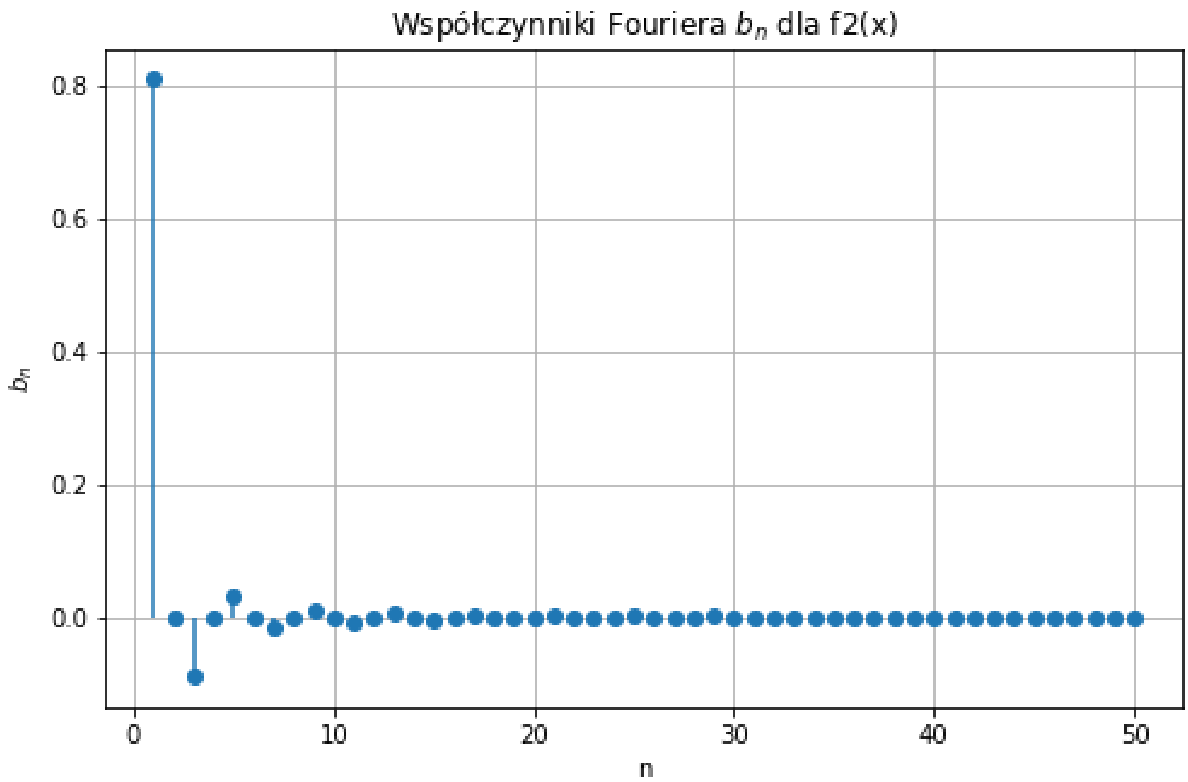


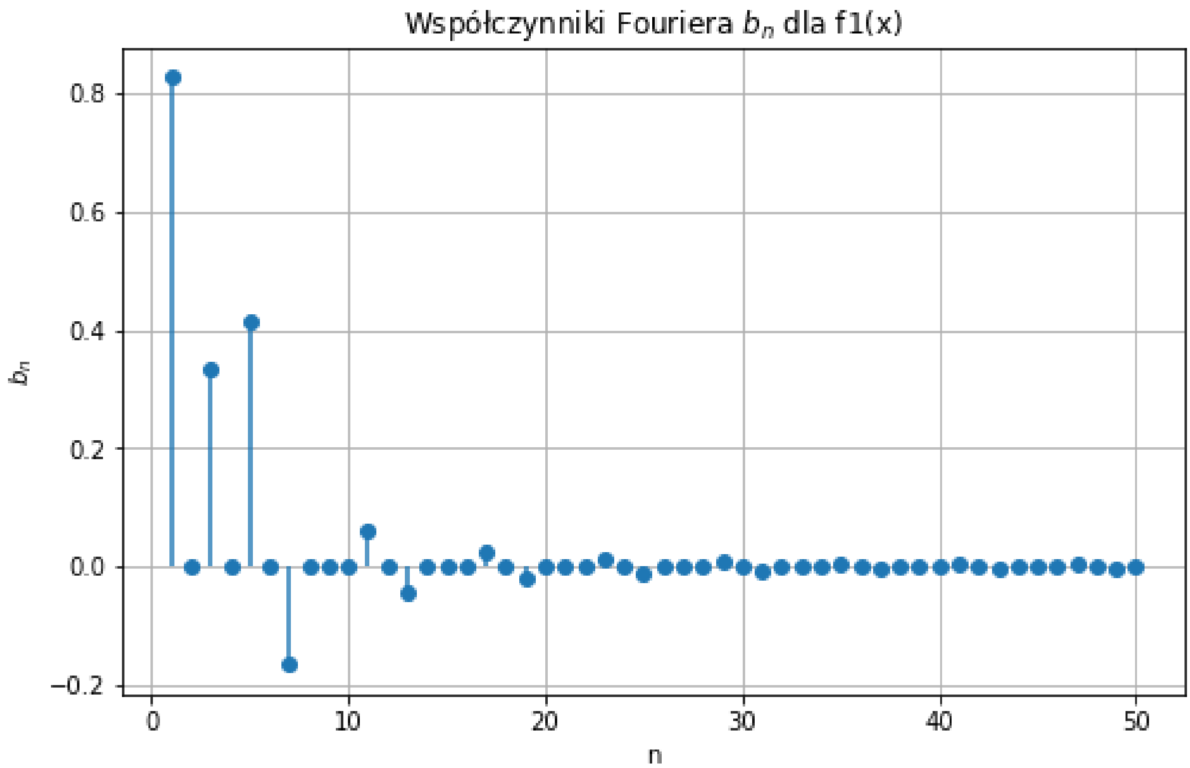
Współczynniki Fouriera b_n dla f3(x) - funkcja prostokątna
b_n
n

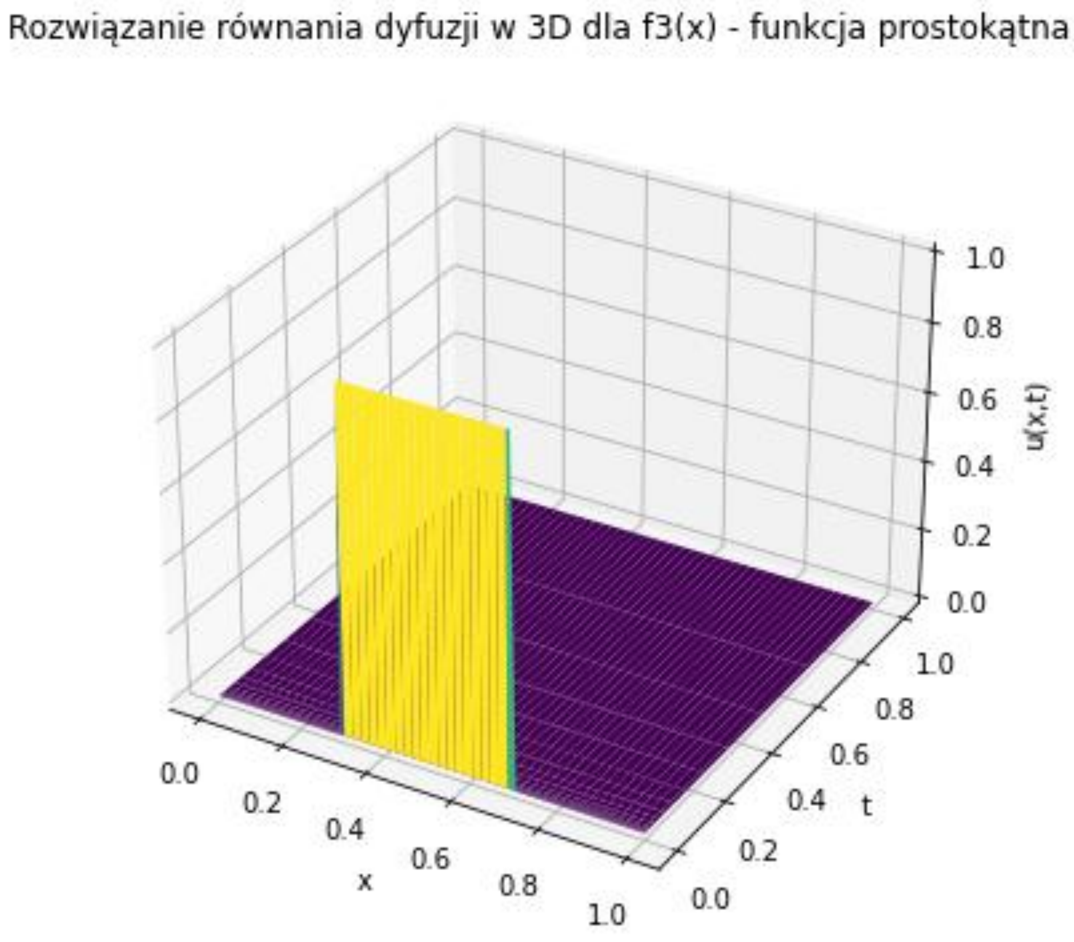
Zad2.

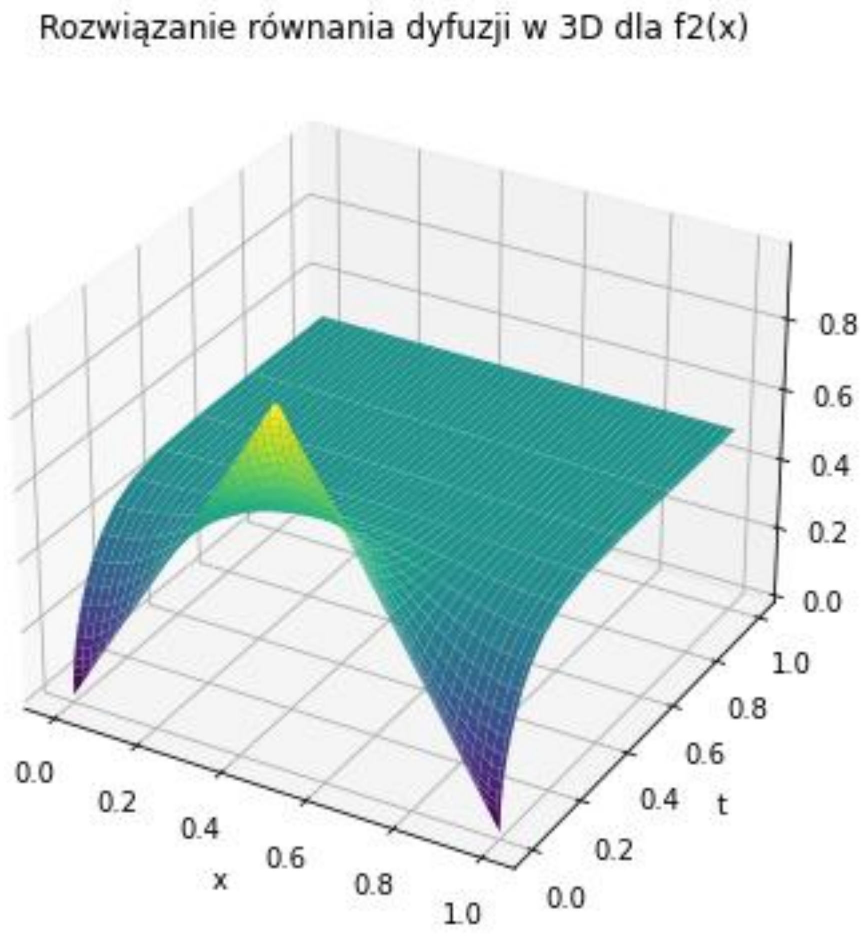
```python

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Parametry
L = 1.0       # Długość przedziału
D = 0.25      # Współczynnik dyfuzji
Nx = 100      # Liczba punktów przestrzennych
dx = L / Nx   # Rozdzielczość przestrzenna
dt = 0.00005   # Zmniejszamy dt, aby zapewnić stabilność

# Współczynniki Fouriera dla początkowego rozkładu f(x)
def f(x):
    return np.sin(2 * np.pi * x)  # Przykładowa funkcja początkowa

x = np.linspace(0, L, Nx)
u = f(x)  # Warunek początkowy

# Warunki początkowe
def f1(x):  # Pierwsza funkcja początkowa
    return np.abs(np.sin(3 * np.pi * x / L))

def f2(x):  # Druga funkcja początkowa
    return 2 * np.abs(np.abs(x - L / 2) - L / 2)
def f3(x):  # Trzecia funkcja - prostokątna
    return np.where((x > 0.3) & (x < 0.7), 1, 0)

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

# Warunki brzegowe Neumanna (pochodne zerowe na brzegach)
def apply_neumann_bc(u):
    u[0] = u[1]
    u[-1] = u[-2]
    return u

# Metoda różnic skończonych (schemat explicite)
def solve_diffusion(u, D, dx, dt,t,terms = 50):
    Nt = int(t / dt)
    r = D * dt / dx**2  # Liczba Couranta
    if r > 0.5:
        raise ValueError(f"Unstable: Courant number r = {r} exceeds 0.5.")
    for _ in range(Nt):
        u_new = u.copy()
        u_new[1:-1] = u[1:-1] + r * (u[2:] - 2 * u[1:-1] + u[:-2])
        u = apply_neumann_bc(u_new)
    return u

t_vals = [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]  # Wybrane momenty czasowe
solutions_f1 = [f1(x)]  # Przechowywanie wyników
solutions_f1 = np.array([solve_diffusion(solutions_f1[-1], D,dx,dt,t) for t in t_vals])
solutions_f2 = [f2(x)]  # Przechowywanie wyników
solutions_f2 = np.array([solve_diffusion(solutions_f2[-1], D,dx,dt,t) for t in t_vals])
solutions_f3 = [f3(x)]  # Przechowywanie wyników
solutions_f3 = np.array([solve_diffusion(solutions_f3[-1], D,dx,dt,t) for t in t_vals])
```

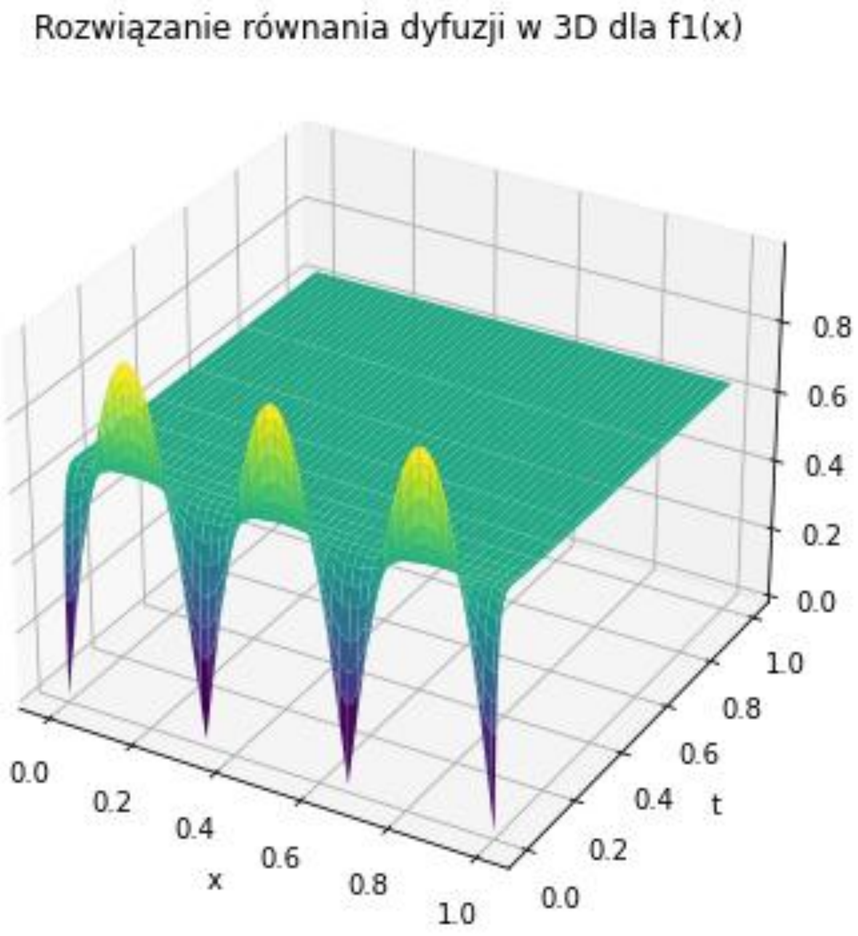


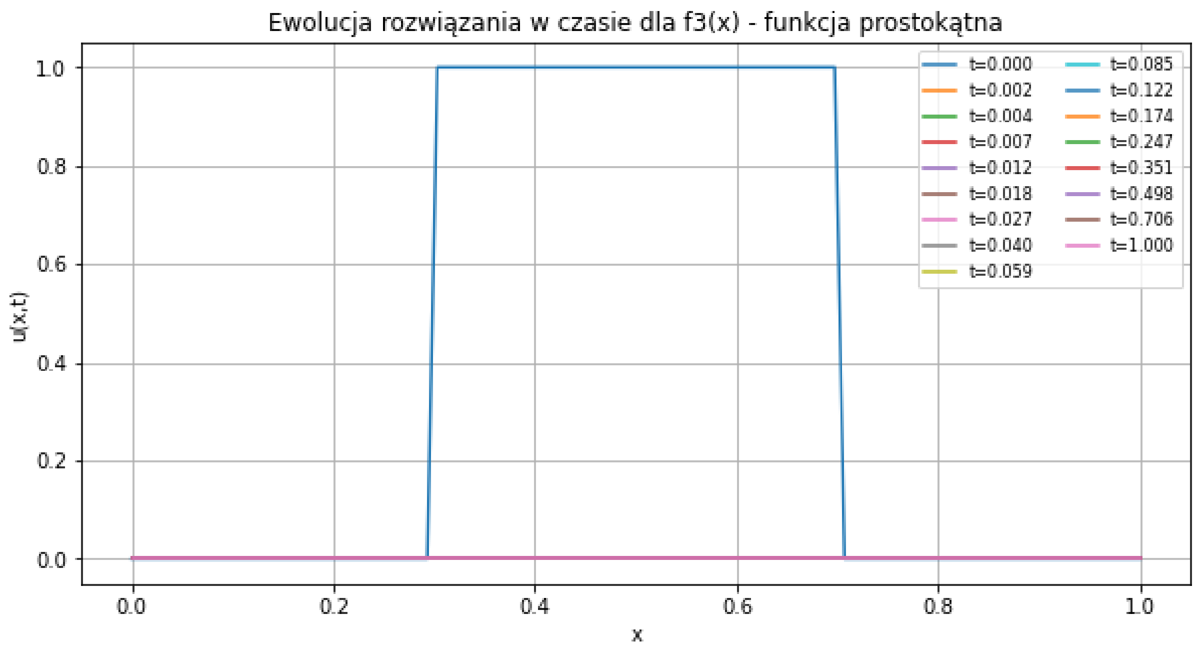

Współczynniki Fouriera b_n dla f2(x)
b_n
n
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50

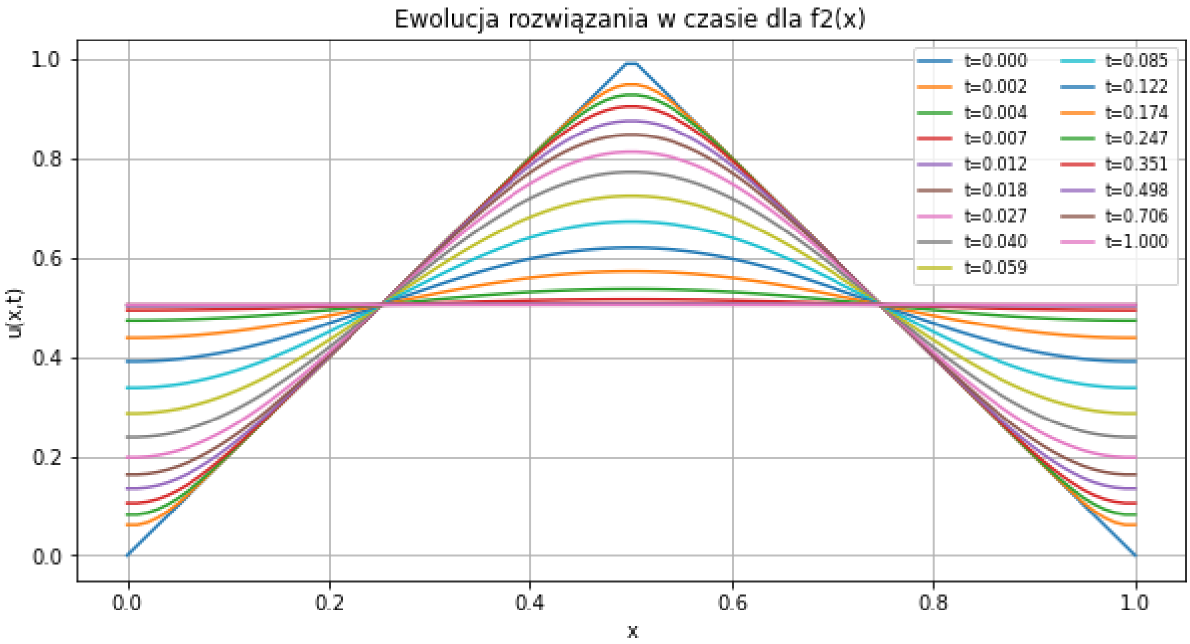

Współczynniki Fouriera b_n dla f1(x)
b_n
n
0.8
0.6
0.4
0.2
0.0
−0.2
0
10
20
30
40
50

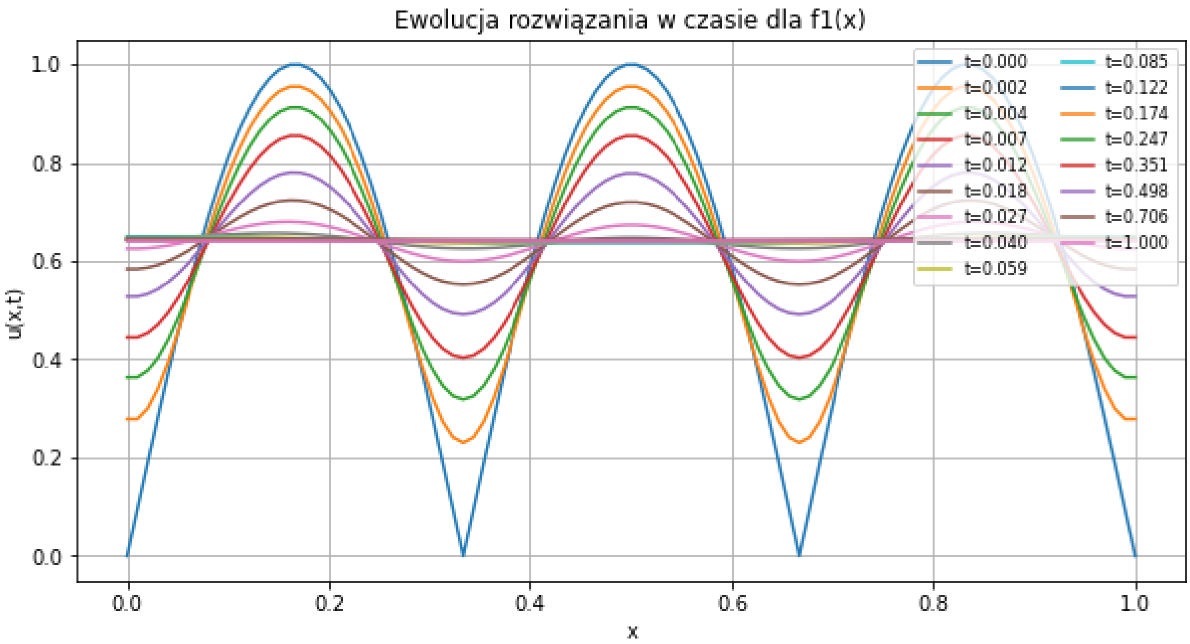

Rozwiązanie równania dyfuzji w 3D dla f3(x) - funkcja prostokątna
1.0
0.8
0.6
0.4
0.2
0.0
u(x,t)
0.0
0.2
0.4
0.6
0.8
1.0
x
t


Rozwiązanie równania dyfuzji w 3D dla f2(x)
0.0
0.2
0.4
0.6
0.8
1.0
x
t


Rozwiązanie równania dyfuzji w 3D dla f1(x)
0.0
0.2
0.4
0.6
0.8
1.0
x
t

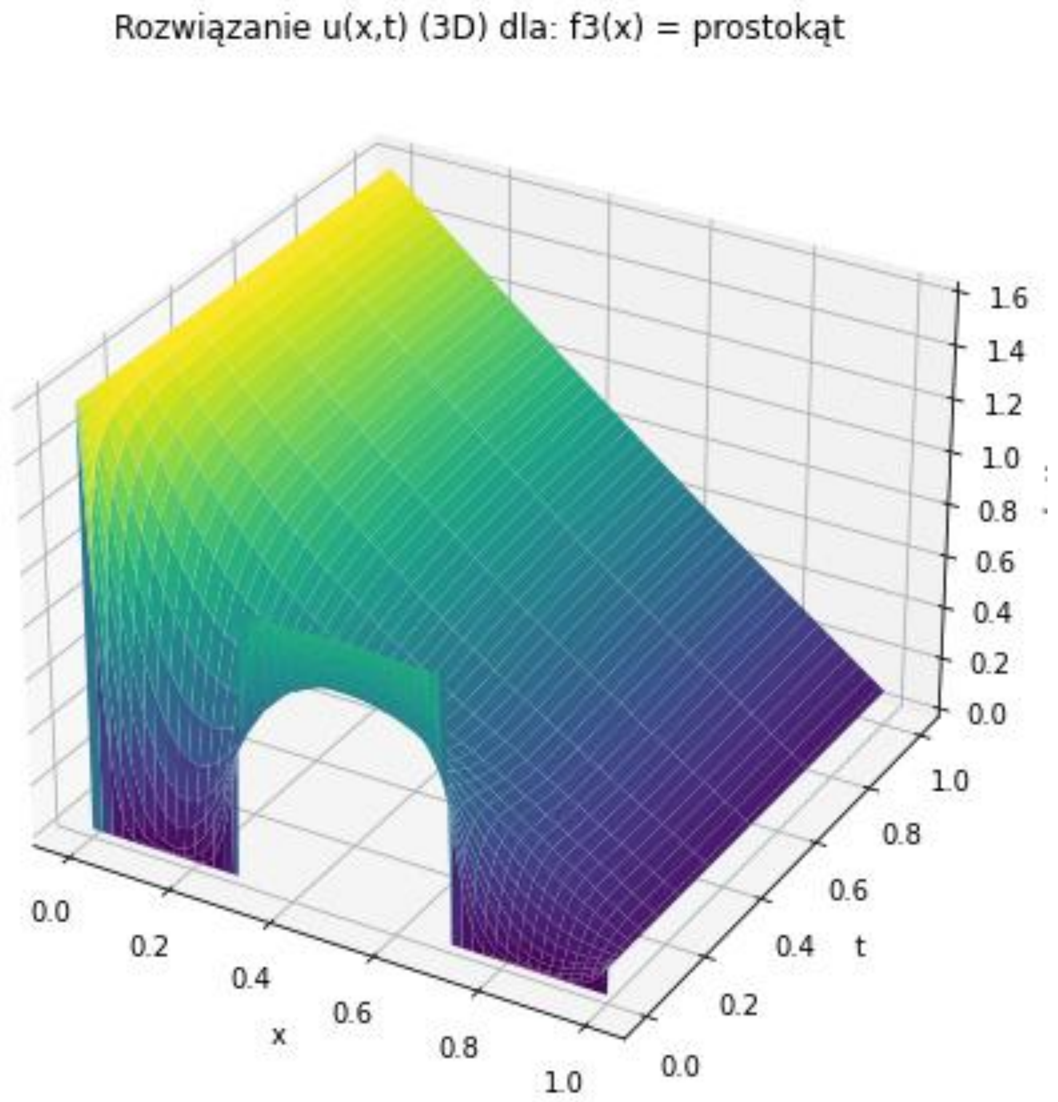

Ewolucja rozwiązania w czasie dla f3(x) - funkcja prostokątna
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000

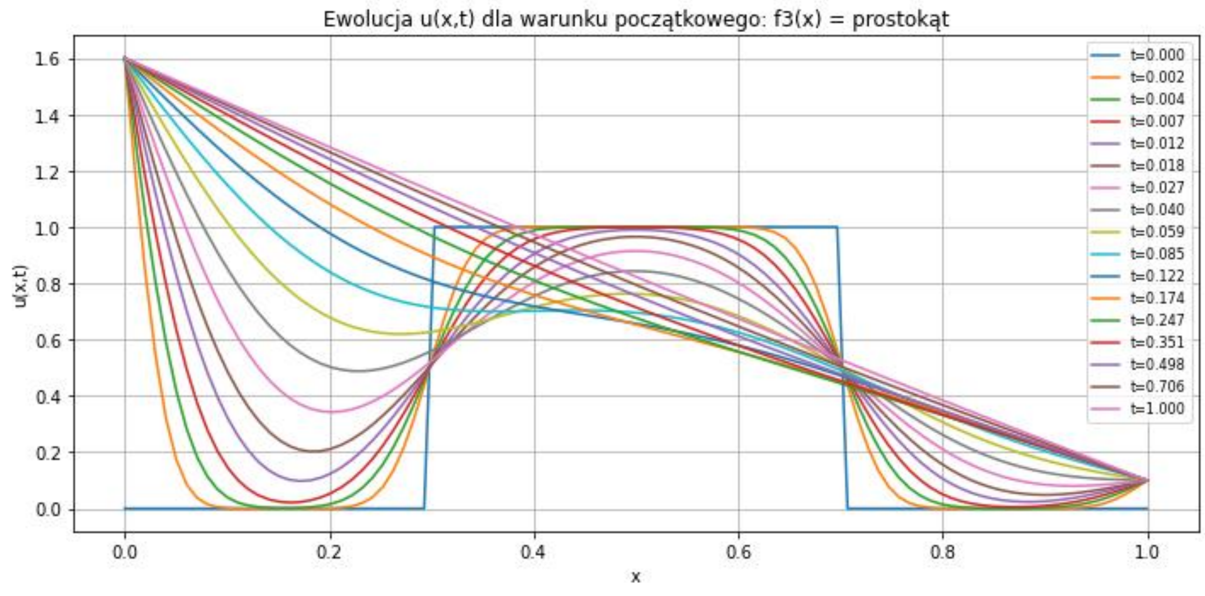

Ewolucja rozwiązania w czasie dla f2(x)
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000

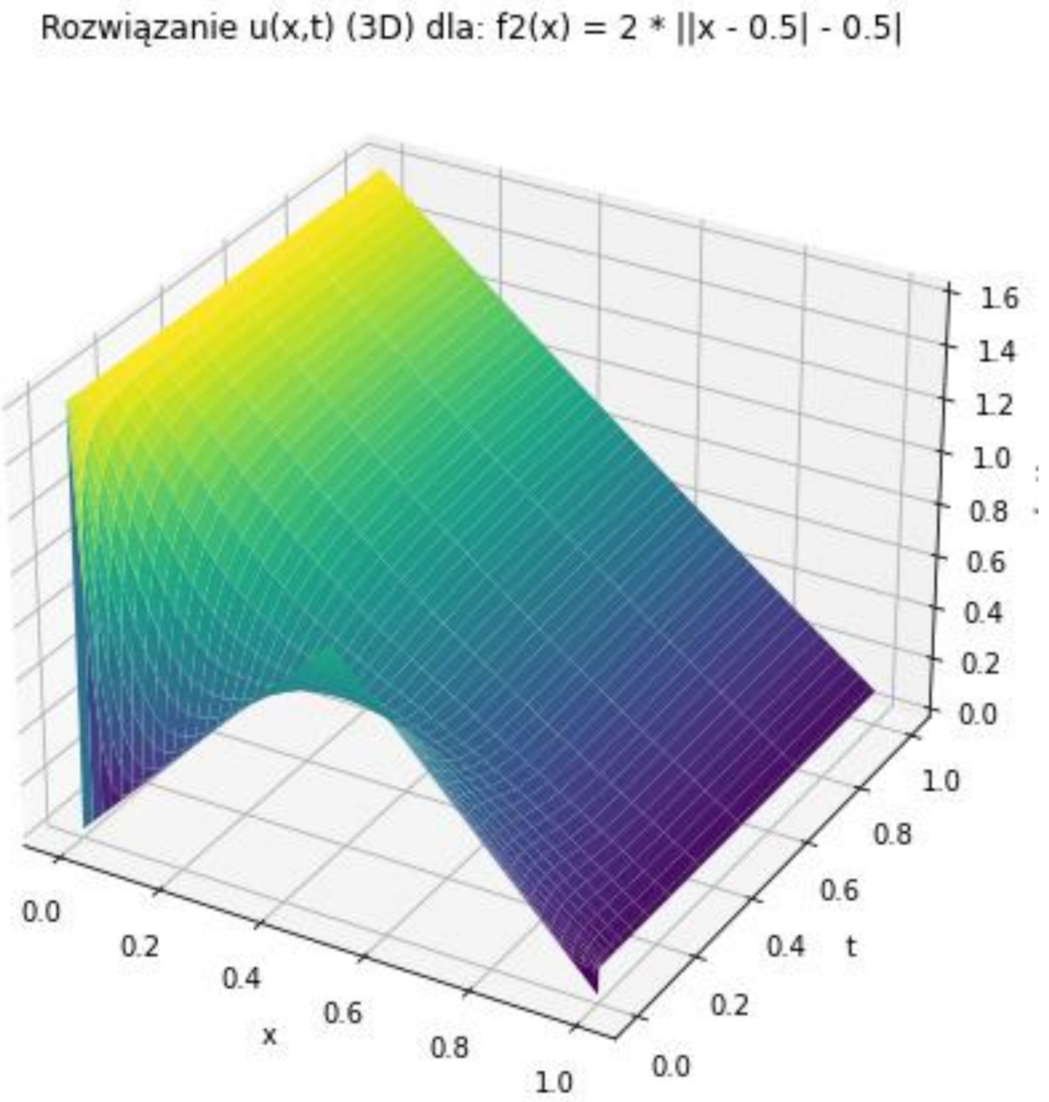

Ewolucja rozwiązania w czasie dla f1(x)
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000

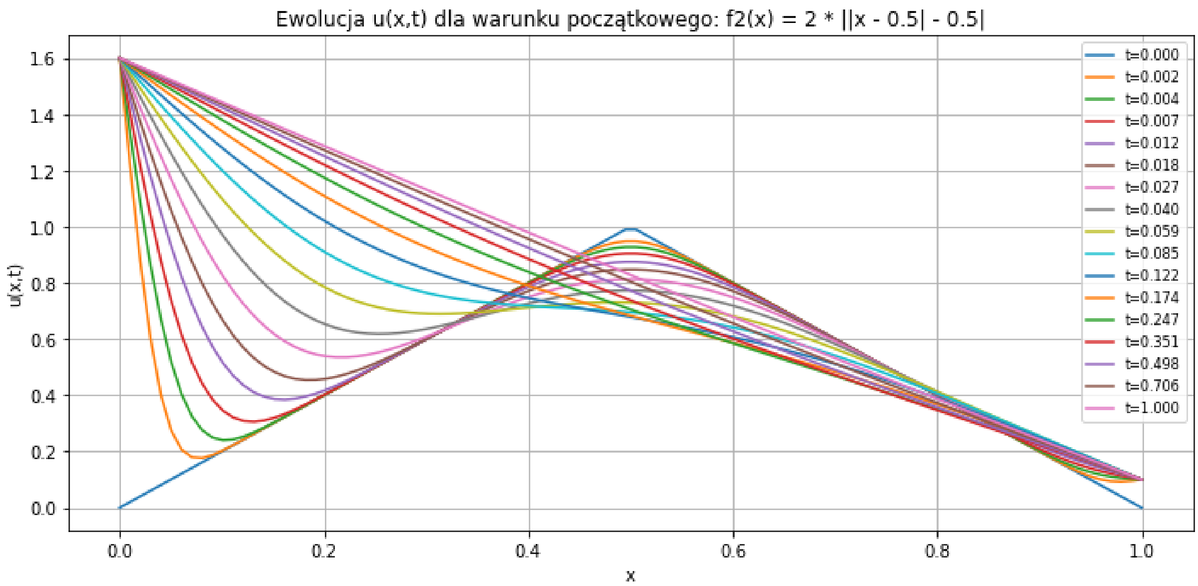
Zad3.

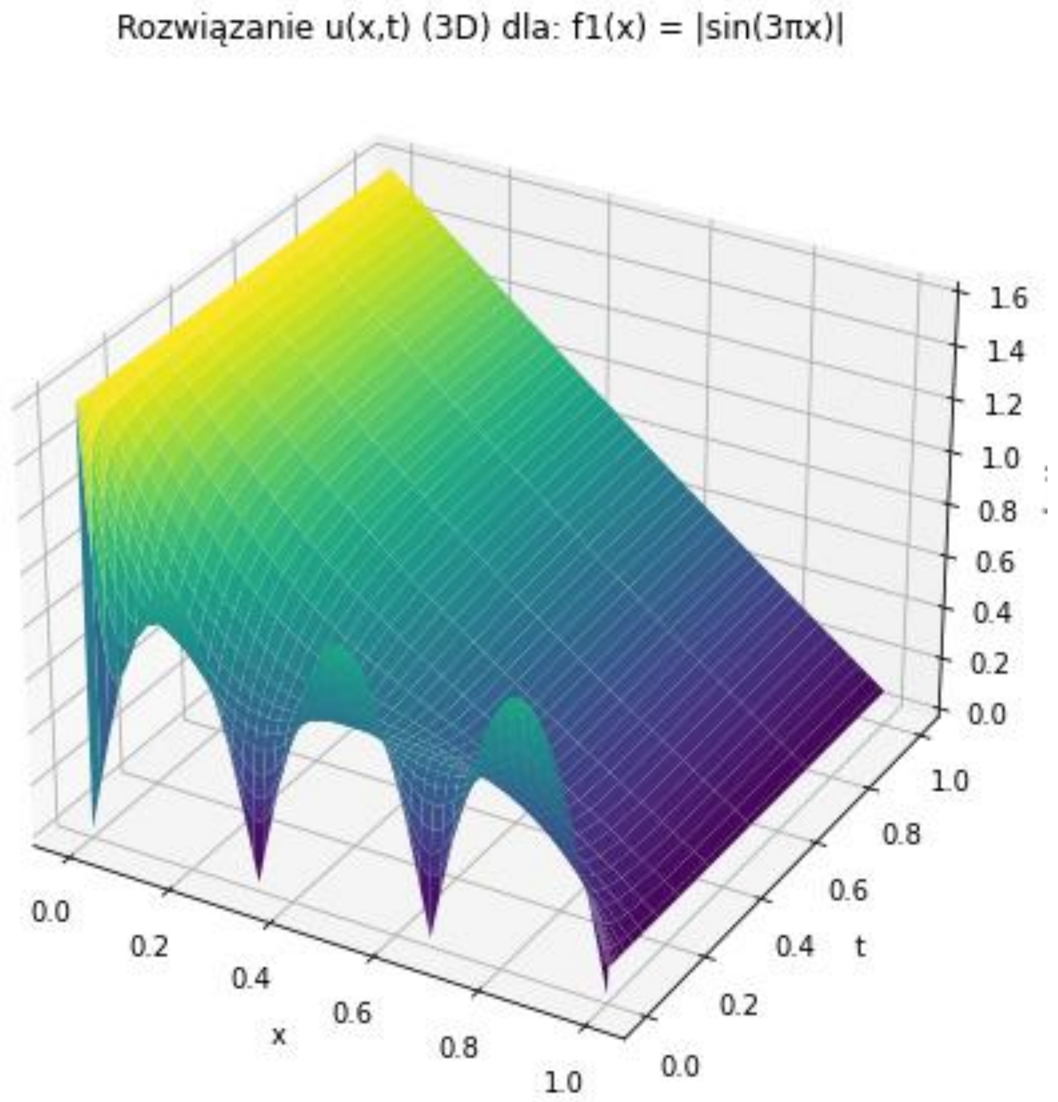
```python
import numpy as np
import matplotlib.pyplot as plt

# --- Parametry ---
L = 1.0        # Długość rury
D = 0.25       # Współczynnik dyfuzji
Nx = 100
dx = L / Nx
dt = 0.0001
T_max = 1.0
Nt = int(T_max / dt)

# Warunki brzegowe
C1 = 0.6
C2 = 0.1
x = np.linspace(0, L, Nx)

# --- Funkcje początkowe ---
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 > 0.3) & (x < 0.7), 1, 0)

initial_conditions = [(f1, "f1(x) = |sin(3πx)|"),
                      (f2, "f2(x) = 2 * ||x - 0.5| - 0.5|"),
                      (f3, "f3(x) = prostokąt")]

# --- Rozwiązanie stacjonarne ---
def v(x, C1, C2, L):
    A = C1
    B = (C2 - C1) / L
    return A + B * x

# --- Warunki brzegowe ---
def apply_inhomogeneous_bc(u):
    u[0] = 1.0
    u[-1] = 0.0
    return u

# --- Solver dyfuzji ---
def solve_diffusion(w_initial, D, dx, dt, t):
    Nt = int(t / dt)
    w = np.copy(w_initial)
    r = D * dt / dx**2

    if r > 0.5:
        raise ValueError(f"Unstable: Courant number r = {r} exceeds 0.5.")

    for _ in range(Nt):
        w_new = np.copy(w)
        w_new[1:-1] = w[1:-1] + r * (w[2:] - 2 * w[1:-1] + w[:-2])
        w = apply_inhomogeneous_bc(w_new)
    return w

# --- Wartości czasu ---
t_vals = [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]

# --- Pętla po funkcjach początkowych ---
for fx, label in initial_conditions:
    w_initial = fx(x) - v(x, C1, C2, L)
    solutions = []

    for t in t_vals:
        w_t = solve_diffusion(w_initial, D, dx, dt, t)
        u_t = v(x, C1, C2, L) + w_t
        solutions.append(u_t)
```

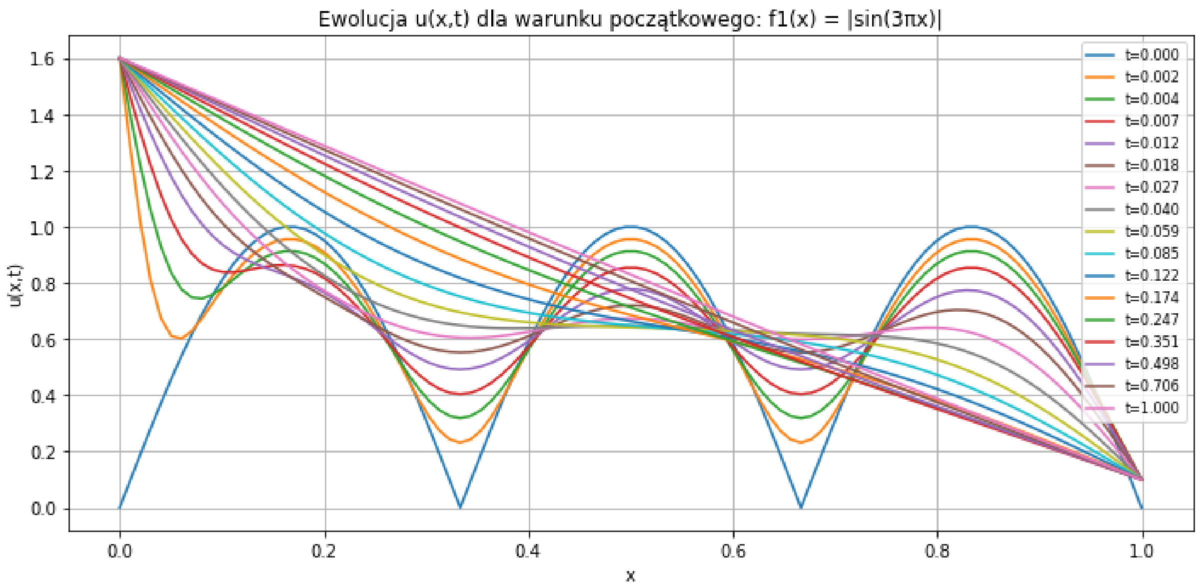

Rozwiązanie u(x,t) (3D) dla: f3(x) = prostokąt
x
t
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6


Ewolucja u(x,t) dla warunku początkowego: f3(x) = prostokąt
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


Rozwiązanie u(x,t) (3D) dla: f2(x) = 2 * ||x - 0.5| - 0.5|
x
t
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

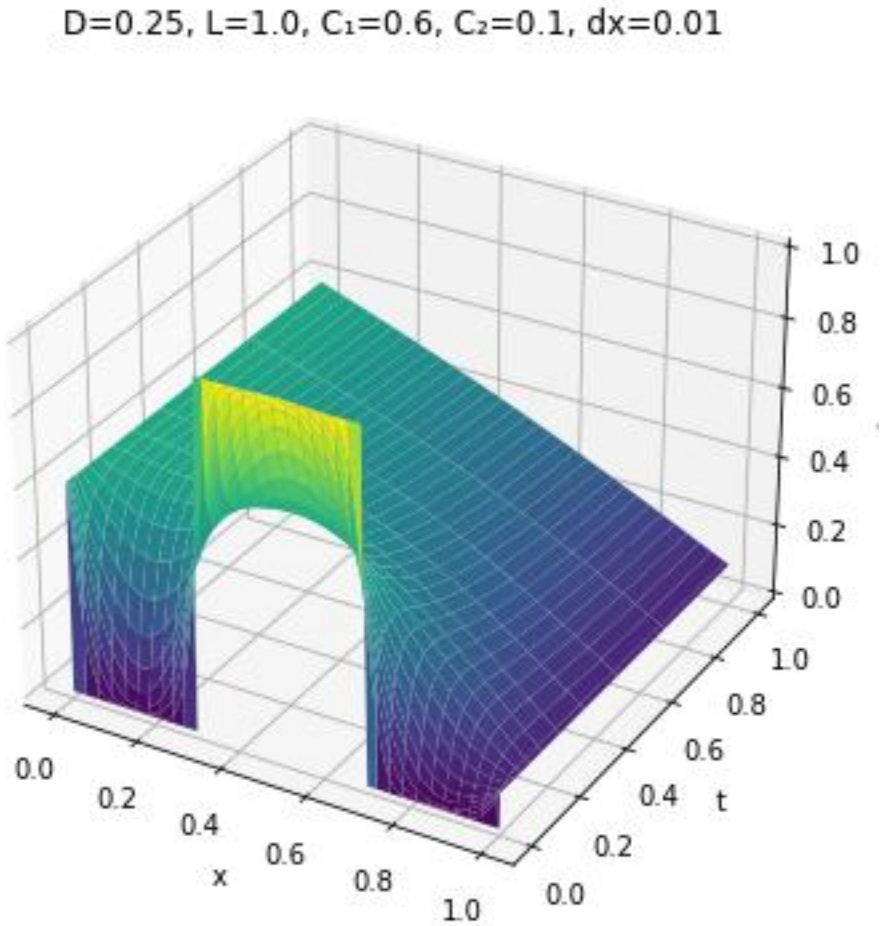

Ewolucja u(x,t) dla warunku początkowego: f2(x) = 2 * ||x - 0.5| - 0.5|
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000

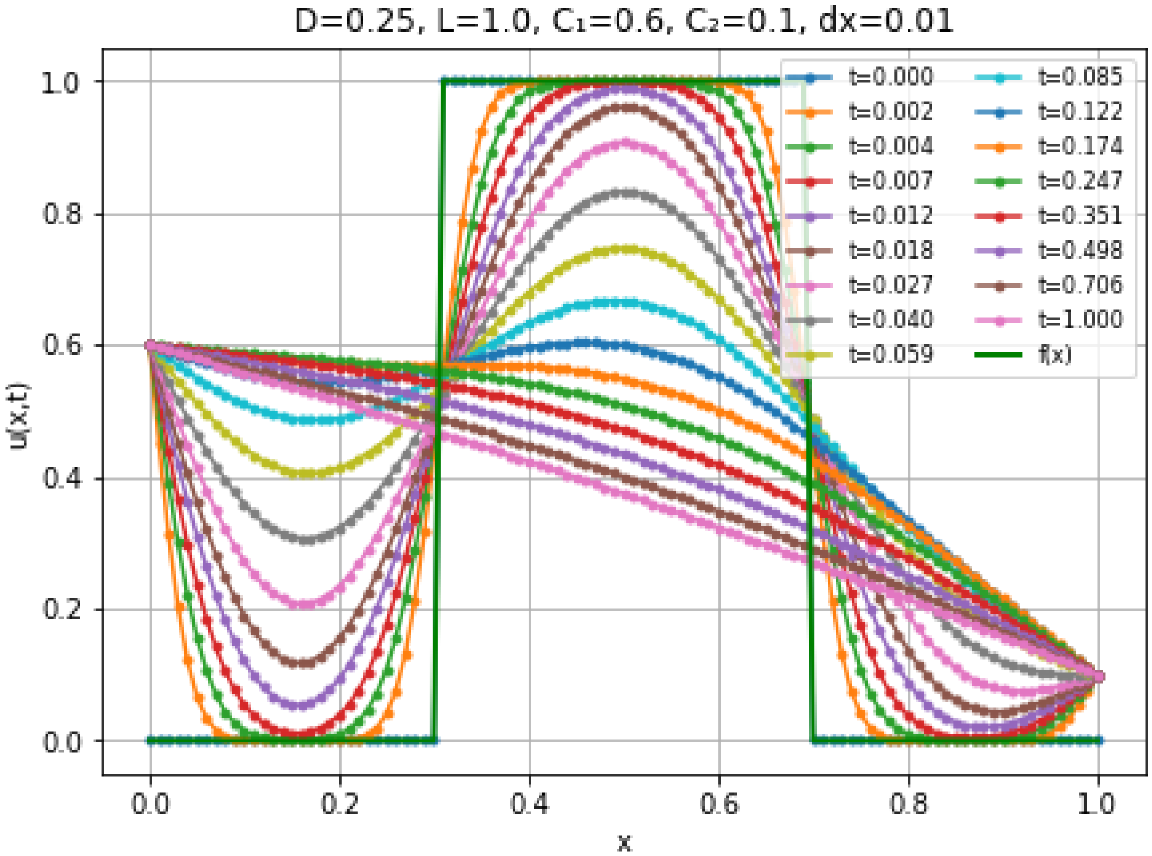

Rozwiązanie u(x,t) (3D) dla: f1(x) = |sin(3πx)|
x
t

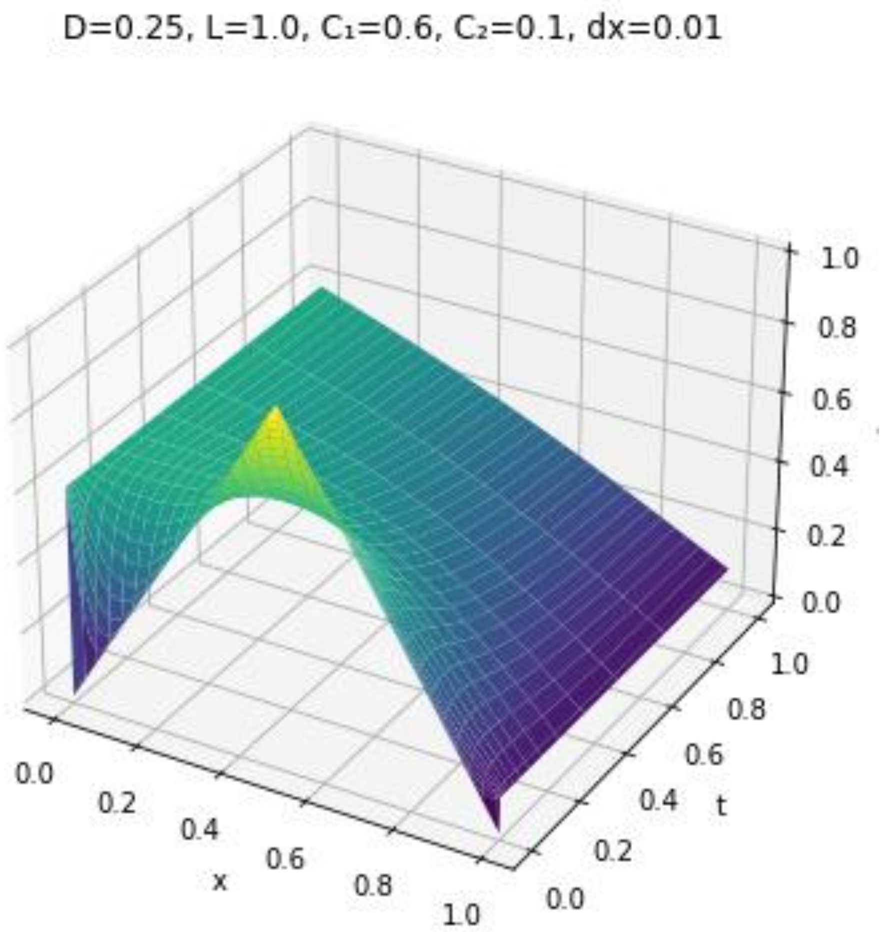

Ewolucja u(x,t) dla warunku początkowego: f1(x) = |sin(3πx)|
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000

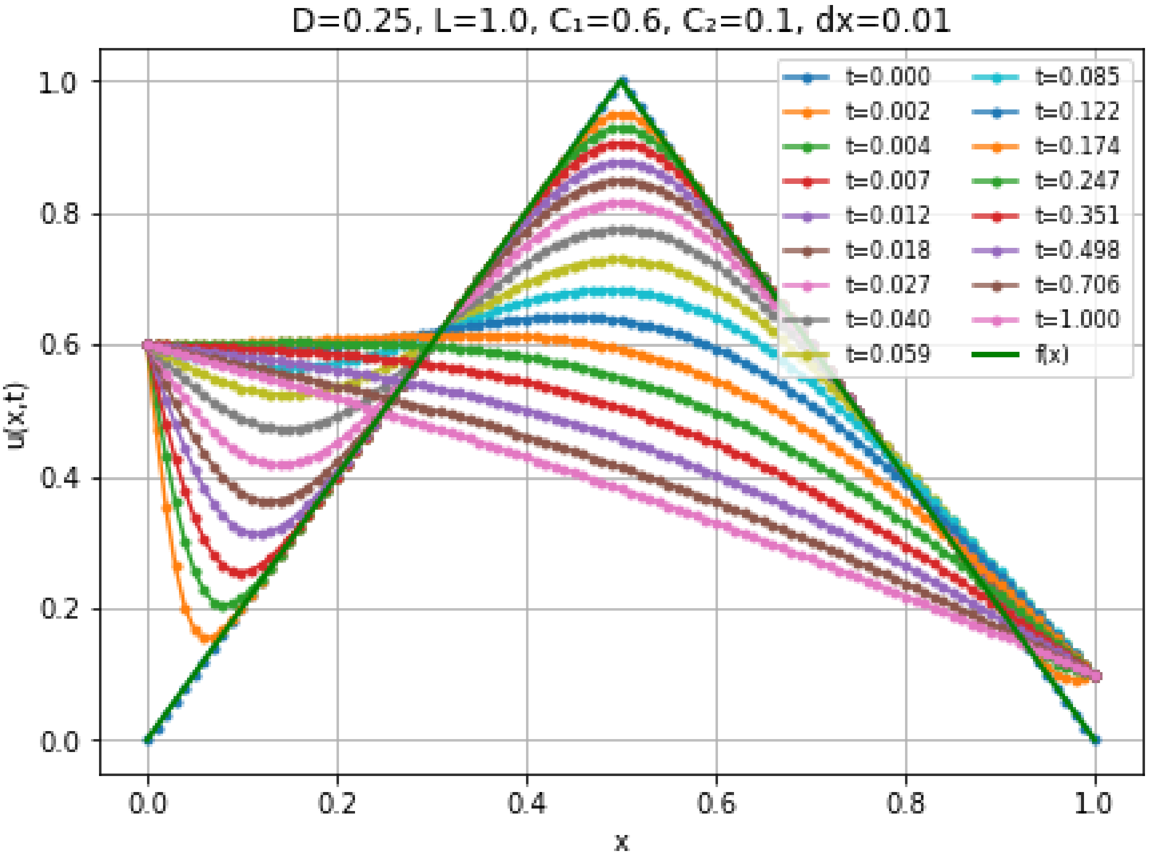
Zad4.

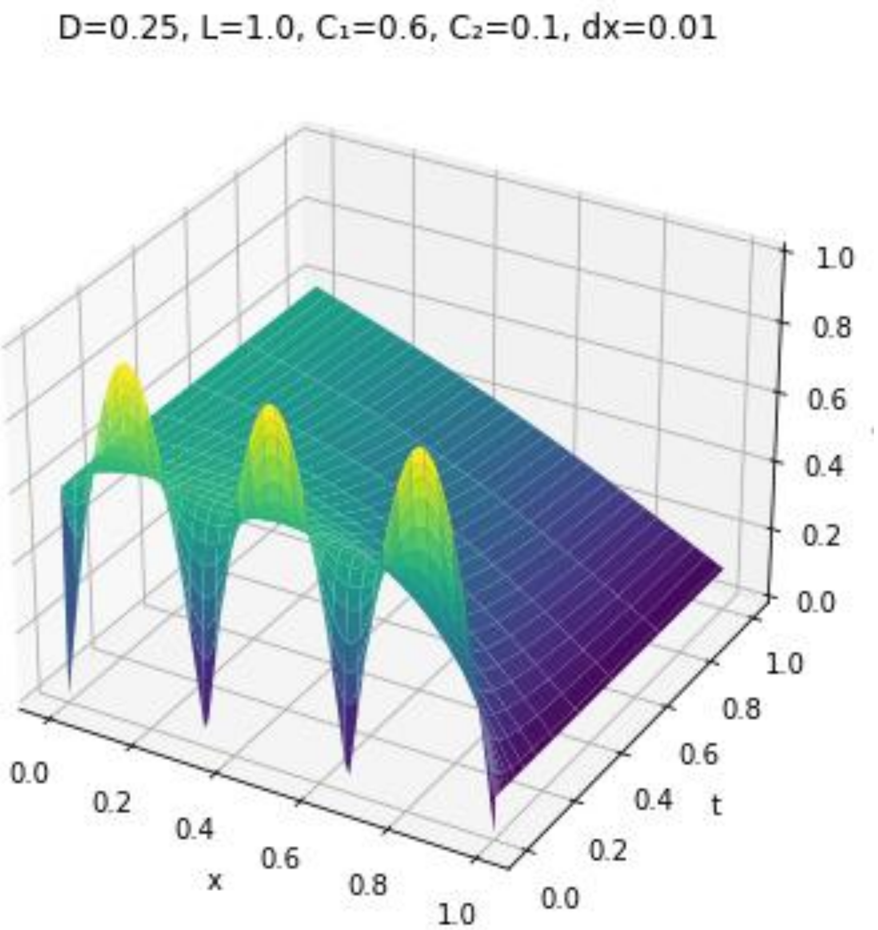
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Parametry
L = 1.0
D = 0.25
Nx = 100
dx = L / Nx
x = np.linspace(0, L, Nx + 1)

C1 = 0.6
C2 = 0.1
A = C1
B = (C2 - C1) / L

# Warunek początkowy f(x)
def f(x):
    return np.sin(5 * np.pi * x) + 1  # przykład jak na wykresie

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

def f2(x):  # Druga funkcja początkowa
    return 2 * np.abs(np.abs(x - L / 2) - L / 2)
def f3(x):  # Trzecia funkcja - prostokątna
    return np.where((x > 0.3) & (x < 0.7), 1, 0)
# V(x) - rozwiązanie stacjonarne
def v(x):
    return A + B * x

# W(x,0)
def w0_1(x):
    return f1(x) - v(x)
def w0_2(x):
    return f2(x) - v(x)
def w0_3(x):
    return f3(x) - v(x)
# Rozwiązanie równania dyfuzji dla jednorodnych brzegów
def solve_diffusion(u0, D, dx, dt, t_max):
    Nx = len(u0) - 1
    Nt = int(t_max / dt)
    alpha = D * dt / dx**2

    u = u0.copy()
    for _ in range(Nt):
        u_new = u.copy()
        u_new[1:-1] = u[1:-1] + alpha * (u[2:] - 2*u[1:-1] + u[:-2])
        u_new[0] = 0
        u_new[-1] = 0
        u = u_new
    return u

# Ewolucja czasowa
dt = 0.00005
t_vals = [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]

w_solutions = np.array([solve_diffusion(w0_1(x), D, dx, dt, t) for t in t_vals])
u_solutions = w_solutions + v(x)  # dodajemy funkcję v(x)
w_solutions1 = np.array([solve_diffusion(w0_2(x), D, dx, dt, t) for t in t_vals])
u_solutions1 = w_solutions1 + v(x)  # dodajemy funkcję v(x)
w_solutions2 = np.array([solve_diffusion(w0_3(x), D, dx, dt, t) for t in t_vals])
u_solutions2 = w_solutions2 + v(x)  # dodajemy funkcję v(x)
```

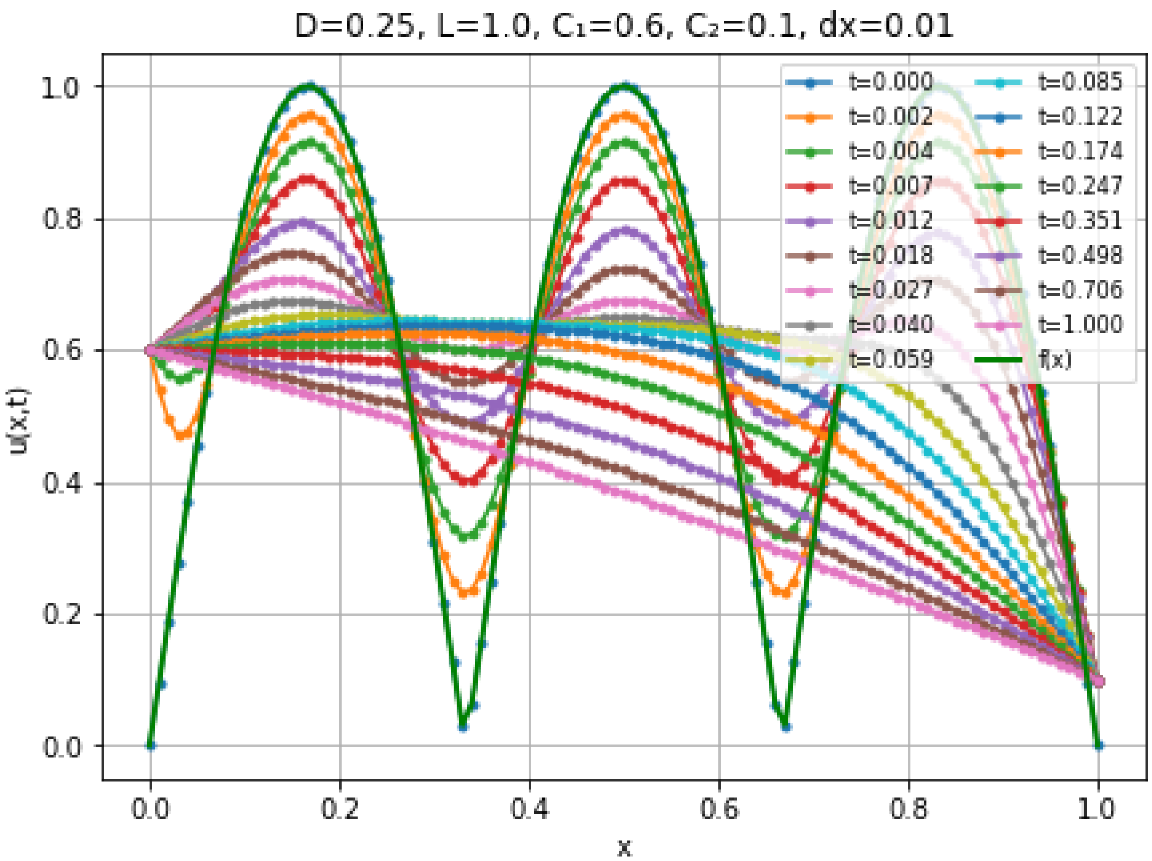

D=0.25, L=1.0, C₁=0.6, C₂=0.1, dx=0.01
x
t


D=0.25, L=1.0, C₁=0.6, C₂=0.1, dx=0.01
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
f(x)


D=0.25, L=1.0, C₁=0.6, C₂=0.1, dx=0.01
x
t
0.0
0.2
0.4
0.6
0.8
1.0

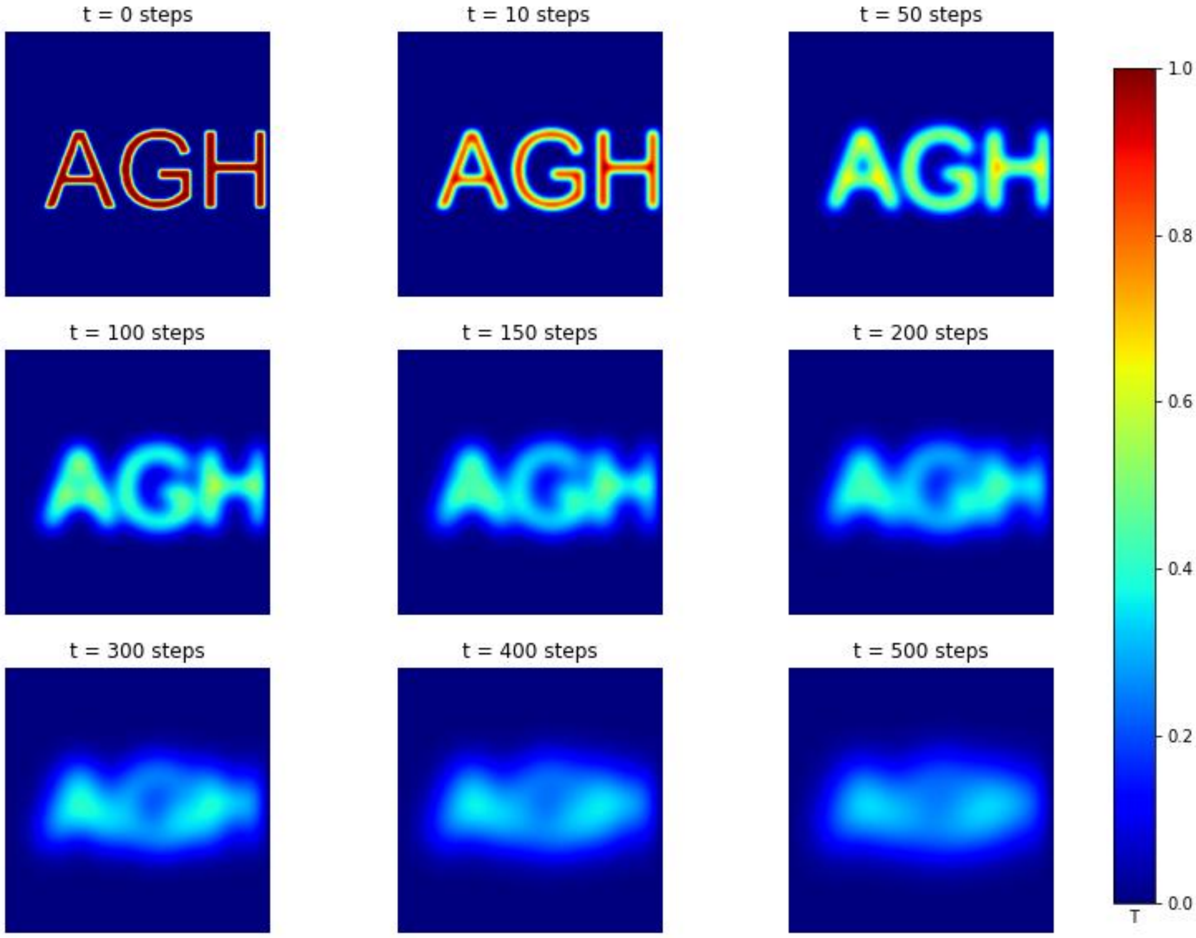

D=0.25, L=1.0, C₁=0.6, C₂=0.1, dx=0.01
u(x,t)
x
t=0.000
t=0.002
t=0.004
t=0.007
t=0.012
t=0.018
t=0.027
t=0.040
t=0.059
t=0.085
t=0.122
t=0.174
t=0.247
t=0.351
t=0.498
t=0.706
t=1.000
f(x)
0.0
0.2
0.4
0.6
0.8
1.0


D=0.25, L=1.0, C₁=0.6, C₂=0.1, dx=0.01
x
t


D=0.25, L=1.0, C₁=0.6, C₂=0.1, dx=0.01
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
f(x)

Zad5.

```python
# Plate size and resolution
w = h = 1.0
nx, ny = 200, 200  # Increased resolution

# Diffusion coefficient
D = 1

# Time step (stability condition)
dx2, dy2 = (1.0 / nx) ** 2, (1.0 / ny) ** 2
dt = dx2 * dy2 / (2 * D * (dx2 + dy2))

# Initial and boundary conditions
Tcool, Thot = 0.0, 1.0
u0 = Tcool * np.ones((nx, ny))

# Generate text mask
text = "AGH"
img = Image.new('L', (nx, ny), color=0)
draw = ImageDraw.Draw(img)
try:
    font = ImageFont.truetype("arial.ttf", 80)  # Increased font size
except OSError:
    font = ImageFont.load_default()  # Use default font if "arial.ttf" is not found

draw.text((30, 60), text, fill=255, font=font)  # Adjusted position for larger text
mask = np.array(img) > 128

# Apply initial heat distribution
u0[mask] = Thot
u = u0.copy()

def do_timestep(u0, u):
    u[1:-1, 1:-1] = u0[1:-1, 1:-1] + D * dt * (
        (u0[2:, 1:-1] - 2*u0[1:-1, 1:-1] + u0[:-2, 1:-1])/dx2 +
        (u0[1:-1, 2:] - 2*u0[1:-1, 1:-1] + u0[1:-1, :-2])/dy2
    )
    u0[:] = u
    return u0, u

# Number of timesteps
nsteps = 500
save_steps = [0, 10, 50, 100, 150, 200, 300, 400,500]

# Plot results
fig, axes = plt.subplots(3, 3, figsize=(12, 10))
cbar_ax = fig.add_axes([0.92, 0.15, 0.03, 0.7])
fignum = 0

for m in range(nsteps + 1):
    u0, u = do_timestep(u0, u)

    if m in save_steps:
        ax = axes[fignum // 3, fignum % 3]
        im = ax.imshow(gaussian_filter(u, sigma=1), cmap='jet', vmin=Tcool, vmax=Thot)
        ax.set_title(f't = {m} steps')
        ax.set_axis_off()
        fignum += 1

fig.colorbar(im, cax=cbar_ax)

cbar_ax.set_xlabel('T')
# plt.tight_layout()
plt.show()
```


t = 0 steps
t = 10 steps
t = 50 steps
t = 100 steps
t = 150 steps
t = 200 steps
t = 300 steps
t = 400 steps
t = 500 steps
AGH
1.0
0.8
0.6
0.4
0.2
0.0
T