Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»
Кафедра информатики

Отчет по лабораторной работе №6
Метод сеток решения волнового уравнения

Выполнил:
студент гр. 953505
Басенко К. А.

Руководитель:
доцент
Анисимов В. Я.

Минск 2021

Цель выполнения работы:

- изучить метод разностных аппроксимаций для волнового уравнения, составить алгоритмы решения волнового уравнения методом сеток, применимым для организации вычислений на ПЭВМ;
- составить программы волнового уравнения по разработанным алгоритмам;
- выполнить тестовые примеры и проверить правильность работы программ.
- получить численное решение волнового уравнения.

Краткие теоретические сведения

Волновое уравнение

Рассмотрим смешанную задачу для волнового уравнения:

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = f(x, t),\ 0 < x < 1,\ 0 < t \le T, \qquad (2.63)$$

$$u(x,0) = \rho(x),\ u'_t(x,0) = q(x),\ 0 \le x \le 1, \qquad (2.64)$$

$$u(0, t) = 0,\ u(1, t) = 0,\ 0 \le t \le T, \qquad (2.65)$$

где $f(x,t)$, $\rho(x)$, $q(x)$ – заданные достаточно гладкие функции, причем $\rho(0) = \rho(1) = q(0) = q(1) = 0$.

Будем предполагать, что задача (2.63) – (2.65) имеет единственное решение $u(x, t) \in C_4(\overline{D})$, $\overline{D} = \{(x, t):\ 0 \le x \le 1,\quad 0 \le t \le T\}$ – замкнутый прямоугольник.

Разностная схема.

Будем использовать сетки, построенные на замкнутом прямоугольнике $\overline{D}$ в лабораторной работе №14, и соответствующие обозначения сеточных функций. Заменяем в уравнении (2.63) частную производную u''_{tt} приближенно второй разностной производной в направлении t, а частную производную — u''_{xx} второй разностной производной в направлении x и, заменив u на y, приходим к разностному уравнению

$$\frac{y_k^{\nu+1} - 2y_k^{\nu} + y_k^{\nu-1}}{\tau^2} + \Lambda y_k^{\nu} = f_k^{\nu}, \tag{2.66}$$

$k = 1, 2, \ldots, N$-l, $\nu = 1, 2, \ldots, M$-l. Шаблон разностного уравнения (2.66) показан на рис. 2.9.

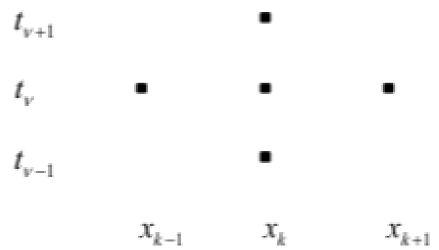


Рис. 2.9

Это уравнение можно разрешить явно относительно $y_k^{\nu+1}$. Но для того, чтобы находить значения разностного решения на $(\nu+1)$-м слое, требуется иметь уже вычисленные значения искомого решения на двух предыдущих слоях. Поэтому нужно получить разностное решение сначала отдельно на слоях, отвечающих значениям $\nu = 0$ и $\nu = 1$. В этом нам помогут начальные условия.

Прежде всего, используя первое начальное условие (2 .64), задаем

$$y_k^0 = \rho_k, \qquad k = 1, 2, \ldots, N-1. \tag{2.67}$$

Кроме того, полагаем при $k = 1, 2, \ldots, N-1$

$$y_k^1 = \rho_k + \tau q_k + \frac{\tau^2}{2}(f_k^0 - \Lambda\rho_k). \tag{2.68}$$

Правая часть формулы (2.68) аппроксимирует многочлен Тейлора $u(x_k,0) + \tau u'_t(x_k,0) + \frac{\tau^2}{2}u''_{tt}(x_k,0)$, поскольку согласно (2.64) $u(x_k,0) = \rho_k$, $u'_t(x_k,0) = q_k$, а из уравнения (2.63) для частных производных решения задачи (2.63) –

(2.65) вытекает связь $u''_{tt}(x_k,0) = f(x_k,0)+u''_{xx}(x_k,0)$. Наконец, согласно краевым условиям (2.65) имеем

$$y_0^\nu = 0,\ y_N^\nu = 0,\ \nu = 0, 1, \ldots, M. \tag{2.69}$$

Теперь разностная схема (2.63) – (2.65) полностью определена. Эта схема явная трехслойная (см. шаблон на рис. 2.7), условно устойчивая в некоторых естественных нормах.

Если $h \to 0$, $\tau \to 0$, причем $\tau/h \le c < 1$, c = const, то решение y разностной схемы (2.66) сходится к рассматриваемому решению u задачи (2.63) – (2.65)в следующем смысле:

$$\|u-y\|_h = O(h^2+\tau^2). \tag{2.70}$$

где $\|u-y\|_h = \max\limits_{0\le\nu\le M}\left(h\sum\limits_{k=1}^{N-1}(u_k^\nu - y_k^\nu)^2\right)^{1/2}$.

Схема (2.66) имеет второй порядок точности и по h, и по τ.

Понятие о методе прямых.

Если в задаче (2.63) – (2.65) ввести дискретность только по x, то мы придем к системе линейных обыкновенных дифференциальных уравнений

$$\frac{d^2 y_k}{dt^2} - \frac{y_{k-1} - 2y_k + y_{k+1}}{h^2} = f(x_k, t), \tag{2.71}$$

где $k = 1, 2, \ldots, N\text{-}1$

с начальными условиями

$$y_k(0) = \rho_k, \quad y'_k(0) = q_k, \tag{2.72}$$

причем $y_0(t) \equiv y_N(t) \equiv 0$.

При сделанном предположении относительно гладкости решения задачи (2.63) – (2.65) имеем

$$\|u - y\|_h = O(h^2), \tag{2.73}$$

где $\|u - y\|_h = \max\limits_{0 \le t \le T} \left(h \sum\limits_{k=1}^{N-1} u(x_k, t) - y_k(t))^2 \right)^{1/2}$, $y_1(t)$, $y_2(t)$, . . . , $y_{N-1}(t)$ – решение задачи Коши (2.63) – (2.65). Данный метод называется *методом прямых*, поскольку приближенное решение задачи (2.63) – (2.65) ищется на прямых

$x = x_k$, $k = 1, 2, \ldots, N\text{-}1$, расположенных в плоскости xt. Разностный же метод часто называется *методом сеток*.

ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ №16

Задача 1

Продольные колебания $u(x,t)$ тяги описываются уравнением

$$\frac{\partial^2 u}{\partial t^2} - \frac{\rho}{E}\frac{\partial^2 u}{\partial x^2} = 0, \qquad 0 < x < L, \tag{2.73}$$

$$u(0,t) = 0, \;\; u(L,t) = 0, \;\; 0 \le t \le T, \tag{2.74}$$

где E –модуль упругости, ρ –плотность материала стержня. Тяга имеет длину L и закреплена на концах. Захватив тягу в центре (см. рис. 2.10),

Рис. 2.10

ее деформируют так, что продольное перемещение становится равным Δu:

$$u(x,0) = -\frac{\rho}{E}\frac{\partial^2 u}{\partial x^2}$$

Затем тяга освобождается.

Рассчитайте колебания $u(x,t)$ при заданных в таблице 2.12 параметрах.

Программная реализация:

Зависимости:

```
import numpy as np
import sympy as sp
from matplotlib import pyplot as plt
```

Начальные данные:

```
L = 0.18
du = 0.002
E = 120e9
rho = 5900

tau = 0.005
h = 0.005
T = 1
c = 0.1
```

```
xs, ts = np.arange(0, L + h, h), np.arange(0, T + tau, tau)
nx, nt = len(xs), len(ts)
```

```
u = np.zeros((nt, nx))

xs_center = xs[1:-1]
gamma = tau / h * c
```

Записываем:

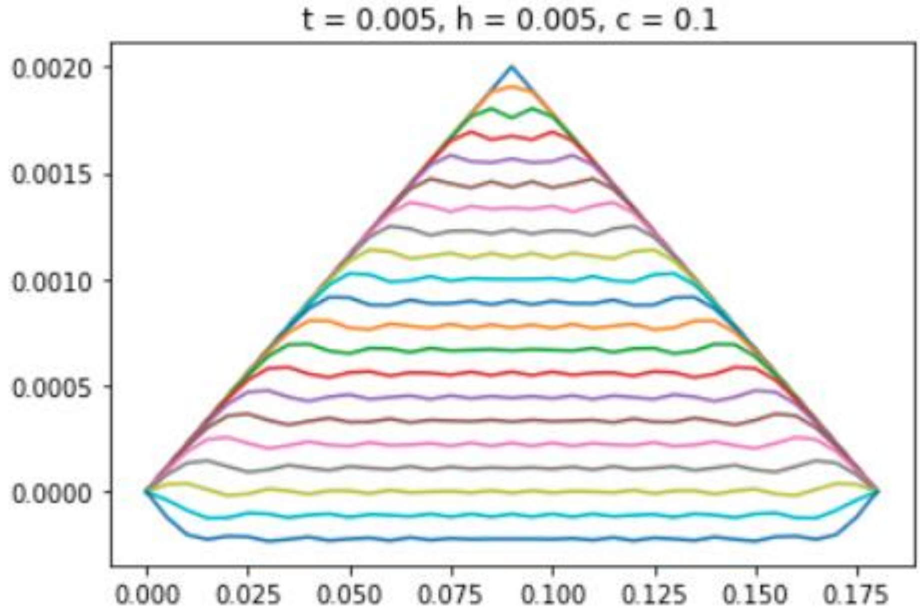
```
for i, x in enumerate(xs):
    coeff = x / L if x <= L / 2 else (1 - x / L)
    u[0][i] = 2 * du * coeff
```

```
for i, x in enumerate(xs_center, 1):
    u[1][i] = u[0][i] + gamma**2 / 2 * (u[0][i + 1] - 2 * u[0][i] + u[0][i - 1])
```

```
for n, t in enumerate(ts[2:], 2):
    for i, x in enumerate(xs_center, 1):
        u[n][i] = 2 * u[n - 1][i] - u[n - 2][i] + gamma**2 * (u[n - 1][i + 1] - 2 * u[n - 1][i] + u[n - 1][i - 1])
```

Фиксируем:

```
for n in range(0, nt, 10):
    plt.title('t = {}, h = {}, c = {}'.format(tau, h, gamma))
    plt.plot(xs, u[n])

plt.show()
```



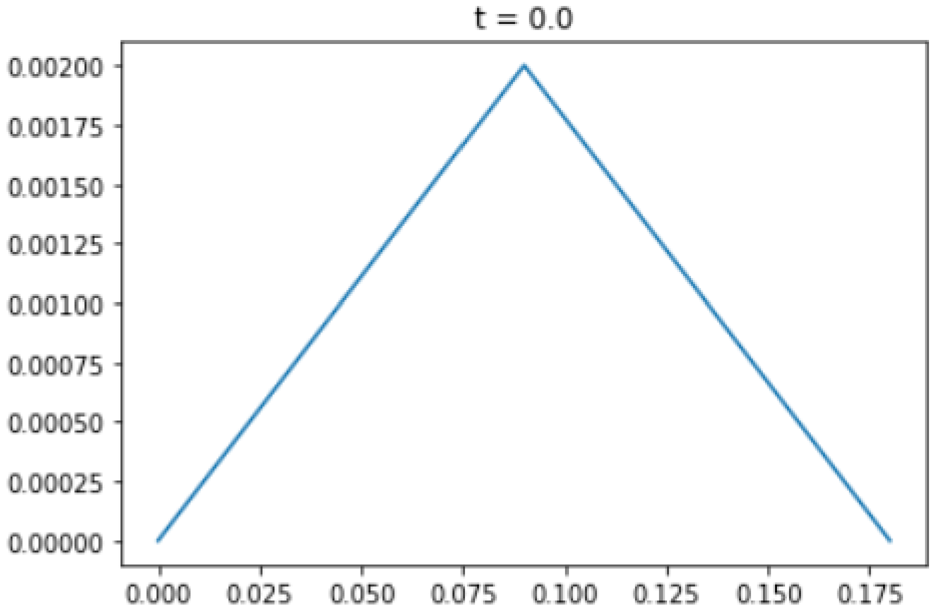
Колебания в разные моменты времени:

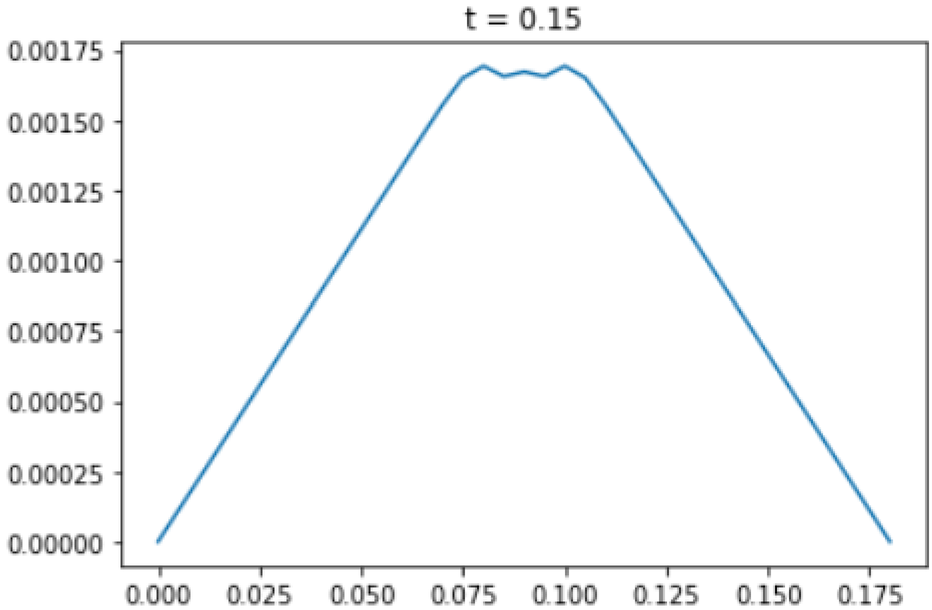
Записываем:

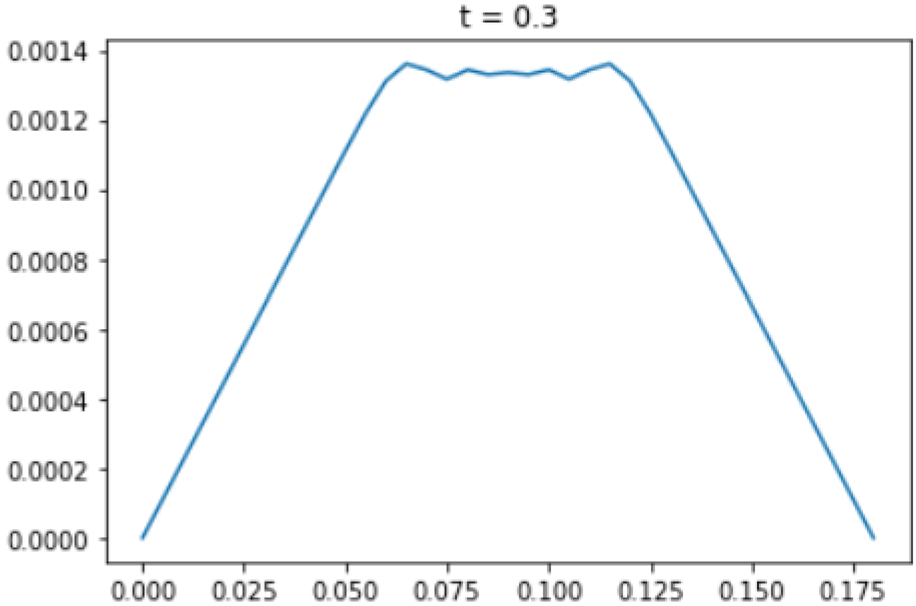
```
from matplotlib import animation

fig = plt.figure()
ax = plt.axes(xlim=(0, L), ylim=(-du, du))
line, = ax.plot([], [], lw=2)

# initialization function: plot the background of each frame
def init():
    line.set_data([], [])
    return line,

# animation function. This is called sequentially
def animate(i):
    line.set_data(xs, u[i])
    return line,

# call the animator. blit=True means only re-draw the parts that have changed.
anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=600, interval=30, blit=True)

plt.show();
```

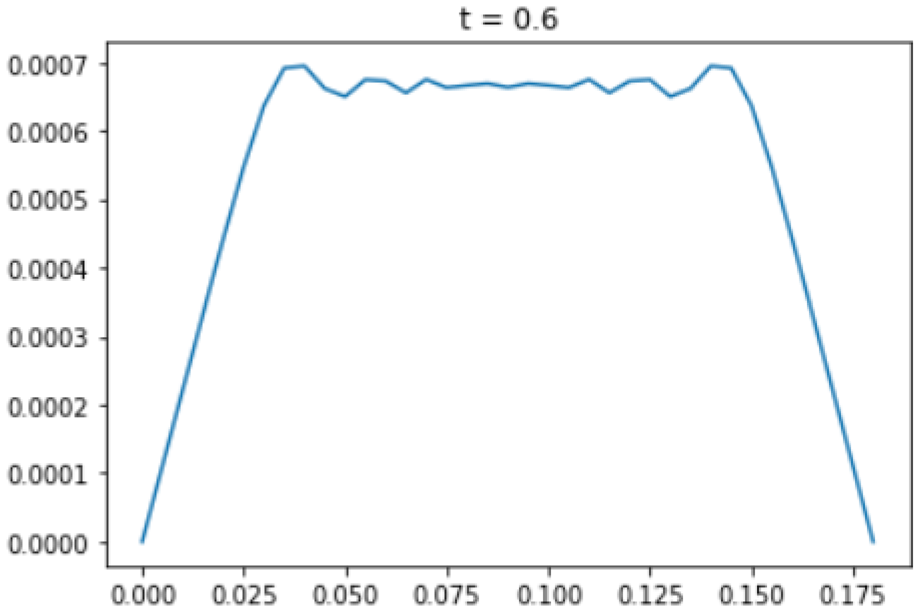
Фиксируем:

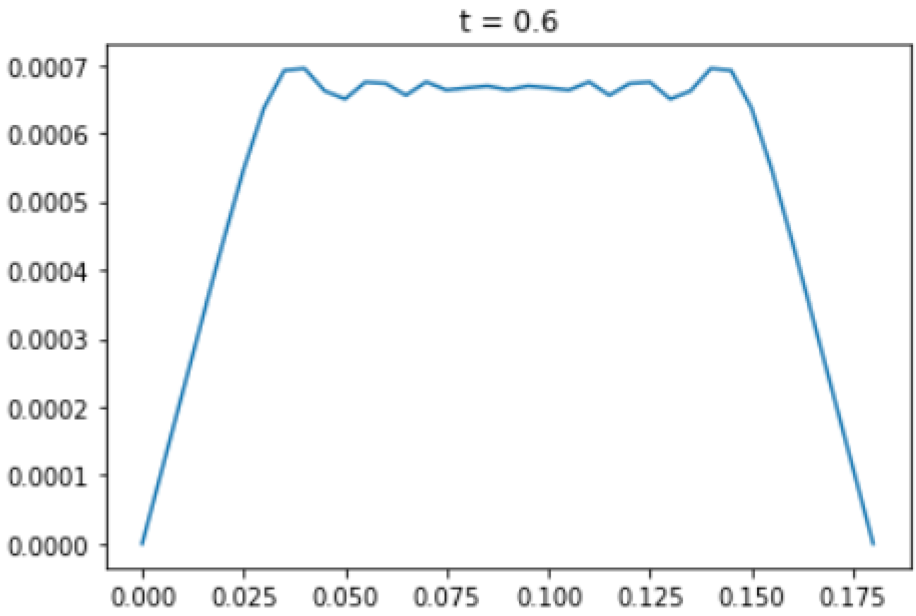
```python
for n in range(0, nt, 30):
    plt.plot(xs, u[n])
    plt.title('t = {}'.format(ts[n]))
    plt.show()
```

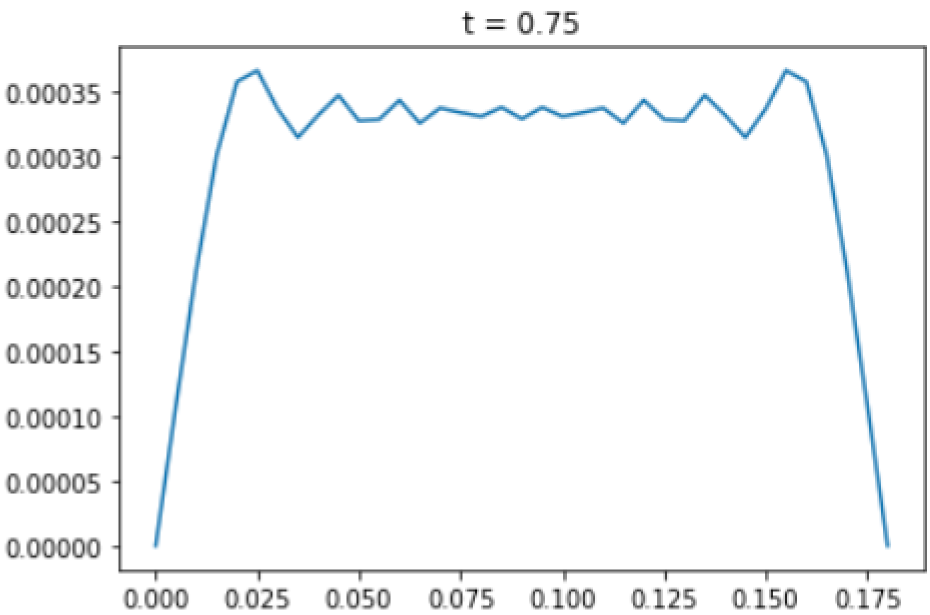


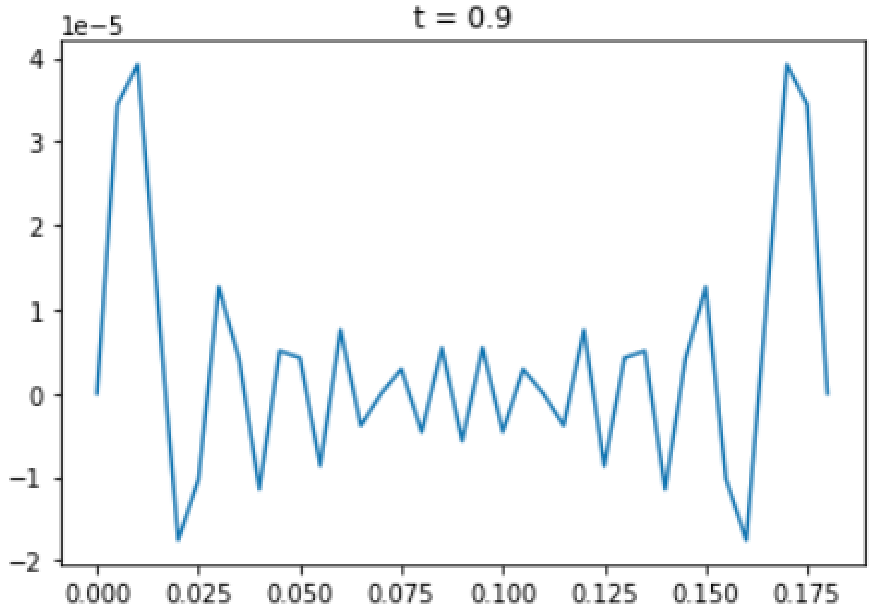



t = 0.3
0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175


t = 0.6
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175


t = 0.6
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175


t = 0.75
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175



Задача 2 Рассчитать колебания тонкой пластины

Колебания тонкой пластины (см. рис.2.11) без учета потерь на трение описываются нормированным волновым уравнением вида

$$\frac{\partial^2 u}{\partial t^2} - \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0$$

где *u(x, y, t)* –деформация пластины, *x,y* –координаты, *t*- время.

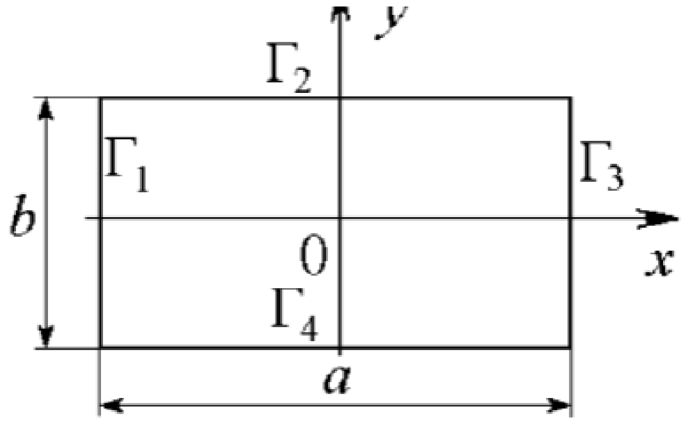


Рис.2.11

Рассчитать колебания пластины при заданных в таблице размерах *a* и *b,*граничных $Г_1, Г_2, Г_3, Г_4$, и начальных $u(x, y, 0)$ и $\frac{\partial\, u(x,y,0)}{\partial t}$ условиях.

Зависимости:

```python
import math
import chart_studio.plotly as py
import plotly.graph_objs as go
import plotly
```

Начальные данные:

```python
a = 2
b = 1

h = 0.1
tau = 0.05
T = 4

gamma = tau / h
```

```python
xs = np.arange(-a / 2, a / 2 + h, h)
ys = np.arange(-b / 2, b / 2 + h, h)
ts = np.arange(0, T + tau, tau)

nx = len(xs)
ny = len(ys)
nt = len(ts)

u = np.zeros((nt, ny, nx))
```

Записываем:

```python
u[0, :] = np.arctan(np.cos(math.pi * xs / a))
```

```python
for i in range(0, ny):
    for j in range(1, nx - 1):
        if i == 0:
            y_coeff = -2 * u[0, i, j] + 2 * u[0, i + 1, j]
        elif i == ny - 1:
            y_coeff = -2 * u[0, i, j] + 2 * u[0, i - 1, j]
        else:
            y_coeff = u[0, i - 1, j] - 2 * u[0, i, j] + u[0, i + 1, j]

        u[1, i, j] = u[0, i, j] + gamma**2 / 2 * (
            u[0, i, j - 1] - 2 * u[0, i, j] + u[0, i, j + 1] + \
            y_coeff
        ) + tau * np.sin(2 * math.pi * xs[j] / a) * np.sin(math.pi * ys[i] / b)
```

```
for t in range(2, nt):
    for i in range(0, ny):
        for j in range(1, nx - 1):
            if i == 0:
                y_coeff = -2 * u[t - 1, i, j] + 2 * u[t - 1, i + 1, j]
            elif i == ny - 1:
                y_coeff = -2 * u[t - 1, i, j] + 2 * u[t - 1, i - 1, j]
            else:
                y_coeff = u[t - 1, i - 1, j] - 2 * u[t - 1, i, j] + u[t - 1, i + 1, j]

            u[t, i, j] = 2 * u[t - 1, i, j] - u[t - 2, i, j] + gamma**2 * (
                u[t - 1, i, j - 1] - 2 * u[t - 1, i, j] + u[t - 1, i, j + 1] + y_coeff)
```

Дописываем:

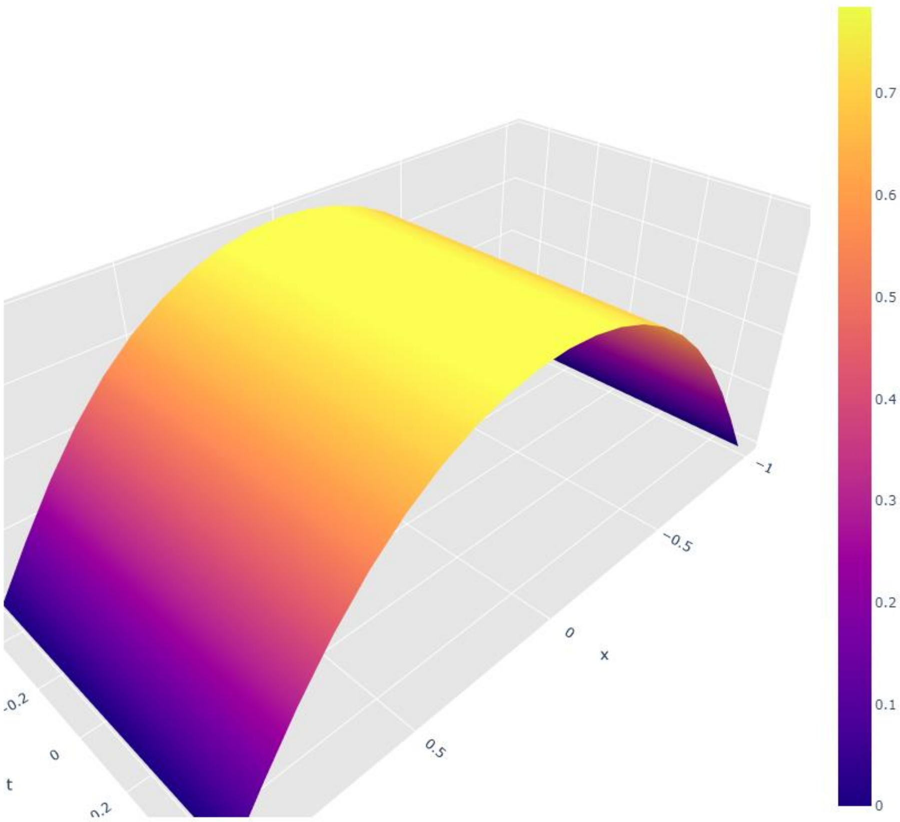
```
def plot_2(u):
    x_grid, y_grid = np.meshgrid(xs, ys)

    surface = go.Surface(x=x_grid, y=y_grid, z=u)
    data = [surface]

    layout = go.Layout(
        title='Parametric Plot',
        scene=dict(
            xaxis=dict(
                gridcolor='rgb(255, 255, 255)',
                showbackground=True,
                backgroundcolor='rgb(230, 230,230)'
            ),
            yaxis=dict(
                title='t',
                gridcolor='rgb(255, 255, 255)',
                showbackground=True,
                backgroundcolor='rgb(230, 230,230)'
            ),
            zaxis=dict(
                title='u(x, t)',
                gridcolor='rgb(255, 255, 255)',
                showbackground=True,
                backgroundcolor='rgb(230, 230,230)'
            )
        )
    )

    fig = go.Figure(data=data, layout=layout)
    plotly.offline.plot(fig, auto_open=True)
```

Фиксируем:

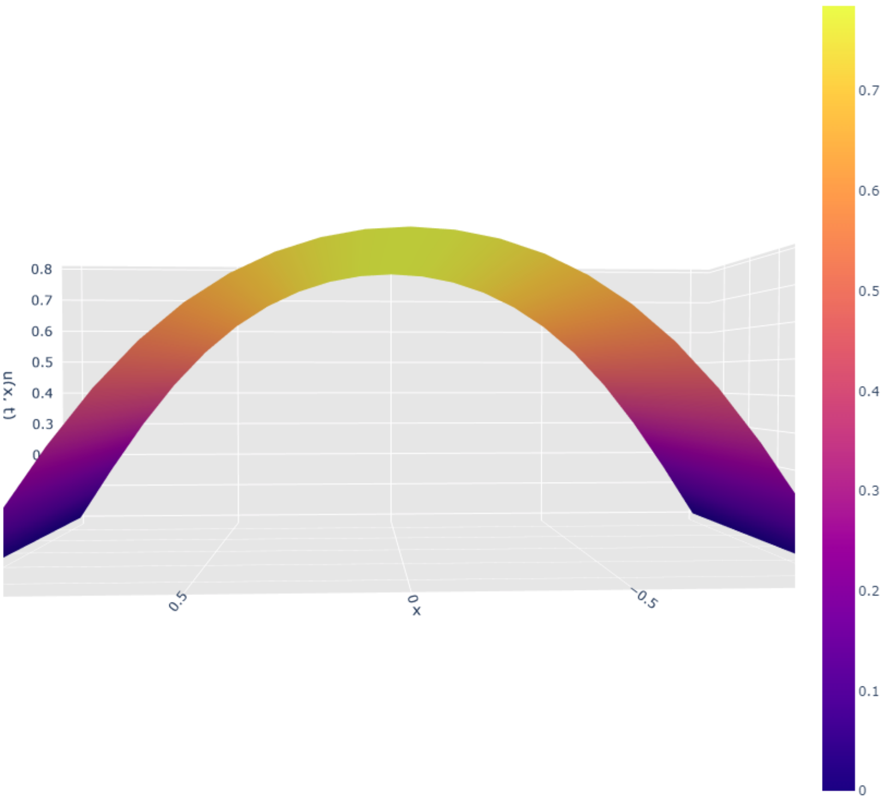
```
import time

for i in range(0, nt, 30):
    plot_2(u[i])
    time.sleep(0.5)
```

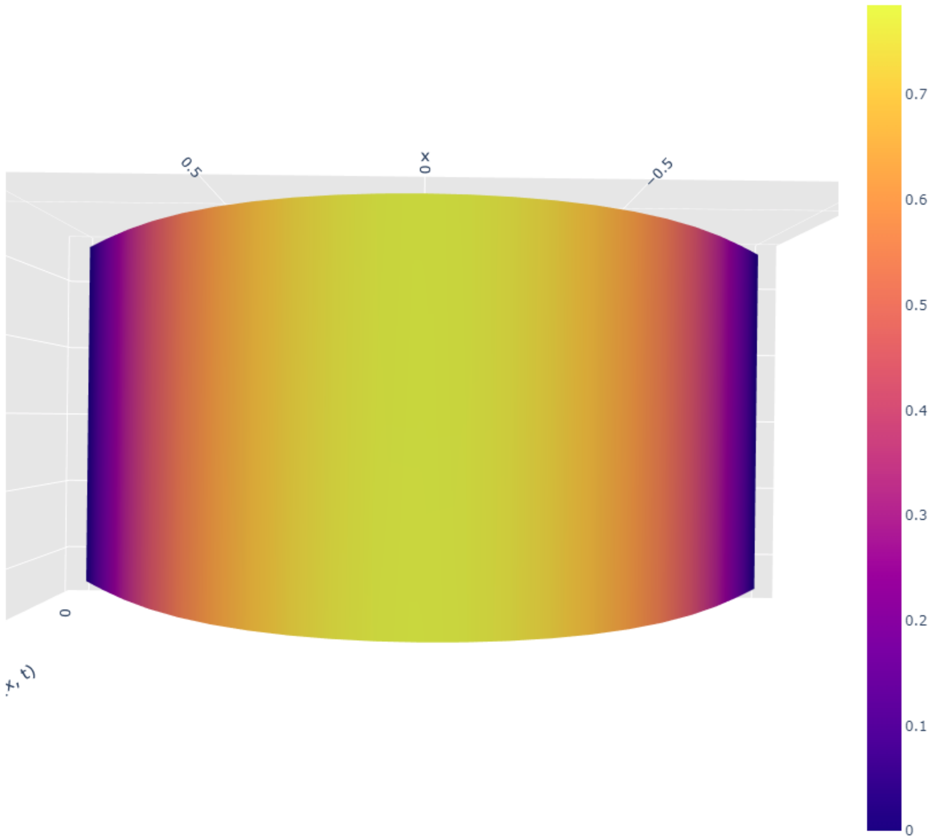
Результаты в разные моменты времени:

Parametric Plot



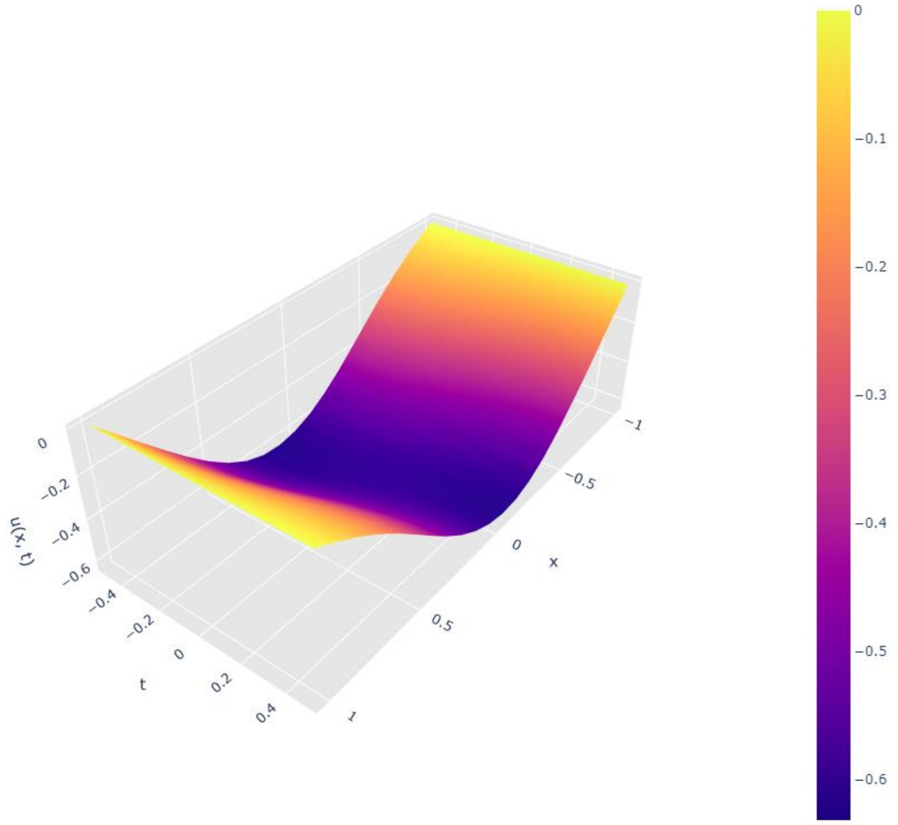
Parametric Plot

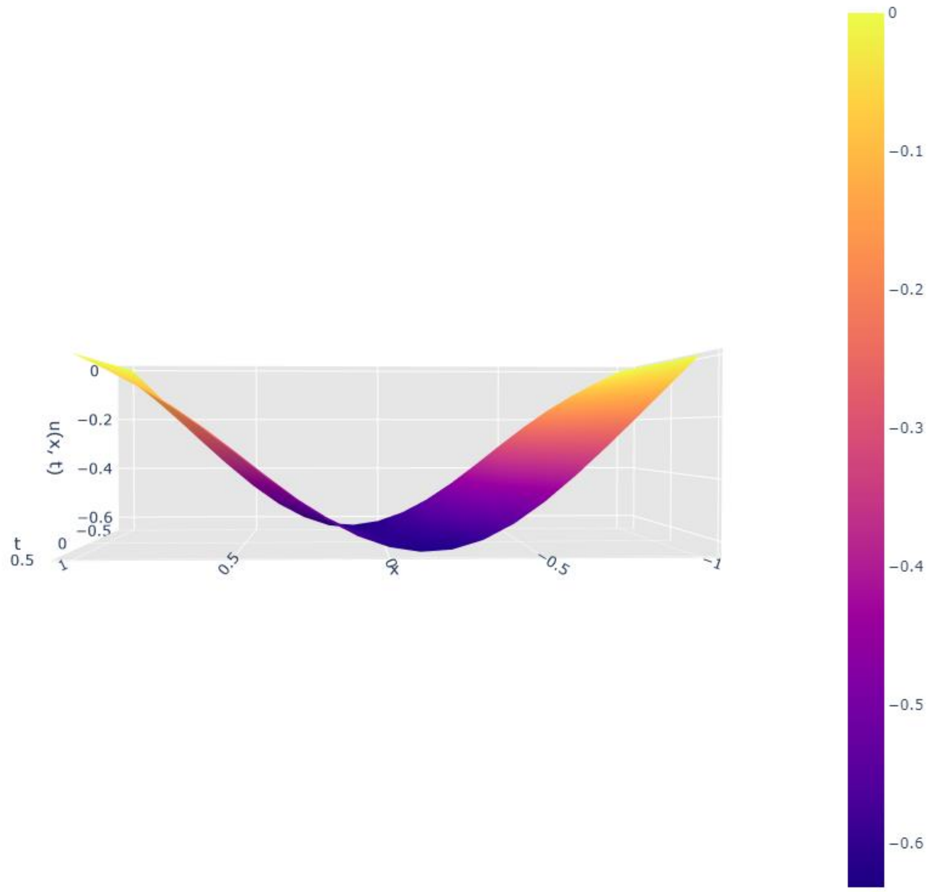




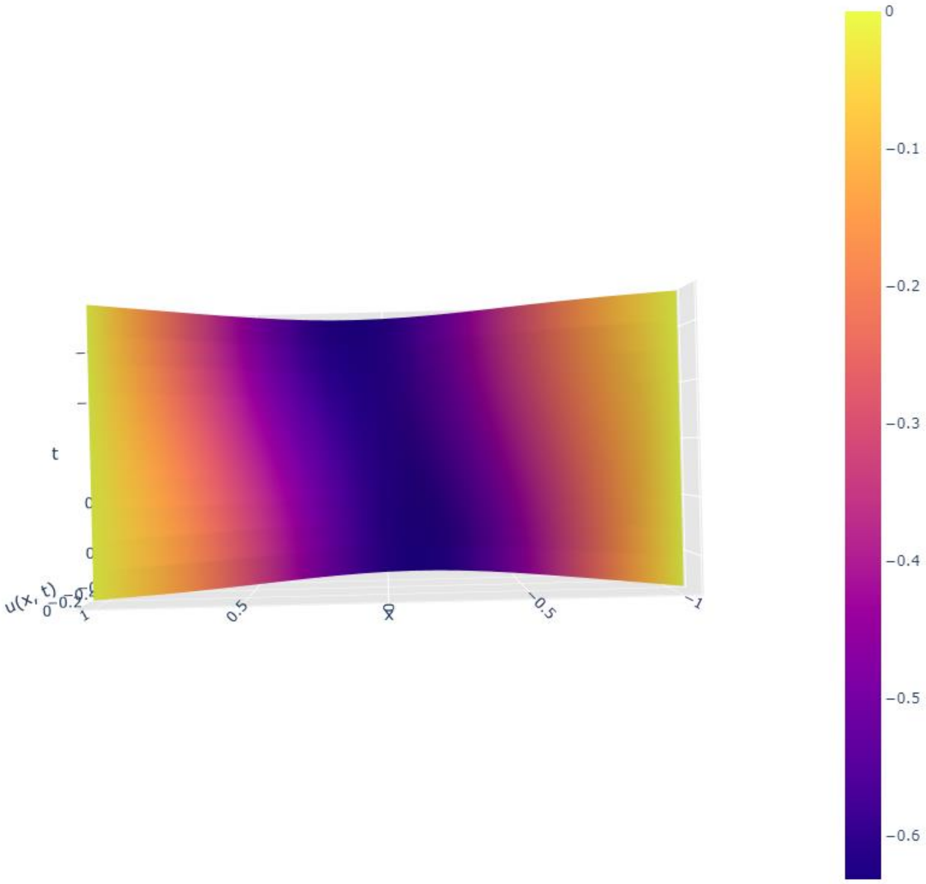
0.5
x
0
−0.5
0
x, t)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Parametric Plot

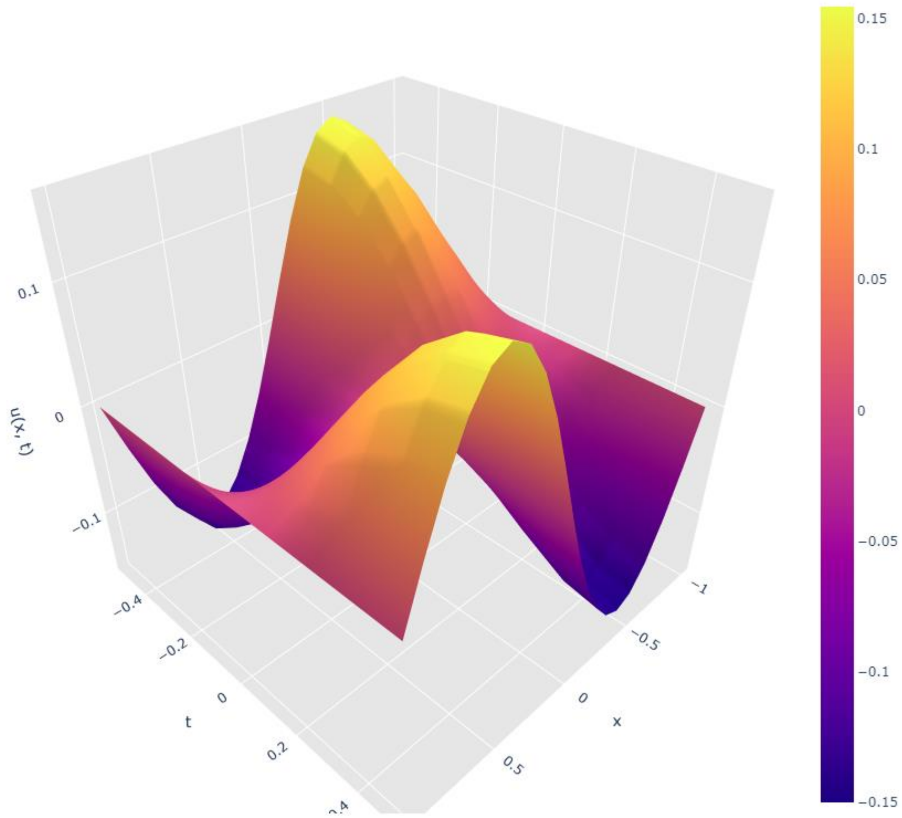




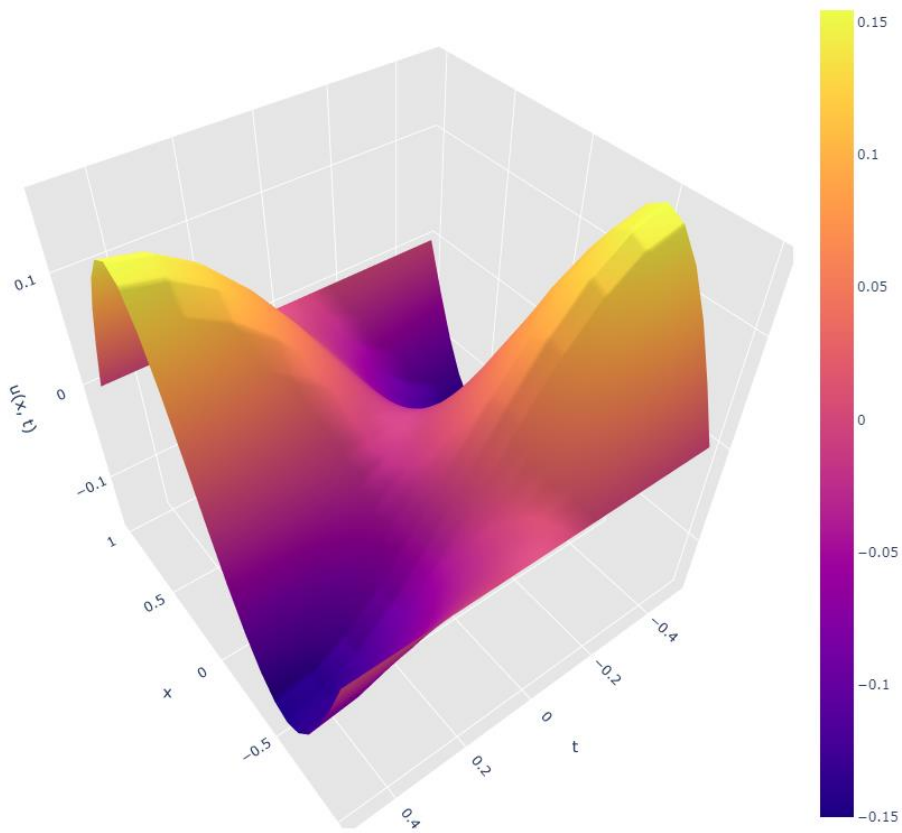
u(x, t)
0
−0.2
−0.4
−0.6
t
−0.5
0
0.5
1
0.5
x
−0.5
−1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6



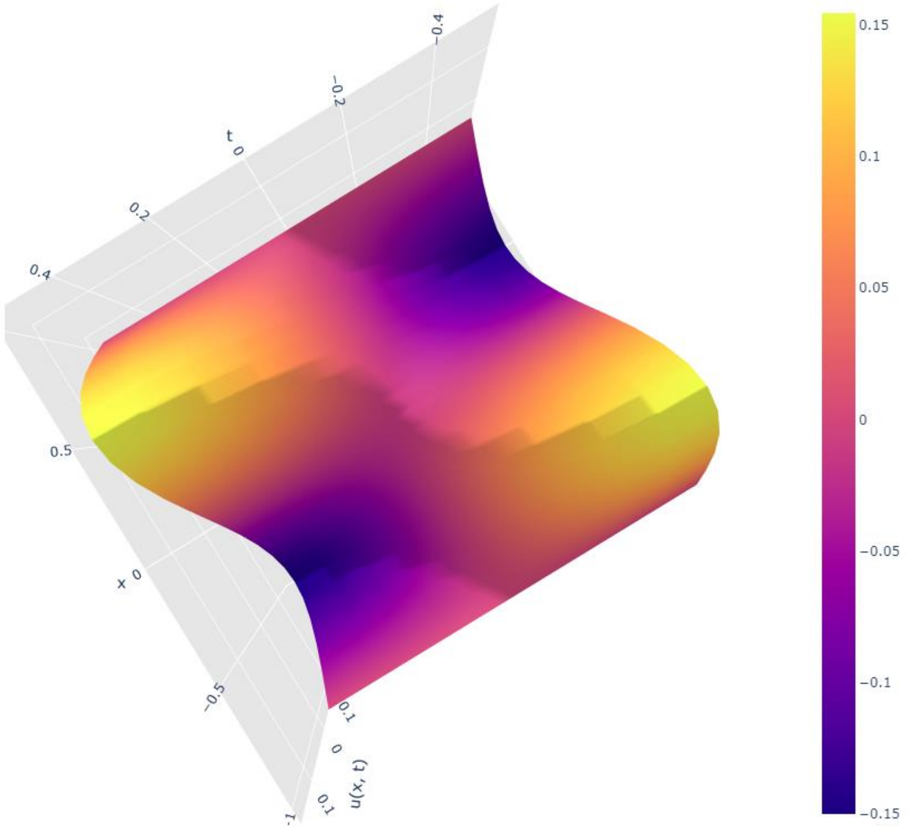
t
u(x, t)
x
0.5
0
−0.5
−1
1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6



u(x, t)
0.1
0
−0.1
t
−0.4
−0.2
0
0.2
x
−1
−0.5
0
0.5
0.15
0.1
0.05
0
−0.05
−0.1
−0.15



u(x, t)
0.1
0
−0.1
x
1
0.5
0
−0.5
t
−0.4
−0.2
0
0.2
0.4
0.15
0.1
0.05
0
−0.05
−0.1
−0.15

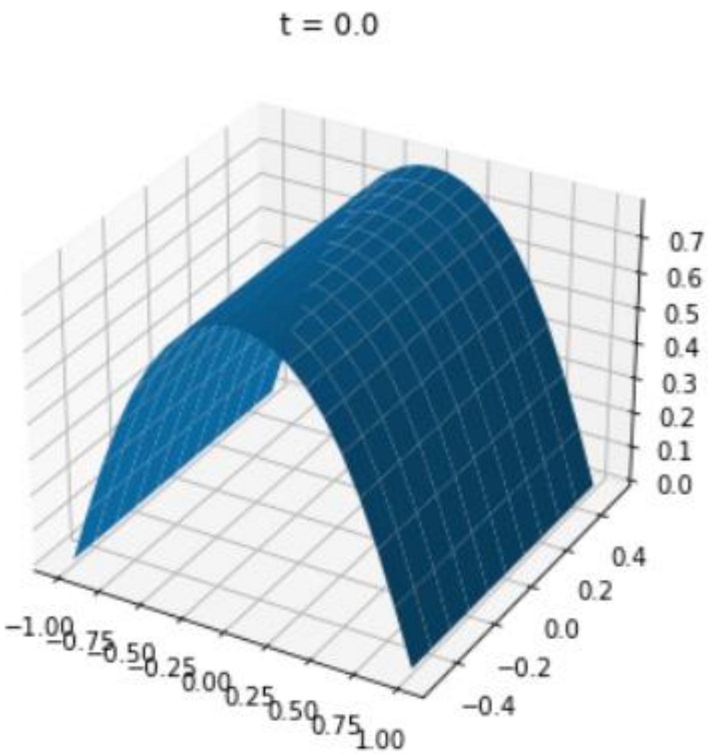


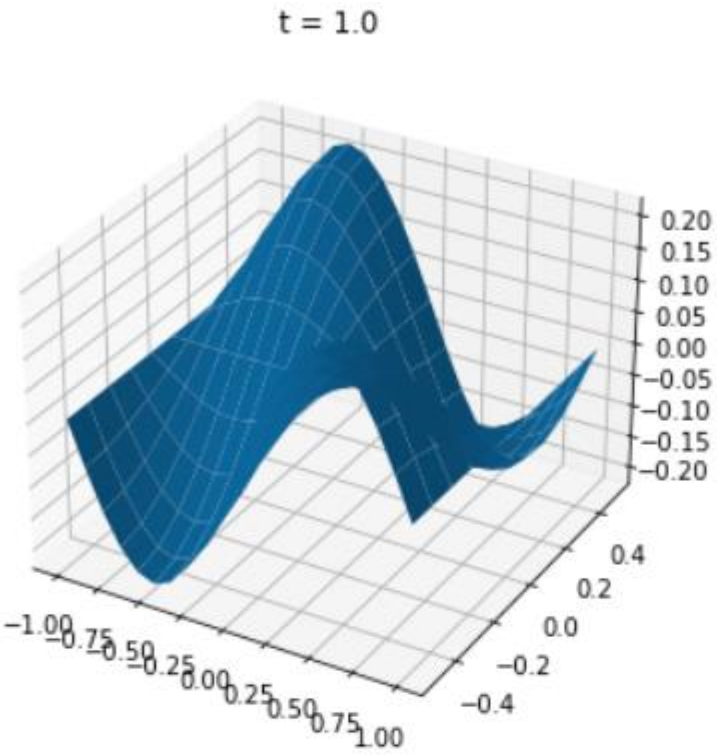
Дописываем:

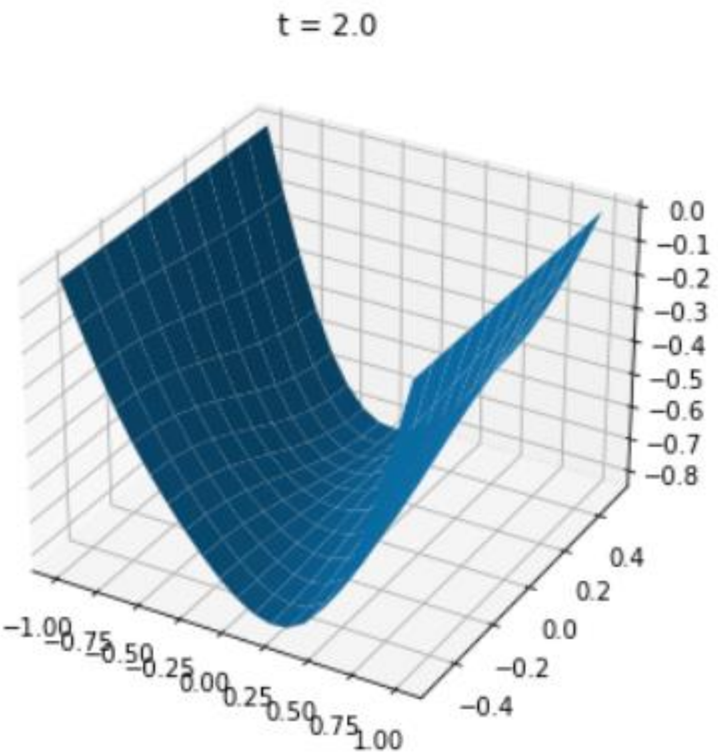
```python
import mpl_toolkits.mplot3d.axes3d as p3
import matplotlib.animation as animation

x_grid, y_grid = np.meshgrid(xs, ys)

fig = plt.figure()
ax = p3.Axes3D(fig)
line = ax.plot_surface(x_grid, y_grid, u[0])

# initialization function: plot the background of each frame
def init():
    line = ax.plot_surface(x_grid, y_grid, u[0])
    return line,

# animation function. This is called sequentially
def animate(i):
    ax.clear()
    line = ax.plot_surface(x_grid, y_grid, u[i])
    return line,

ax.set_xlim3d([-a/2, a/2])
ax.set_xlabel('X')

ax.set_ylim3d([-b/2, b/2])
ax.set_ylabel('Y')

ax.set_zlim3d([-2.5, 2.5])
ax.set_zlabel('Z')

# call the animator. blit=True means only re-draw the parts that have changed.
anim = animation.FuncAnimation(fig, animate, init_func=init,
                               frames=800, interval=20, blit=True)

plt.show()
```

Фиксируем:

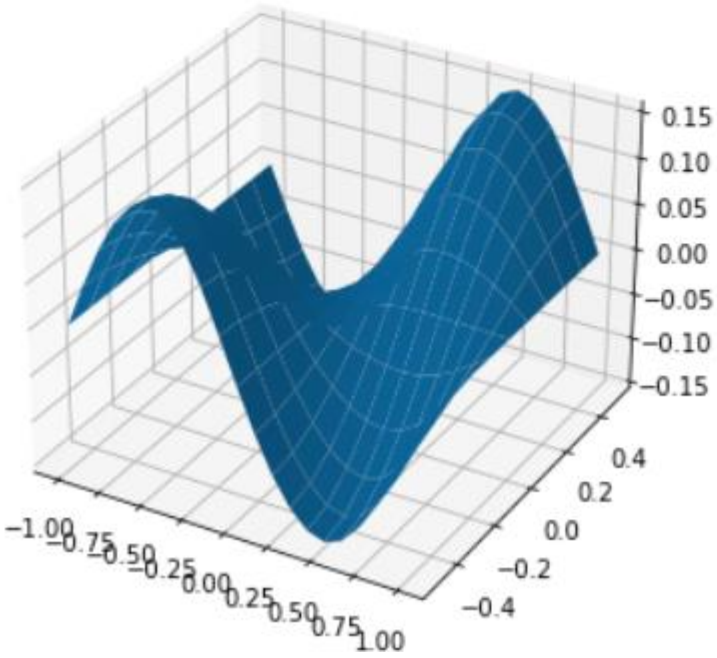
```python
import mpl_toolkits.mplot3d.axes3d as p3
import matplotlib.animation as animation

x_grid, y_grid = np.meshgrid(xs, ys)



for i in range(0, nt, 20):
    fig = plt.figure()
    ax = p3.Axes3D(fig)
    ax.clear()
    plt.title('t = {}'.format(ts[i]))
    line = ax.plot_surface(x_grid, y_grid, u[i])
    plt.show();
```



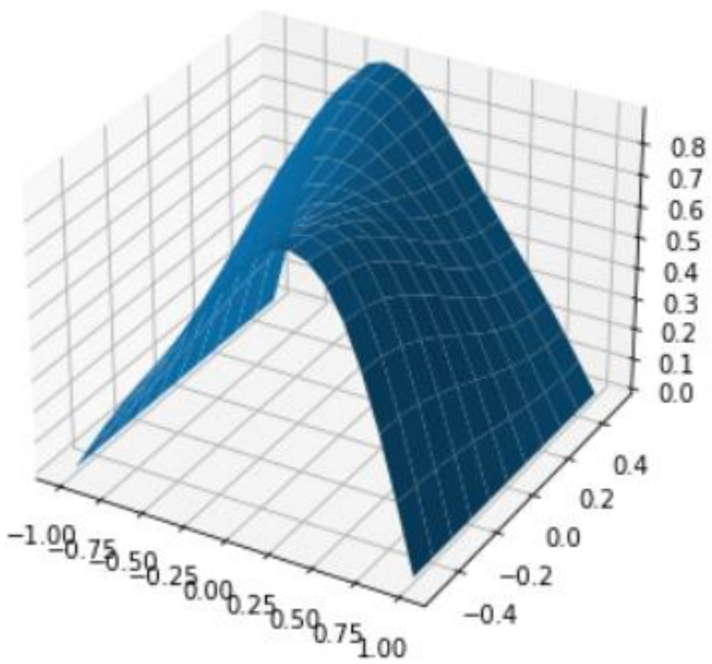

t = 1.0
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
0.4
0.2
0.0
−0.2
−0.4
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


t = 2.0
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
0.4
0.2
0.0
−0.2
−0.4
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

t = 3.0


0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
0.4
0.2
0.0
−0.2
−0.4
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

t = 4.0


0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.4
0.2
0.0
−0.2
−0.4
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Заключение:

В результате данной работы был изучен метод сеток численного решения одномерного и двумерного уравнений теплопроводности, а также составлен алгоритм и программный продукт, решающий эту задачу.