**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ М.В.ЛОМОНОСОВА»**

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

**Отчет о практическом задании по курсу Численные методы в физике по теме:**

«Численное моделирование гидродинамики методом решёточных уравнений Больцмана»

Выполнил студент 403 группы:
Микушин Павел Владимирович

Преподаватель:
Приклонский Владимир Иванович
подпись преподавателя

________________

Москва
2022

# Содержание

# 1 Постановка задачи и аналитическое решение

В этой работе рассматривается применимость метода решеточных уравнений Больцмана для решения одномерного уравнения Бюргерса. Результаты численного моделирования сравниваются с аналитическим решением уравнения Бюргерса.

Уравнение Бюргерса является частным случаем уравнений Навье — Стокса в одномерном случае:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \frac{1}{\mathrm{Re}}\frac{\partial^2 u}{\partial x^2} = 0, \quad \mathrm{Re} > 0,$$

С соответствующими начальными и граничными условиями:

$$u(x,0) = f(x), \quad 0 \leqslant x \leqslant 1,$$
$$u(0,t) = g_1(t), \quad u(1,t) = g_2(t), \quad t \geqslant 0,$$

где Re - число Рейнольдса, связанное с коэффициентом кинематической вязкости следующим соотношением $\nu = 1/Re$.

В качестве дополнительных условий возьмем:

$$u(0,t) = u(1,t) = 0, \quad t > 0,$$

$$u(x,0) = \sin \pi x, \quad 0 \leqslant x \leqslant 1.$$

Аналитическое решение этой задачи представлено в статье [1] и может быть построено в виде отношения рядов. Ниже представлены коэффициенты разложения и само аналитическое решение:

$$u(x,t) = \frac{2\pi\upsilon\sum_{k=1}^{\infty} kA_k \sin(k\pi x)\exp(-k^2\upsilon\pi^2 t)}{A_0 + \sum_{k=1}^{\infty} A_k \cos(k\pi x)\exp(-k^2\upsilon\pi^2 t)},$$

$$A_0 = \int_0^1 \exp\{-(2\pi\upsilon)^{-1}[1-\cos(\pi x)]\}dx,$$

$$A_k = 2\int_0^1 \exp\{-(2\pi\upsilon)^{-1}[1-\cos(\pi x)]\}\cos(k\pi x)dx, \quad k \geqslant 1.$$

## 2 Решеточные уравнения Больцмана

Методы решёточных уравнений Больцмана — класс методов вычислительной гидродинамики для моделирования жидкостей. В отличие от многих других методов, этот метод не решает уравнения Навье — Стокса, а моделирует поток ньютоновской жидкости дискретным кинетическим уравнением Больцмана.

Методы решёточных уравнений Больцмана рассматривают жидкость как совокупность относительно небольшого числа частиц, причём на каждом шаге рассматривается их распространение и столкновения (релаксация). В каждой ячейке решётки поток жидкости рассматривается как совокупность элементарных потоков (например, идущих в соседние и следующие за соседними ячейки). Таким образом, с учетом модели столкновений LBGK, справедливы следующие уравнения для дискретных функций распределения:

$$f_i(x + c_i\Delta t, t + \Delta t) - f_i(x,t) = -\frac{1}{\tau}\left[f_i(x,t) - f_i^{eq}(x,t)\right],$$

Где $f_i$ - функция распределения плотности; $f_i^{eq}$ - функция распределения, соответствующая локальному равновесию; $\tau$ - безразмерное время релаксации; $c_i$ - дискретные скорости: $\{c0; c1; c2\} = \{0; c; -c\}$; $c = \Delta x/\Delta t$; $\Delta x$- расстояние между элементами решетки, $\Delta t$ - шаг по времени.

Равновесные функции распределения могут быть заданы следующим образом.

$$f_i^{eq} = \begin{cases} u - \frac{u^3}{3c^2} - \frac{\alpha u}{3}, & i = 0, \\ \frac{1}{2}\left(\frac{u^2}{2c} + \frac{u^3}{3c^2} + \frac{\alpha u}{3}\right), & i = 1, \\ \frac{1}{2}\left(-\frac{u^2}{2c} + \frac{u^3}{3c^2} + \frac{\alpha u}{3}\right), & i = 2, \end{cases}$$

Здесь мы используем одномерную решетку D1Q3. Каждый элемент решетки состоит из трех точек и частицы могут иметь три различных скорости.

Интересующая нас макроскопическая величина $u$ может быть выражена через функцию распределения.Попробуем получить уравнение Бюргерса из решеточных уравнений Больцмана. Для этого будем считать, что:

$$u = \sum_i f_i,$$

$$\sum_i c_i f_i^{eq} = \frac{u^2}{2},$$

$$\sum_i c_i c_i f_i^{eq} = \frac{u^3}{3} + \alpha c_s^2 u.$$

Далее, следуя теории Чепмена-Энскога, разложим функцию распределения по параметру  и воспользуемся разложением компонент исходных решеточных уравнений в ряд Тейлора.

$$f_i = f_i^{eq} + \varepsilon f_i^{(1)} + \varepsilon^2 f_i^{(2)},$$

$$\frac{\partial}{\partial t} = \varepsilon \frac{\partial}{\partial t_1} + \varepsilon^2 \frac{\partial}{\partial t_2},$$

$$\frac{\partial}{\partial x} = \varepsilon \frac{\partial}{\partial x_1},$$

$$\Delta t \frac{\partial f_i}{\partial t} + \Delta t \frac{\partial}{\partial x}(c_i f_i) + \frac{1}{2}\Delta t^2 \frac{\partial^2 f_i}{\partial t^2} + \Delta t^2 \frac{\partial^2}{\partial t \partial x}(c_i f_i) + \frac{1}{2}\Delta t^2 \frac{\partial^2}{\partial x^2}\left(c_i^2 f_i\right) = -\frac{1}{\tau}\left(f_i - f_i^{eq}\right).$$

$$O(\varepsilon) : \frac{\partial f_i^{eq}}{\partial t_1} + \frac{\partial}{\partial x_1}\left(c_i f_i^{eq}\right) = -\frac{f_i^{(1)}}{\tau \Delta t},$$

$$O(\varepsilon^2) : \frac{f_i^{eq}}{\partial t_2} + \frac{\partial f_i^{(1)}}{\partial t_1} + \frac{\partial}{\partial x_1}\left(c_i f_i^{(1)}\right) + \frac{\Delta t}{2}\left(\frac{\partial}{\partial t_1} + c_i \frac{\partial}{\partial x_1}\right)^2 f_i^{eq} = -\frac{1}{\tau \Delta t} f_i^{(2)}.$$

$$\frac{f_i^{eq}}{\partial t_2} + \left(\frac{\partial}{\partial t_1} + c_i \frac{\partial}{\partial x_1}\right)\left(1 - \frac{1}{2\tau}\right) f_i^{(1)} = -\frac{1}{\tau \Delta t} f_i^{(2)}.$$

$$\sum_i c_i f_i^{(1)} = -\tau \Delta t \left[ u \frac{\partial u}{\partial t_1} + \frac{\partial}{\partial x_1} \left( \frac{u^3}{3} \right) + \alpha c_s^2 \frac{\partial u}{\partial x_1} \right].$$

$$\frac{\partial u}{\partial t_1} + \frac{\partial}{\partial x_1} \left( \frac{u^2}{2} \right) = 0.$$

$$\frac{\partial u}{\partial t_2} - \alpha c_s^2 \left( \tau - \frac{1}{2} \right) \Delta t \frac{\partial^2 u}{\partial x_1^2} = 0.$$

Таким образом мы получаем уравнение Бюргерса:

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} - \upsilon \frac{\partial^2 u}{\partial x^2} = 0,$$

# 3 Численное решение

Были выбраны следующие параметры: шаг решетки - $h = 1/199$, $\nu = 0.1$, шаг по времени $\Delta t = 0.001$.

Дискретизация уравнений Больцмана обеспечивает первый порядок аппроксимации, при этом рассматриваемая схема может считаться явной. В соответствии с этим ожидается, что погрешность численного решения не будет стремиться к нулю при уменьшении размера соответствующих решеток.

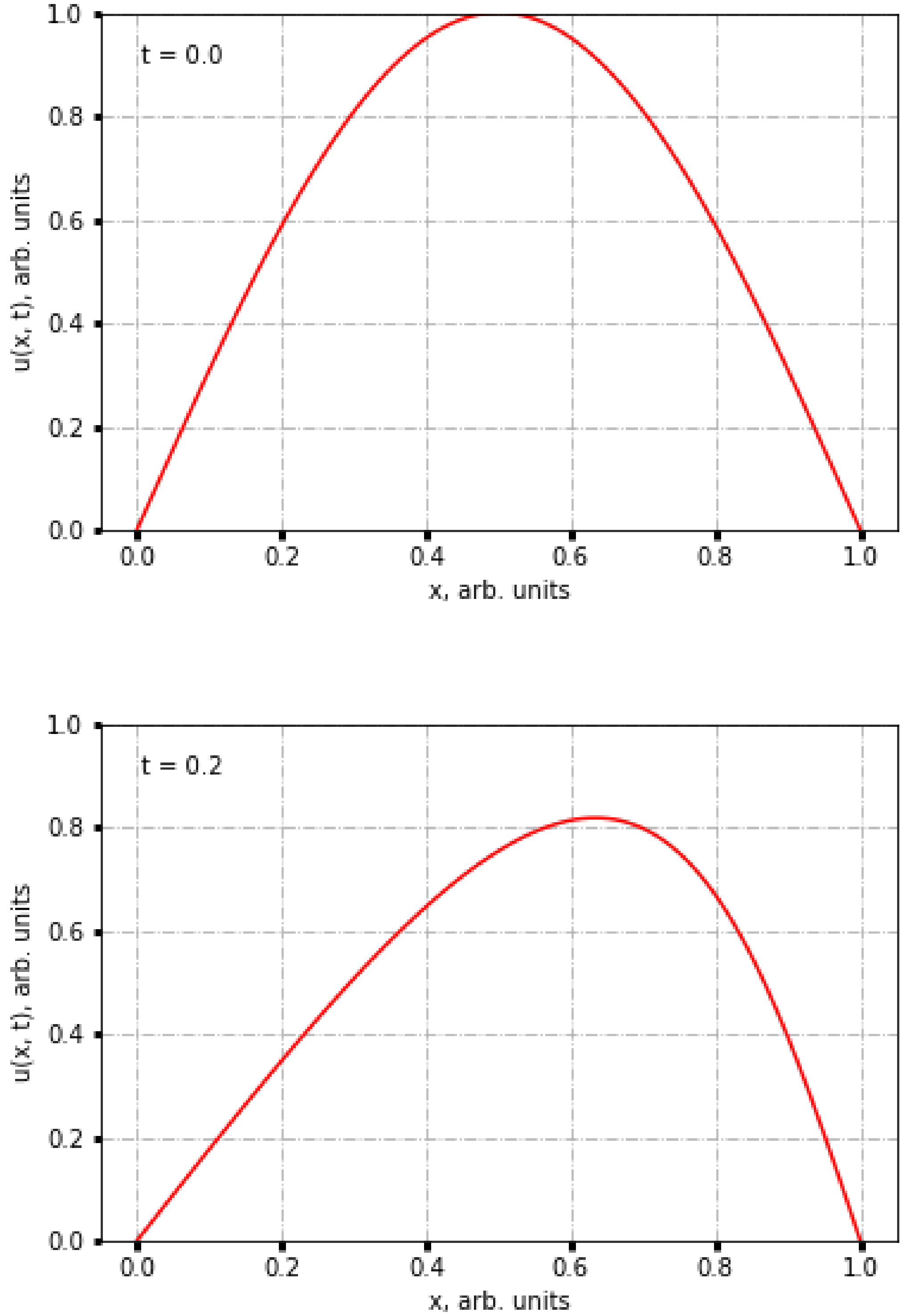



Рис. 1: Аналитическое решение

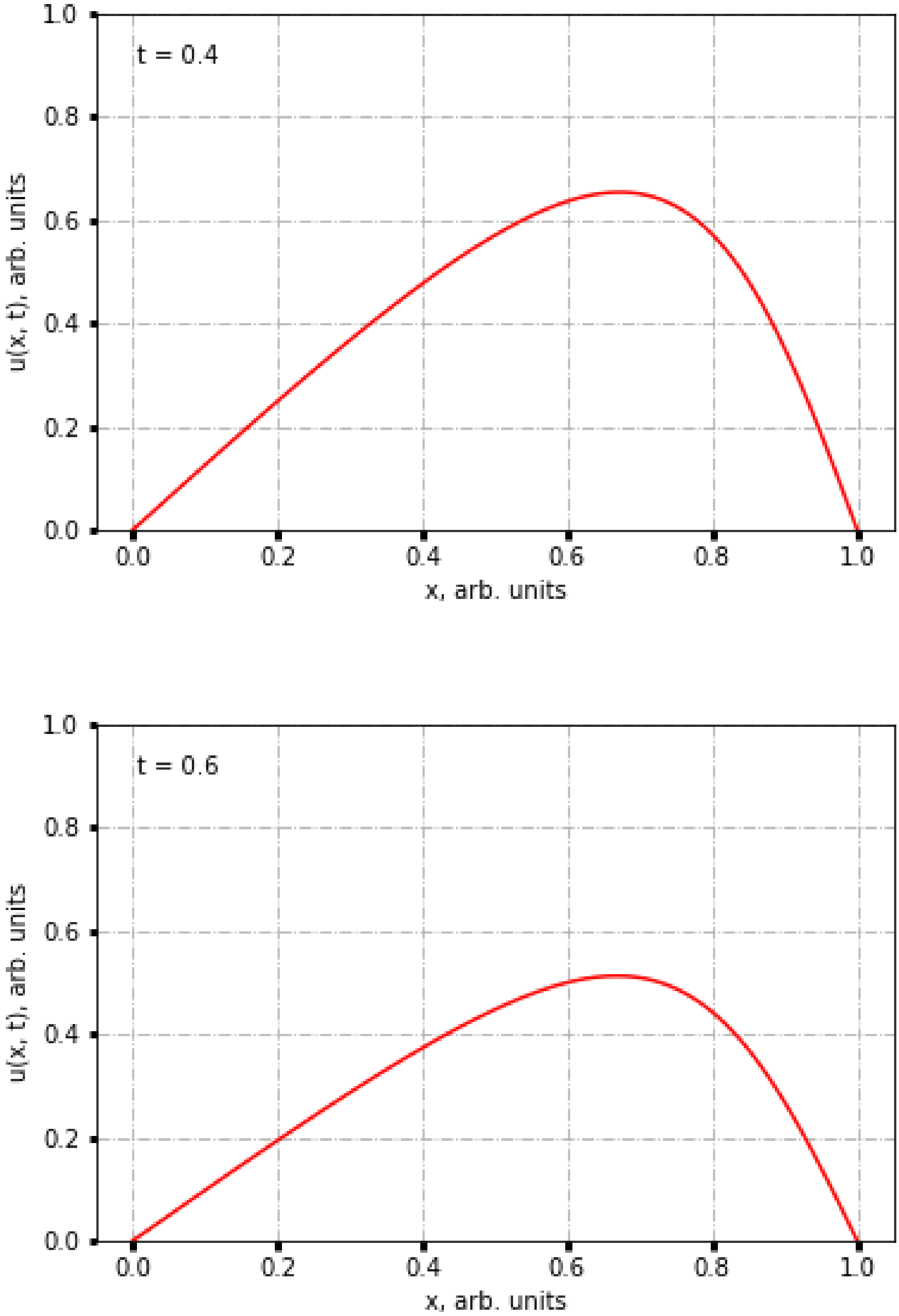
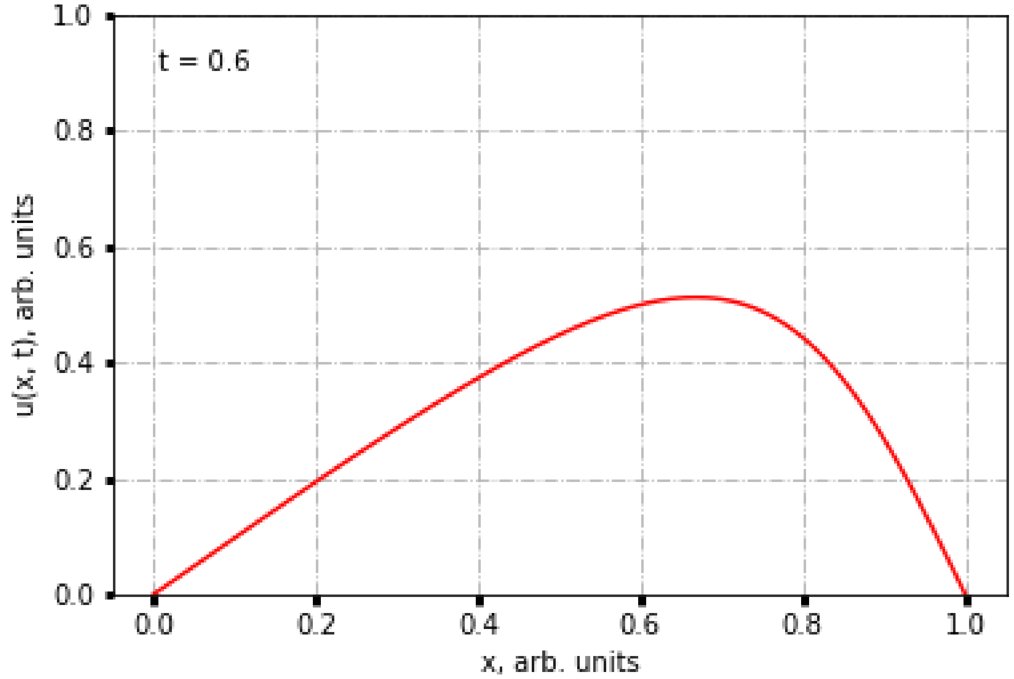
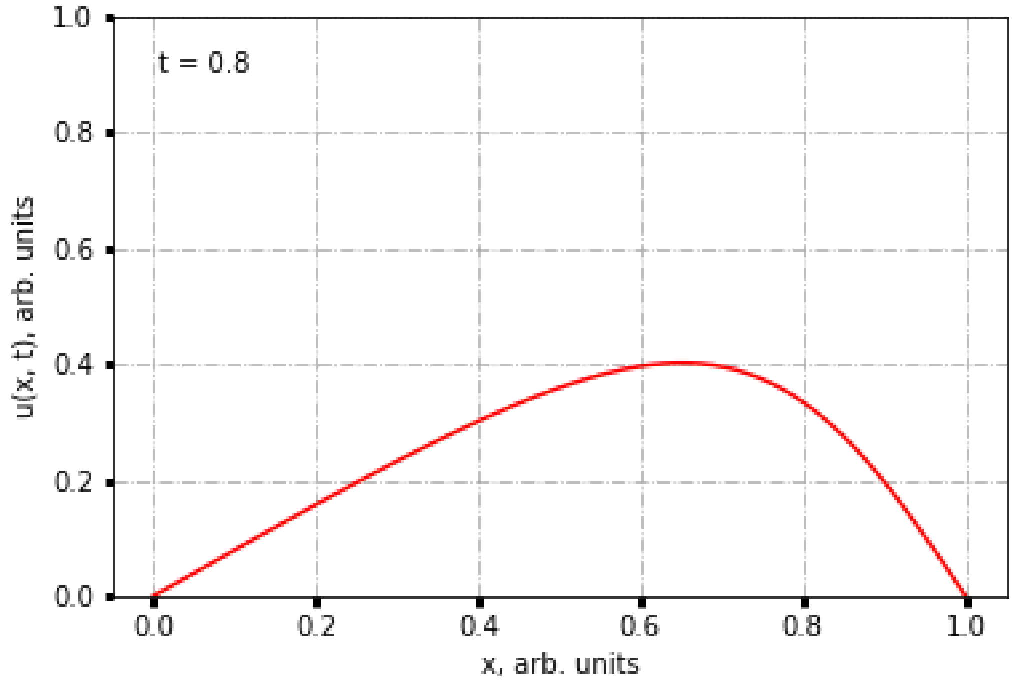



Рис. 2: Аналитическое решение



t = 0.6
u(x, t), arb. units
x, arb. units

t = 0.8
u(x, t), arb. units
x, arb. units

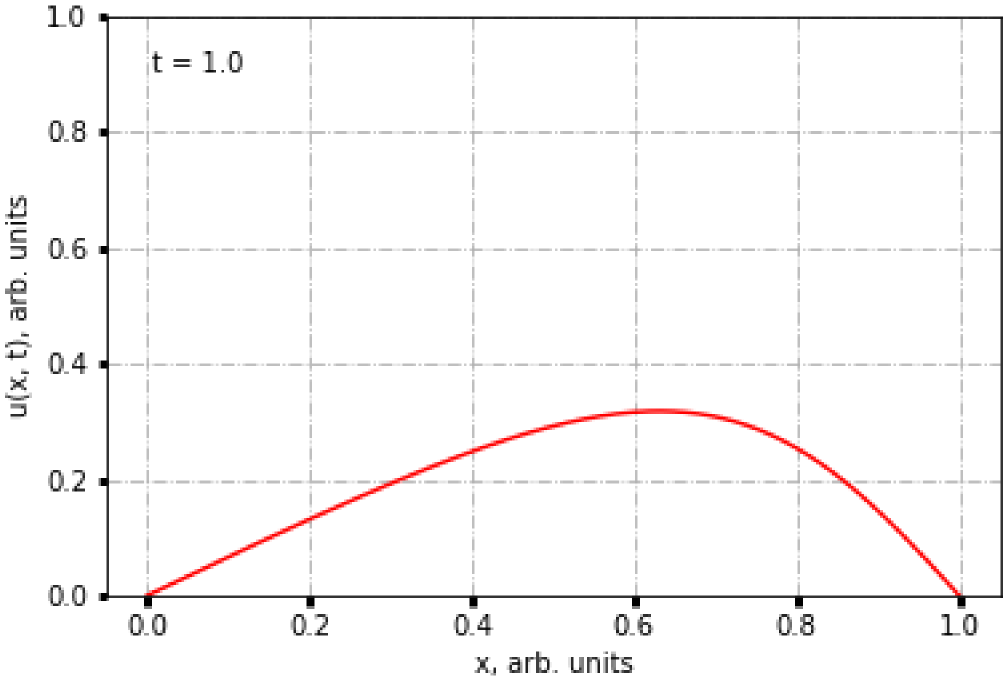



Рис. 3: Аналитическое решение

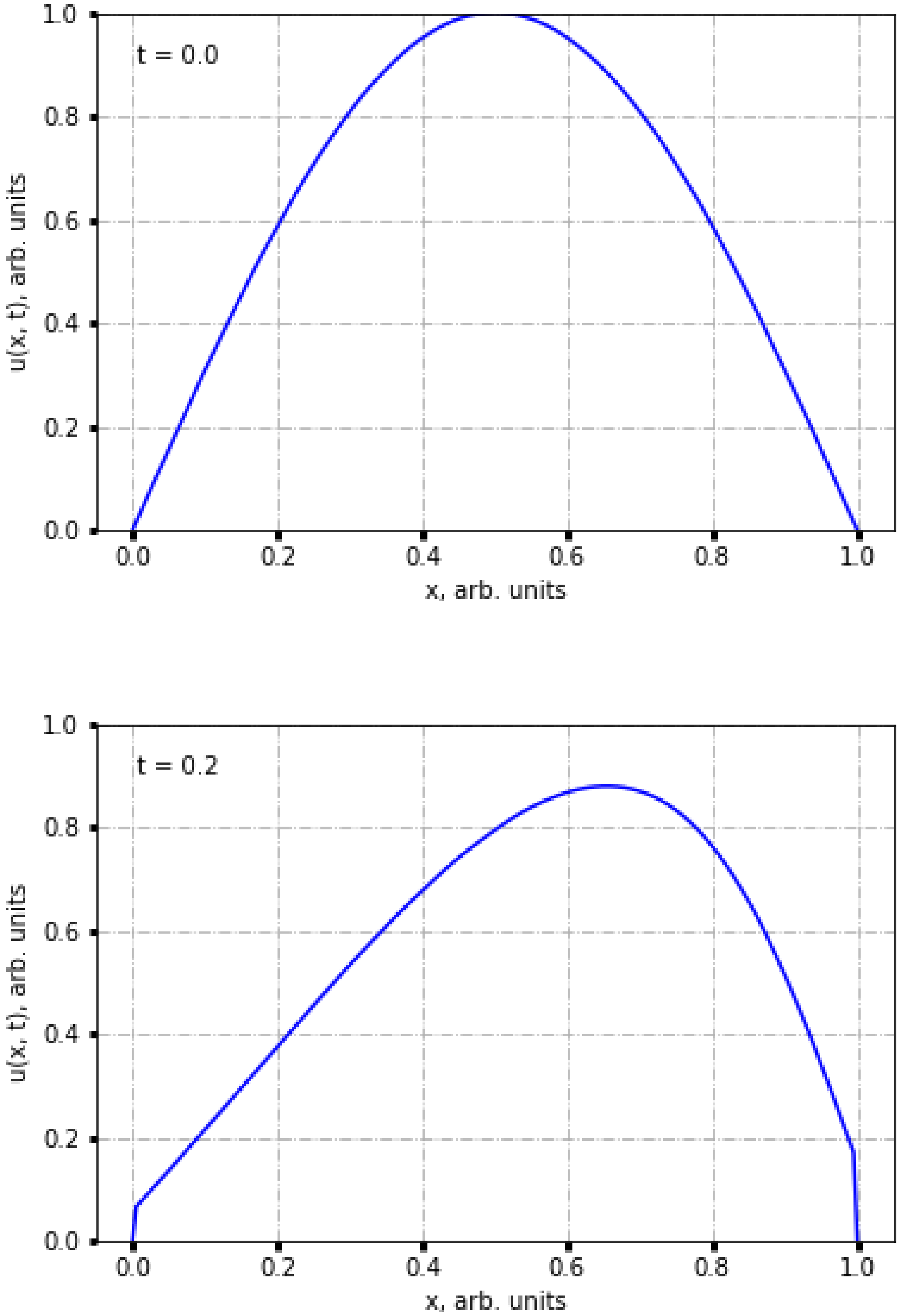



Рис. 4: Численное решение методом Решеточных уравнений Больцмана

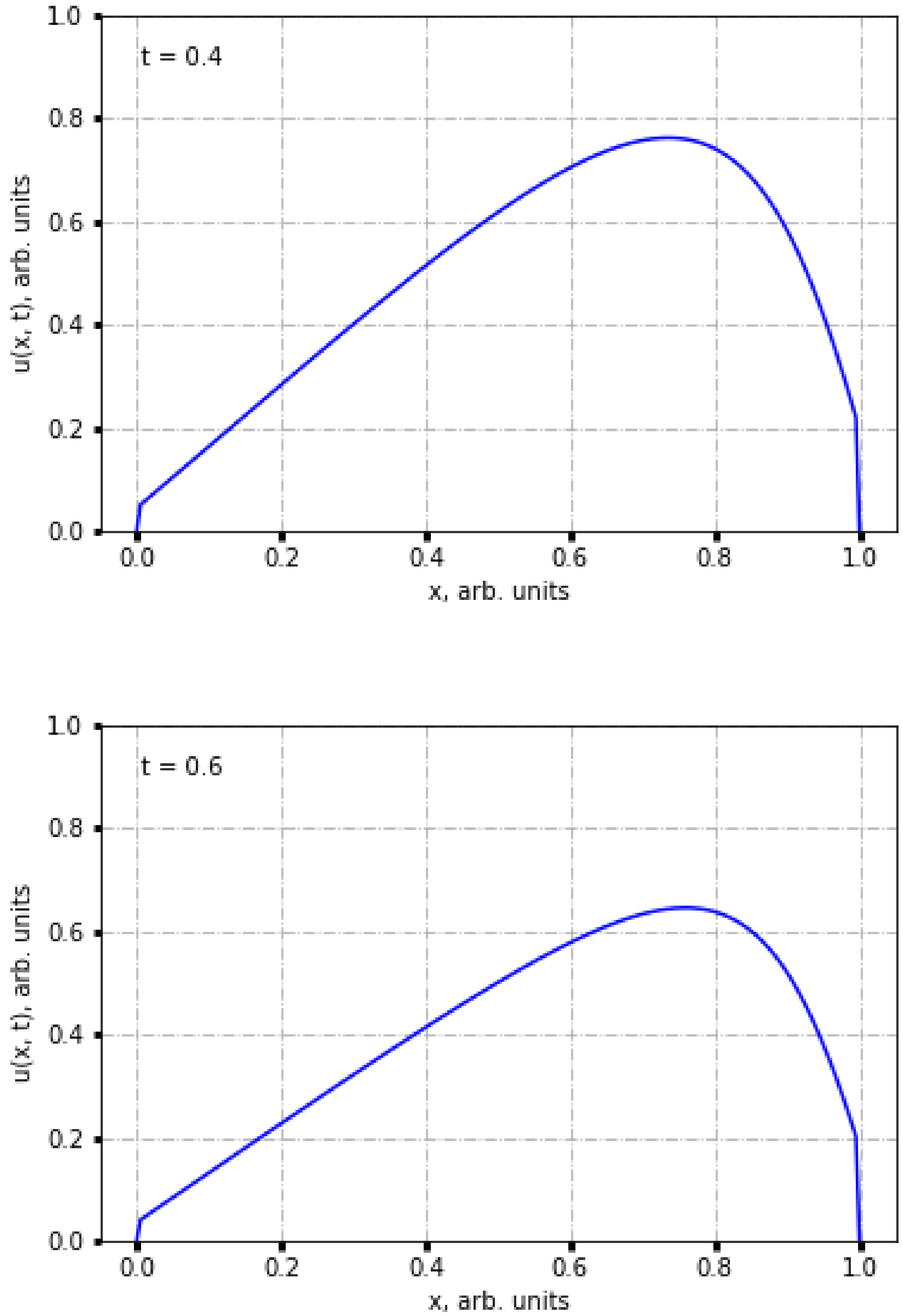



Рис. 5: Численное решение методом Решеточных уравнений Больцмана

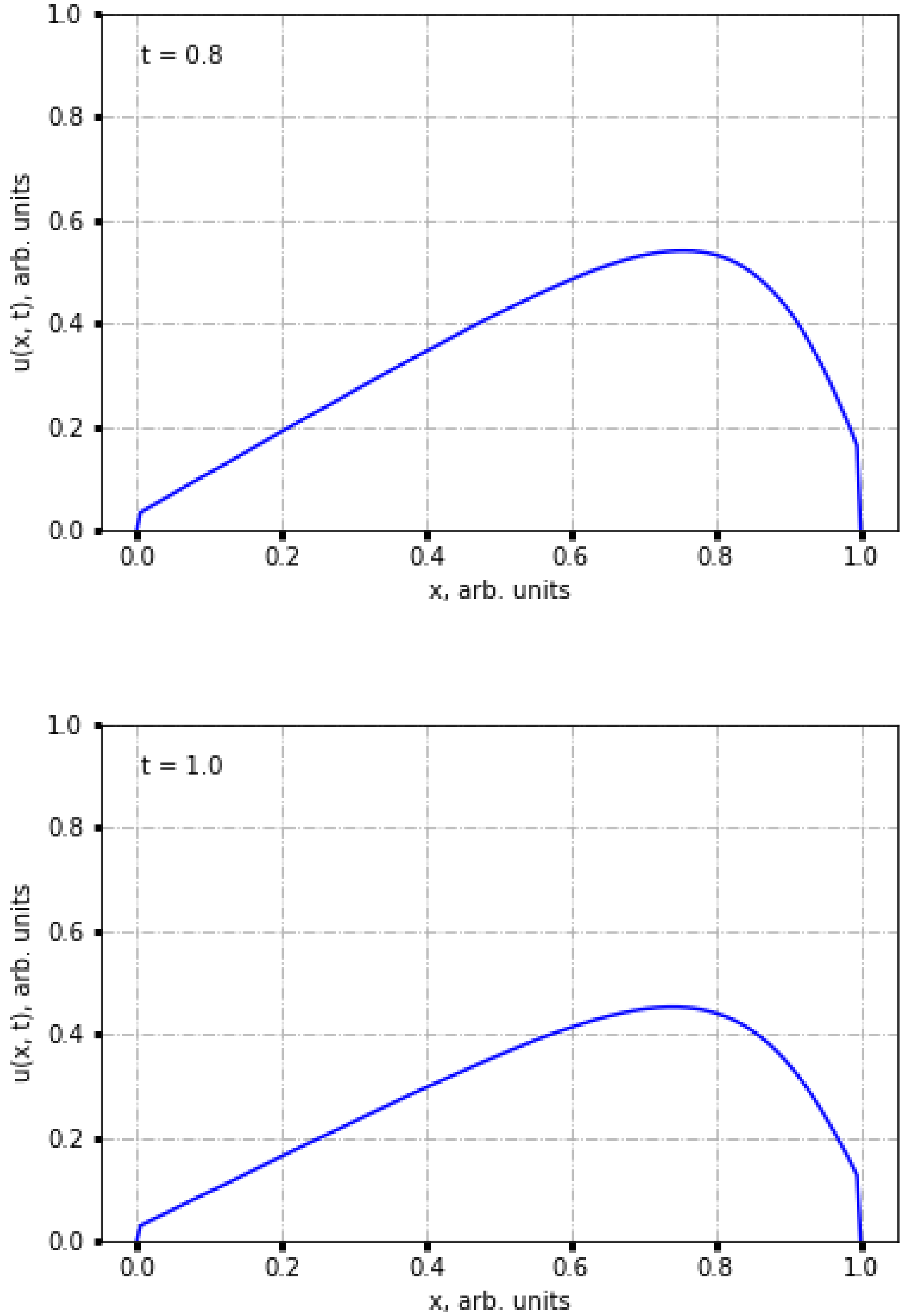



Рис. 6: Численное решение методом Решеточных уравнений Больцмана

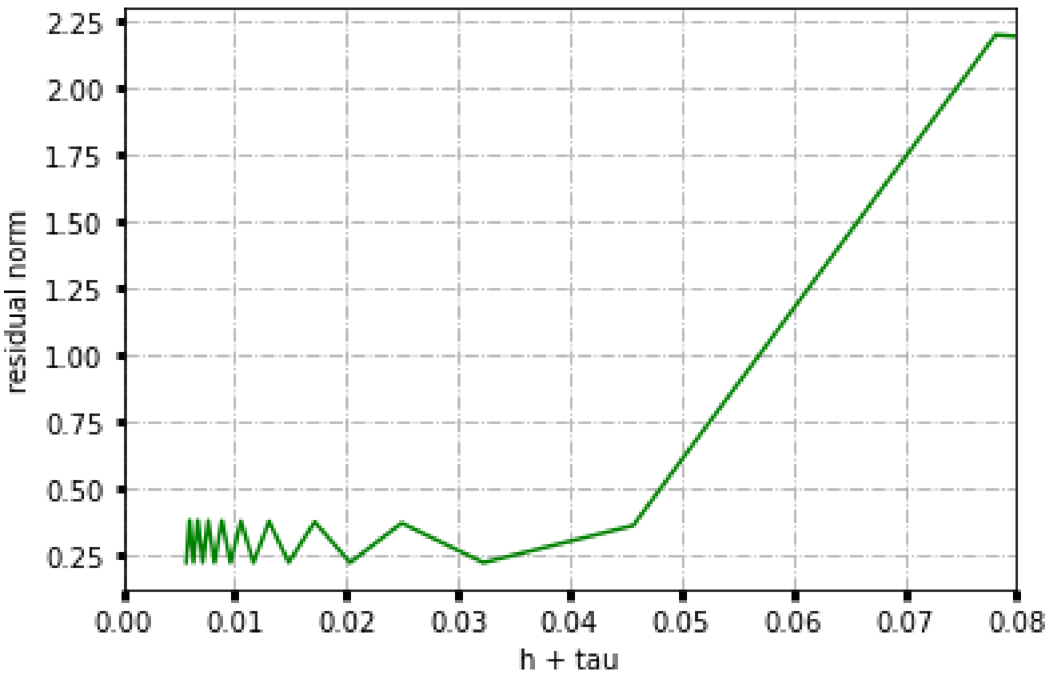



Рис. 7: Норма невязки в зависимости от параметров решетки

# 4 Литература

## Список литературы

[1] M. Xu, R.H. Wang, J.H. Zhang, Q. Fang, A novel numerical scheme for solving Burgers' equation, Appl. Math. Comput. 217 (2011) 4473–4482.

[2] Сайт кафедры математики физического факультета МГУ: http://math.phys.msu.ru/

[3] Численные методы : в 2 кн. Кн. 1. Численный анализ : Ч-671 учебник для студ. учреждений высш. проф. образования/ Н.Н.Калиткин, Е.А.Альшина. - М. : Издательский центр «Академия», 2013. - 304 с. - (Университетский учебник. Сер. Прикладная математика и информатика). ISBN 978-5-7695-5089-8

# 5 Приложение

Для решения задачи использовался язык программирования python, ниже приведены основные ячейки кода из Jupyter Notebook:

In [25]:

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

import scipy.integrate as integrate
from scipy import optimize

# Анимация
from matplotlib import animation
from IPython.display import HTML

import matplotlib
matplotlib.rcParams['animation.embed_limit'] = 2**128
```

In [26]:

```
# Глобальные параметры модели
t_min = 0
t_max = 1

x_min = 0
x_max = 1

v = 0.1
```

In [27]:

```
def A(k, v, X):
    """
    Возвращает коэффициенты разложения в ряд Фурье для построения аналити
    """
    k = int(k)
    if(k == 0):
        funct = lambda x: np.exp(-(1-np.cos(np.pi*x))/(2*np.pi*v))
        return integrate.simpson(funct(X), X, dx=(X[1]-X[0]))
    if(k >= 1):
        funct = lambda x: np.exp(-(1-np.cos(np.pi*x))/(2*np.pi*v))*np.co
        return 2*integrate.simpson(funct(X), X, dx=(X[1]-X[0]))
    return None
```

In [28]:

```
def analytical_solution(v, x, t, k_max, A_array):
    up_eps = 1e-6
    up_k = 0
    previous_up_series = -1
    up_series = 0
    while(np.abs(up_series - previous_up_series) >= up_eps and (up_k < k
        up_k += 1
        previous_up_series = 0
        previous_up_series += up_series
        up_series += A_array[up_k]*up_k*np.sin(up_k*np.pi*x)*np.exp(-up_

    down_eps = 1e-6
    down_k = 0
    previous_down_series = -1
    down_series = 0
    while(np.abs(down_series - previous_down_series) >= down_eps and (do
        down_k += 1
        previous_down_series = 0
        previous_down_series += down_series
        down_series += A_array[down_k]*np.cos(down_k*np.pi*x)*np.exp(-do
```

```
    return 2*np.pi*v*up_series/(A_array[0] + down_series)
```

In [29]:

```
def full_analytical_solver(time_nodes, space_nodes):
    N_t = time_nodes
    N_x = space_nodes

    T = np.linspace(t_min, t_max, num=N_t)
    X = np.linspace(x_min, x_max, num=N_x)

    k_max = 100
    A_array = [A(element, v, X) for element in np.arange(0, k_max+1)]

    u_analytical = np.zeros((N_t, N_x))
    u_analytical[0] = np.sin(np.pi*X)

    for i in range(1, N_t):
        for j in range(1, N_x-1):
            u_analytical[i][j] = analytical_solution(v, X[j], T[i], k_ma

    return u_analytical, X, T
```

In [30]:

```
u_analytical, X, T = full_analytical_solver(1001, 200)
```

In [31]:

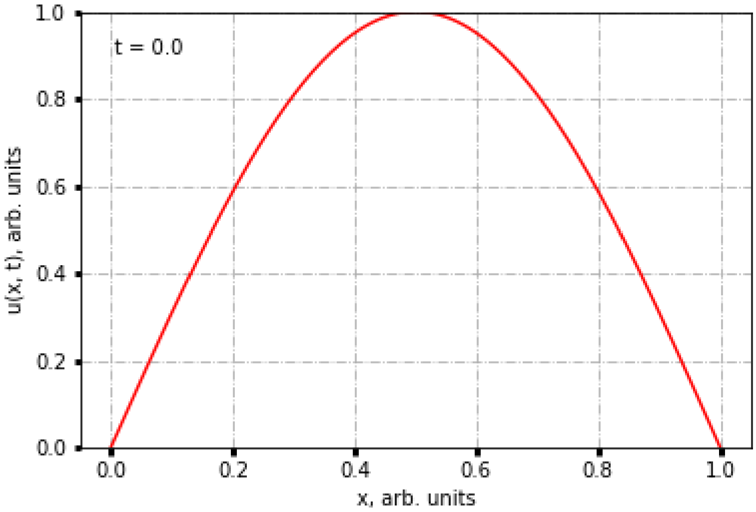
```
# Создаем фоновый рисунок и оси
fig, ax = plt.subplots()
plt.close() # не показывать статичный фон

def animate(i):
    """
    Функция для рисования кадров.
    """
    ax.clear()
    ax.set_ylim([0, 1])
    ax.grid(True, linestyle='-.')
    ax.tick_params(labelcolor='black', labelsize='medium', width=3)
    ax.set_xlabel("x, arb. units")
    ax.set_ylabel("u(x, t), arb. units")
    ax.text(0.05, 0.9, 't = ' + str(np.around(T[i], decimals=3)),
            transform=ax.transAxes, animated=True)

    line = ax.plot(X, u_analytical[i], color='r', animated=True)
    return line

ani = animation.FuncAnimation(fig, animate, frames=T.shape[0], interval=
HTML(ani.to_jshtml())
```

Out[31]:




Once Loop Reflect

In [32]:

```
ani.save("analytical.gif", writer=animation.PillowWriter(fps=30))
```

In [33]:

```
def LBM_solver(time_nodes, space_nodes):
    N_t = time_nodes
    N_x =space_nodes

    T = np.linspace(t_min, t_max, num=N_t)
    X = np.linspace(x_min, x_max, num=N_x)

    delta_x = X[1] - X[0]
    delta_t = T[1] - T[0]
    c = delta_x/delta_t


    # Lattice speeds / weights
    NL = 3
    idxs = np.arange(NL)
    cxs = c*np.array([0, 1, -1])

    weights = np.array([2/4, 1/4, 1/4])
    cs_squared = np.sum(weights*cxs**2)

    t_relax = 0.5 + v/(delta_t*cs_squared)

    shift_indexes = np.array([0, 1, -1])


    def f_0_eq(u):
        return u - (u**3)/(3*c**2) - u/3

    def f_1_eq(u):
        return 0.5*((u**2)/(2*c) + (u**3)/(3*c**2) + u/3)

    def f_2_eq(u):
        return 0.5*(-(u**2)/(2*c) + (u**3)/(3*c**2) + u/3)
```

```
    # Начальные условия
    f = np.zeros((N_x, NL))
    u0 = np.sin(np.pi*X)
    f[:, 0] = u0 - u0**3 /(3*c**2) - u0/3
    f[:, 1] = 0.5*(u0**2 /(2*c) + u0**3 /(3*c**2) + u0/3)
    f[:, 2] = 0.5*(-u0**2 /(2*c) + u0**3 /(3*c**2) + u0/3)

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

    result = []

    # Главный цикл
    for i in range(N_t):

        # Calculate fluid variables
        rho = np.sum(f, 1)
        result.append(rho)

        # Apply Collision
        feq = np.zeros(f.shape)
        feq[:, 0] = f_0_eq(rho)
        feq[:, 1] = f_1_eq(rho)
        feq[:, 2] = f_2_eq(rho)

        f_hat = -(1.0/t_relax)*(f - feq) + f

        # Drift
        for i, cx in zip(idxs, cxs):
            f[:, i] = np.roll(f_hat[:,i], shift_indexes[i], axis=0)

        # Apply boundary
        f[0, :] = 0
        f[-1, :] = 0
    return result, X, T
```

In [34]:

```
result, X, T = LBM_solver(1001, 200)
```

In [35]:

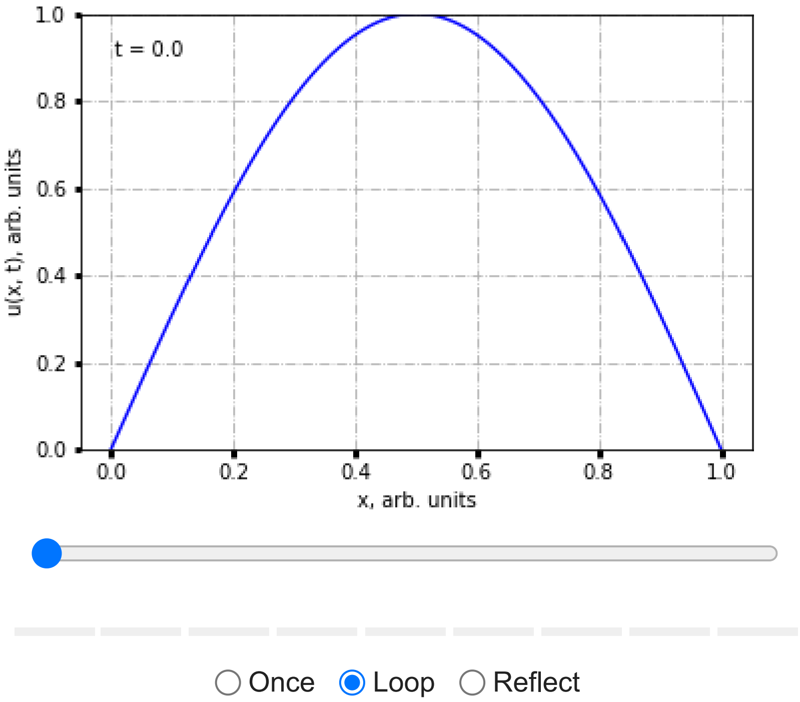
```
fig, ax = plt.subplots()
plt.close() # не показывать статичный фон

def animate(i):
    """
    Функция для рисования кадров.
    """
    ax.clear()
    ax.set_ylim([0, 1])
    ax.grid(True, linestyle='-.')
    ax.tick_params(labelcolor='black', labelsize='medium', width=3)
    ax.set_xlabel("x, arb. units")
    ax.set_ylabel("u(x, t), arb. units")
    ax.text(0.05, 0.9,
            't = ' + str(np.around(T[i], decimals=3)),
                         transform=ax.transAxes,
                         animated=True)

    line = ax.plot(X, result[i], color='b', animated=True)
    return line
```

```
ani = animation.FuncAnimation(fig, animate, frames=T.shape[0], interval=
HTML(ani.to_jshtml())
```

Out[35]:




In [36]:

```
ani.save('LBM.gif', writer=animation.PillowWriter(fps=30))
```

In [37]:

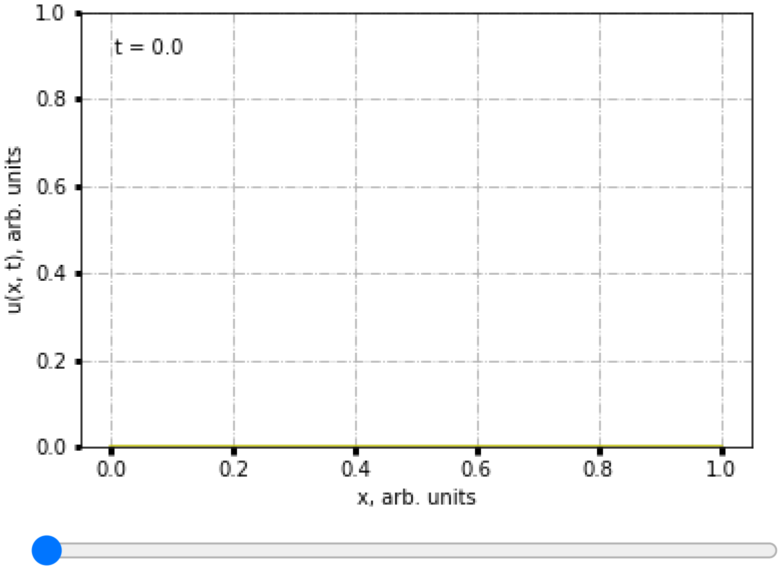
```
fig, ax = plt.subplots()
plt.close() # не показывать статичный фон

def animate(i):
    """
    Функция для рисования кадров.
    """
    ax.clear()
    ax.set_ylim([0, 1])
    ax.grid(True, linestyle='-.')
    ax.tick_params(labelcolor='black', labelsize='medium', width=3)
    ax.set_xlabel("x, arb. units")
    ax.set_ylabel("u(x, t), arb. units")
    ax.text(0.05, 0.9,
            't = ' + str(np.around(T[i], decimals=3)),
                       transform=ax.transAxes,
                       animated=True)

    line = ax.plot(X, np.abs(result[i] - u_analytical[i]), color='y', an
    return line


ani = animation.FuncAnimation(fig, animate, frames=T.shape[0], interval=
HTML(ani.to_jshtml())
```

Out[37]:



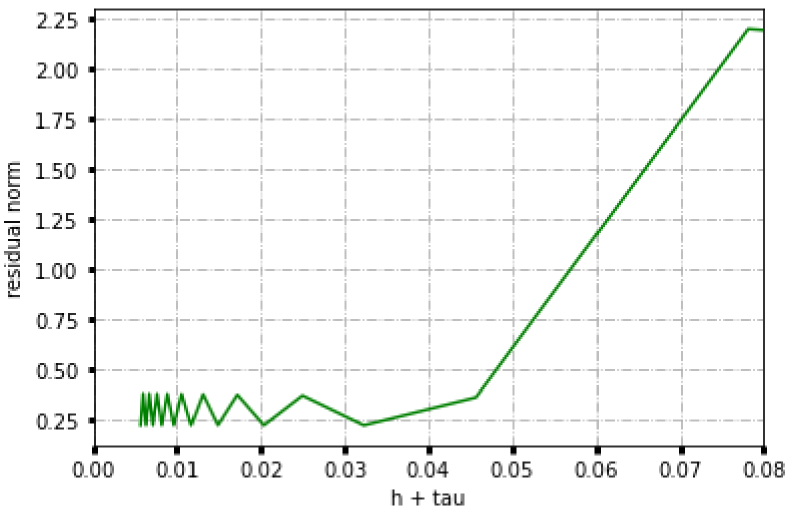

Once Loop Reflect

In [38]:
```
ani.save('pogreshnost.gif', writer=animation.PillowWriter(fps=30))
```

In [50]:
```
num_x_2 = np.arange(5, 200, 10)
num_t_2 = np.arange(50, 2000, 100)
results_list_2 = [LBM_solver(num_t_2[i], num_x_2[i])[0] for i in range(n
analytical_list_2 = [full_analytical_solver(num_t_2[i], num_x_2[i])[0] f
res2 = [np.abs(results_list_2[i] - analytical_list_2[i]).max() for i in 
```

In [58]:
```
fig, ax = plt.subplots()
ax.set_xlim(0, 0.08)
ax.grid(True, linestyle='-.')
ax.tick_params(labelcolor='black', labelsize='medium', width=3)
ax.set_xlabel("h + tau")
ax.set_ylabel("residual norm")
ax.plot(1/(num_x_2-1) + 1/(num_t_2-1), res2, color='g')

plt.show()
```



In [ ]: