**Часть 1. Теоретический материал**

1. **Постановка задачи**

Рассматривается вторая краевая задача для уравнения колебаний:

(1)

Используется явная схема. Колебания моделируются в зависимости от внешнего сосредоточенного импульса:

1. **Решение тестовых примеров**

Первый тестовый пример:

(1.1)

Введем обозначение оператора .

Тогда исходное уравнение в частных производных записывается в виде:

Заметим, что для данного тестового примера , где С – константа. Тогда можно найти частное решение этого уравнения в виде

Тогда

Общее решение можно найти в виде:

Действительно для такого u справедливо:

Так как

Функции h и g найдём из граничных условий уже с учетом найденного частного решения и нашего a = 1:

(2)

Проинтегрируем второе уравнение:

(3)

Решим систему уравнений (2), (3) относительно функций h и g:

Тогда решение имеет вид:

(4)

Вообще этот переход справедлив только в области , хотя мы ищем решение в области , тем не менее можно заметить, что решение определенное формулой (4) уже удовлетворяет всем граничным условиям.

В самом деле:

Таким образом, функция u, заданная уравнением (4), удовлетворяет всем уравнениям тестового примера (1.1), то есть является его аналитическим решением.

Второй тестовый пример:

(1.2)

Найдём аналитическое решение таким же способом.

Общее решение:

Найдем функции h и g

(5)

(6)

Из (6):

(7)

Решая систему (5), (7), получаем:

Тогда решение (8)

Проверим, удовлетворяет ли (8) всем условиям в (1.2)

Таким образом (8) определяет решение тестового примера (1.2)

1. **Построение разностной схемы**

Используем метод конечных разностей, заменяем вторые частные производные на вторые разностные производные:

Из полученного уравнения выразим :

(10)

Формула (10) позволяет высчитать значения на j+1-м временном слое во внутренних точках стержня. Однако для этого необходимо знать значения на j-м и на j–1-м слоях.

Значения на нулевом слое задается из уравнение :

(11)

Значение на первом временном слове находится из уравнения. Однако так как формула (9) имеет второй порядок аппроксимации, то и граничные условия, содержащие производные желательно аппроксимировать со вторым порядком. Воспользуемся центральной разностной производной:

Найдем из (9), подставив j = 0:

Так как 2τ, получаем

(12)

Теперь найдём формулы для вычисления значений в граничных точках.

Уравнение и также аппроксимируем со вторым порядком, используя центральную разностную производную

Для нахождения запишем (10) при i = 0:

Так как , получаем

Окончательно получаем формулу для вычисления значения на левом конце стержня

(13)

Аналогично для левого конца

(14)

Остались неизвестны только формулы для вычисления значений на концах стержня на первом временном слое. Запишем (12) при i = 0:

Снова использовав, что , получаем

(15)

Для правого конца:

(16)

Таким образом получили разностную схему, где (11) задает значения на нулевом временном слое, (12), (15), (16) – на первом слое и (10), (13), (14) задают значения на всех слоях, начиная со второго.

1. **Оценка погрешности и исследование на устойчивость**

Разностная схема была построена таким образом, чтобы сохранить второй порядок точности как по пространственным, так и по временным шагам, то есть справедлива следующая оценка:

, где *E* – величина погрешности численного решения задачи

Схема является явной, а значит лишь условно устойчивой. Условие устойчивости выглядит следующим образом:

или (17)

1. **Построение индивидуального тестового примера**

Построим несколько тестовых примеров для моделирования колебаний в зависимости от внешнего сосредоточенного импульса.

(17)

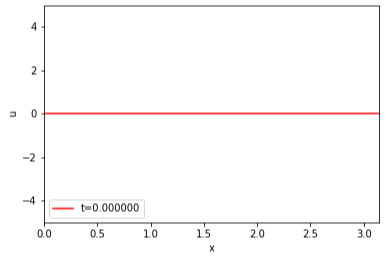
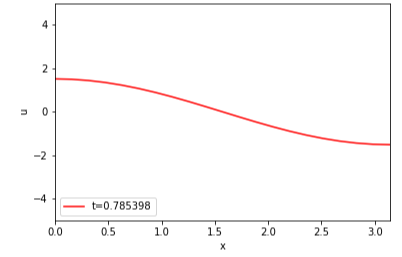
Найти аналитическое решение этой задачи мы не можем, только посмотреть на поведение стержня при такой функции правой части.

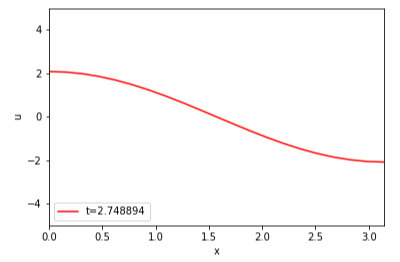
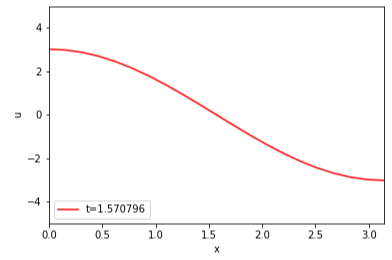
**Часть 2. Практический материал**

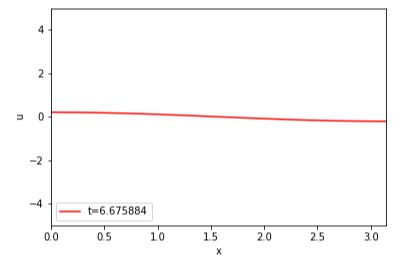
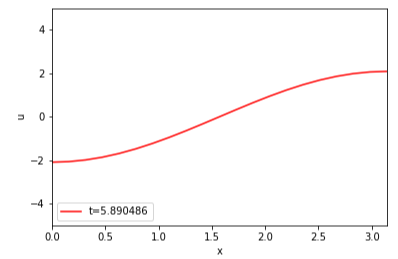
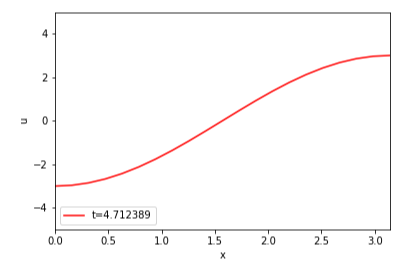
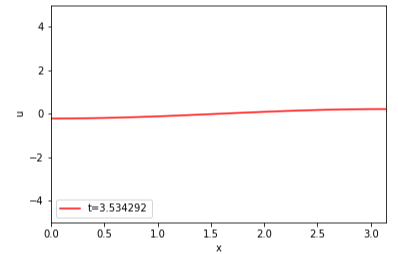
1. **Численное решение задачи для тестовых примеров**

Рассмотрим численное решение первого тестового примера:







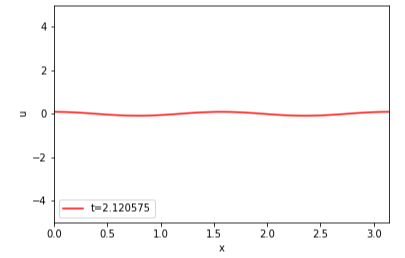
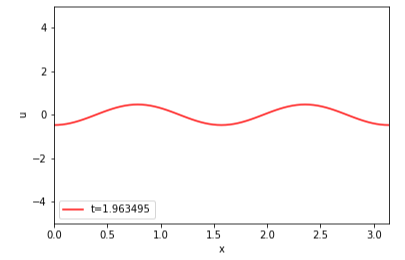
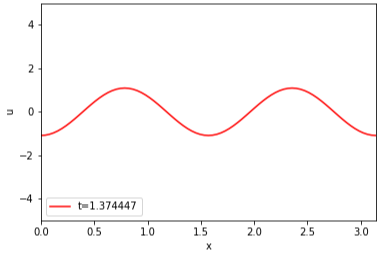
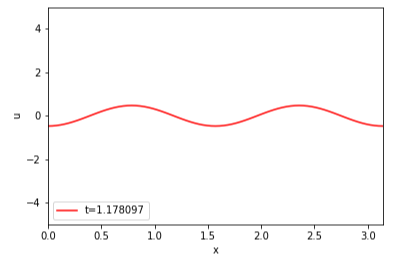
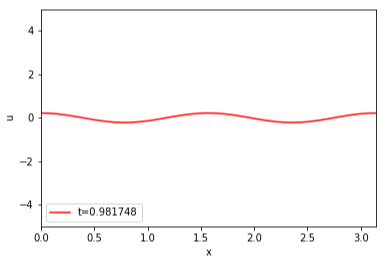
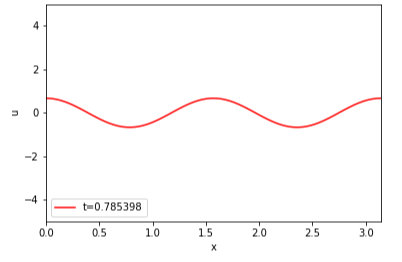
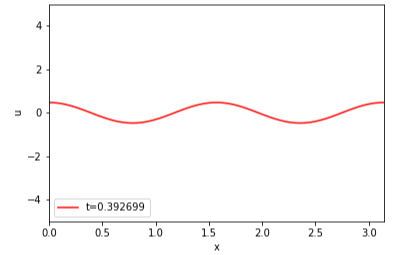
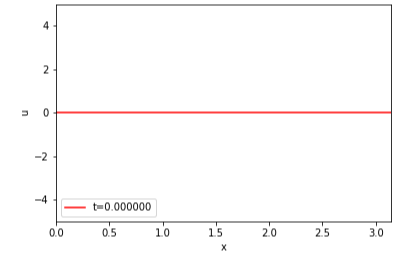
Погрешность вычисляется следующим образом:

, где – аналитическое решение задачи

Уменьшив шаг в 2 раза по обеим переменным получили уменьшение погрешности в 4 раза, что свидетельствует о том, что схема аппроксимирует задачу со вторым порядком



Второй тестовый пример:



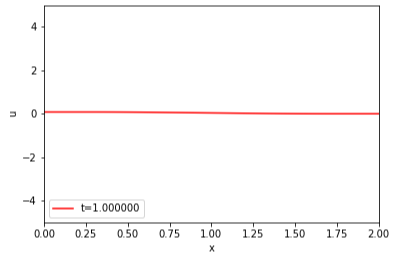
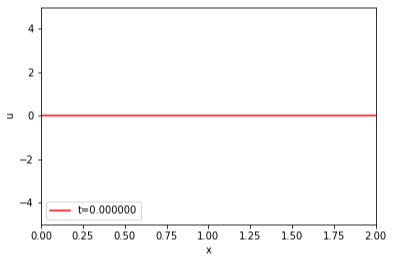
Уменьшим шаги:

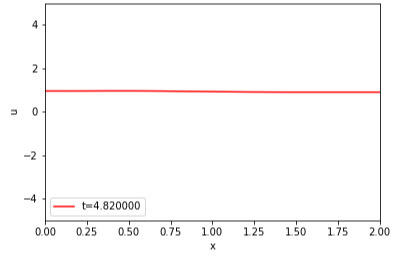
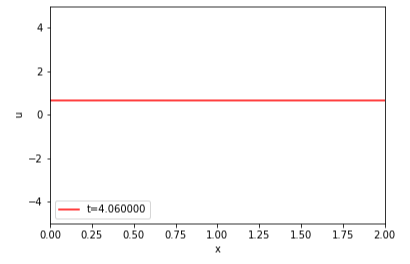
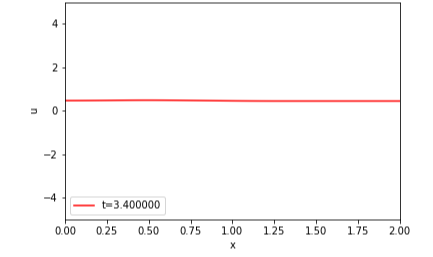
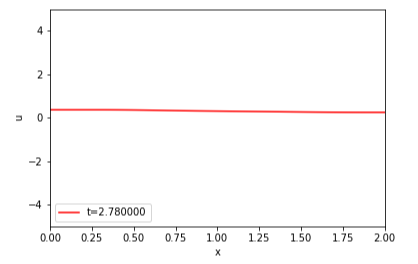
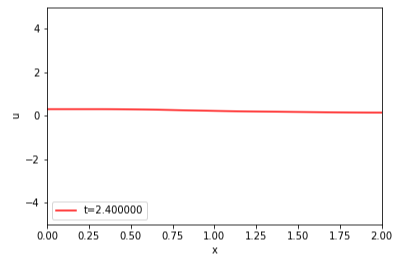
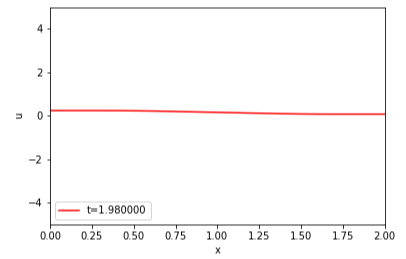
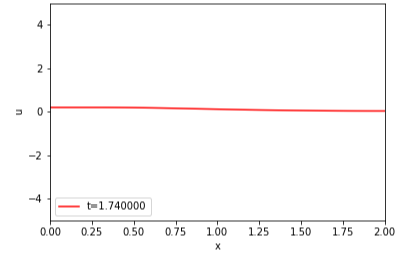
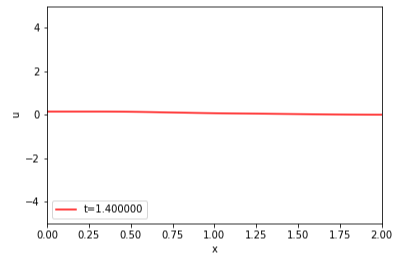




Визуально поведение стержня не меняется при изменении шага, поэтому кадры положений стержня в различные моменты времени приведены только один раз.

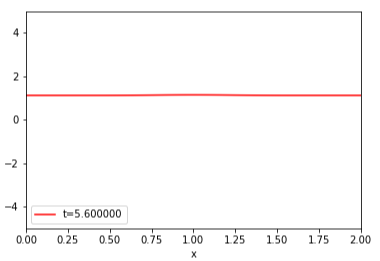
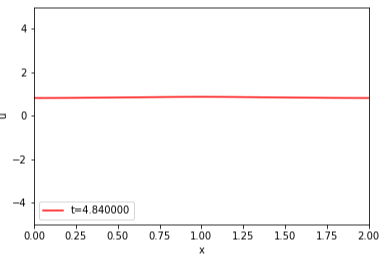
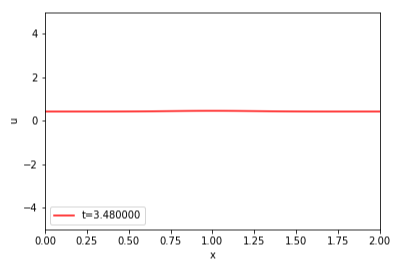
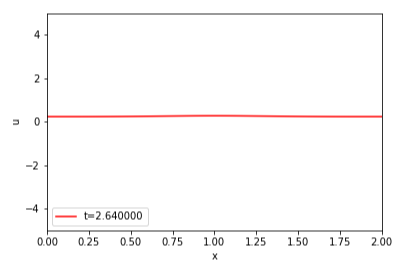
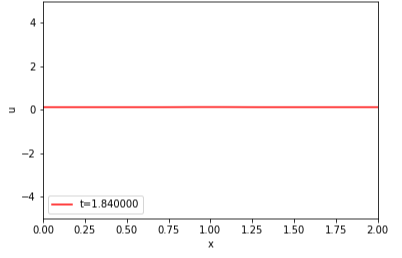
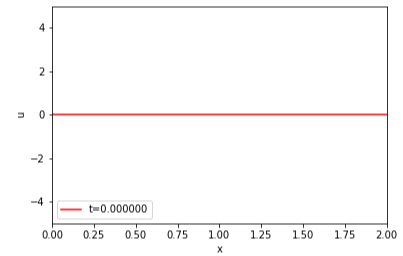
Теперь, когда мы убедились, что построенная схема и её программная реализация способны моделировать колебания с незакрепленными концами (и делать это с достаточно большой точностью), можно протестировать промоделировать процесс в зависимости от внешнего сосредоточенного импульса.





На статичных кадрах может быть плохо видно, но при плавной анимации заметно, что сначала вверх идет небольшая область на отрезке [0.3,0.7], что соответствует сосредоточению там импульса, остальная часть стержня как бы подтягивается за ним

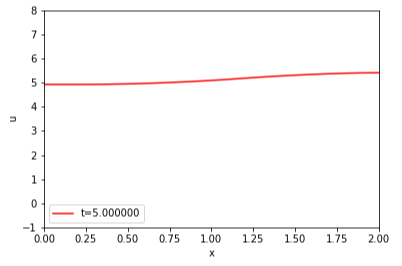
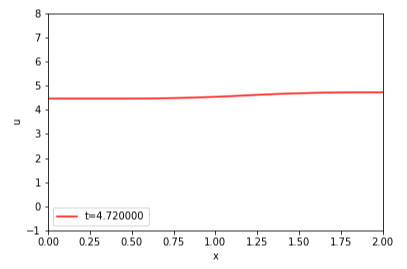
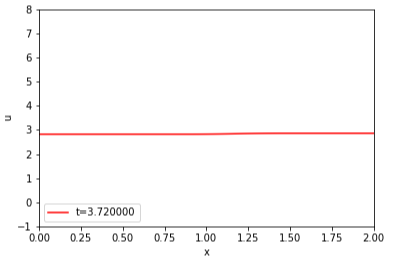
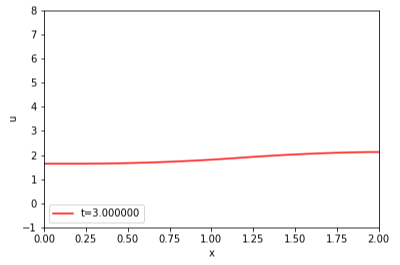
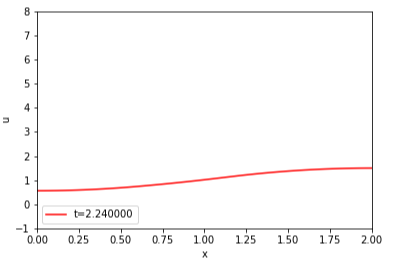
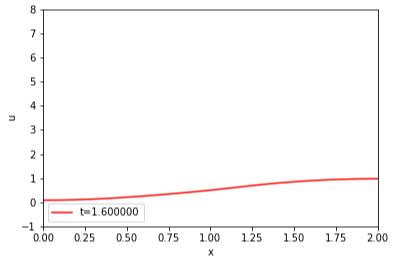
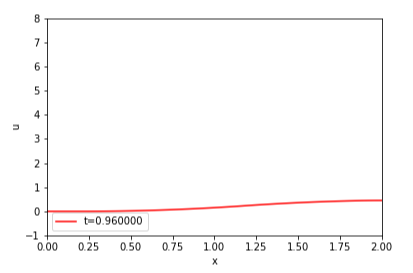
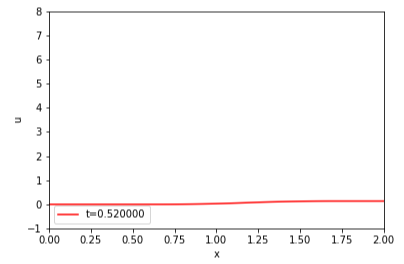
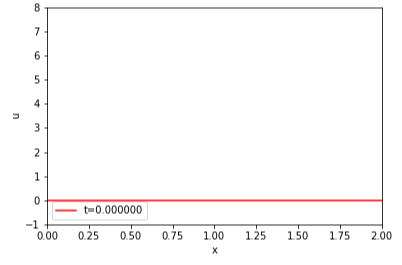
При получаем следующий результат:



Ситуация аналогична предыдущему примеру, только теперь из-за того, что область действия импульса сместилась к центру, стало ещё сложнее увидеть, что стержень не просто поднимается (всегда заметней, когда один из краёв идет выше)

Увеличим величину импульса и сместим центр области его действия к правому краю

При получаем следующий результат:



На этом примере хорошо видно, как сначала поднимается правый край стержня, затем левых край его догоняет, после чего правый снова выходит выше и так далее.

1. **Вывод**

Построенная разностная схема успешно решает задачу моделирования нестационарных процессов для второй краевой задачи. Схема имеет второй порядок точности, поэтому погрешность достаточно быстро уменьшается с уменьшением шага по временной переменно и пространственной. Однако стоит схема является условно устойчивой, поэтому необходимо выполнение условия устойчивости (17).

**Приложение**

1. **Используемая литература**
2. Самарский А.А. «Введение в теорию разностных схем». М:Наука,1989
3. <https://mipt.ru/upload/KolesnikovaSI.pdf>
4. <http://www.mmcs.sfedu.ru/jdownload/finish/16-kafedra-vychislitelnoj-matematiki-i-matematicheskoj-fiziki/1419-uravneniya-matematicheskoj-fiziki-zadachi-i-resheniya-s-v-revina-l-i-sazonov-o-a-tsyvenkova>
5. **Код программы**

import glob, os

import numpy as np

import matplotlib.pyplot as plt

import math

import time

def solveRS(dx,dt,L,a,f,psi,fi,mu1,mu2,T, Action = None, solution = None):

gamma = a\*dt/dx

M = int(round(T/dt))

C2 = gamma\*\*2

N = int(L/dx)

y = np.zeros(N + 1)

y\_1 = np.zeros(N + 1)

y\_2 = np.zeros(N + 1)

x = np.linspace(0, L, N+1)

t = np.linspace(0, M\*dt, M+1)

E = 0

for i in range(N+1): #нулевой слой

y\_1[i] = psi(x[i])

if Action is not None:

Action(y\_1,x,t,0)

if solution != None:

u\_e = solution(x, t[0])

E = max (np.abs(y\_1 - u\_e).max(), E)

for i in range(1,N): #первый слой

y[i] = y\_1[i] +dt\*fi(x[i]) + 0.5\*C2\*(y\_1[i+1] - 2\*y\_1[i] + y\_1[i-1])+dt\*dt/2\*f(x[i],t[0])

y[0] = y\_1[0] +dt\*fi(x[0]) + C2\*(y\_1[1] - y\_1[0] - dx\*mu1(t[0]))+dt\*dt/2\*f(x[0],t[0])

y[N] = y\_1[N] +dt\*fi(x[N]) + C2\*(y\_1[N-1] - y\_1[N] + dx\*mu2(t[0]))+dt\*dt/2\*f(x[N],t[0])

if Action is not None:

Action(y,x,t,1)

if solution != None:

u\_e = solution(x, t[1])

E = max (np.abs(y - u\_e).max(), E)

y\_2[:], y\_1[:] = y\_1, y #переносим слои

for n in range(1,M):

# Пересчитываем значения во внутренних узлах сетки на слое n+1

for i in range(1, N):

y[i] = 2\*y\_1[i] - y\_2[i] + C2\*(y\_1[i+1] - 2\*y\_1[i] + y\_1[i-1])+dt\*dt\*f(x[i],t[n])

y[0] = 2\*y\_1[0] - y\_2[0] + 2\*C2\*(y\_1[1] - y\_1[0] - dx\*mu1(t[n]))+dt\*dt\*f(x[0],t[n])

y[N] = 2\*y\_1[N] - y\_2[N] + 2\*C2\*(y\_1[N-1] - y\_1[N] + dx\*mu2(t[n]))+dt\*dt\*f(x[N],t[n])

if Action is not None:

if Action(y,x,t,n+1):

break

if solution != None:

u\_e = solution(x, t[n+1])

E = max (np.abs(y - u\_e).max(), E)

# Изменяем переменные перед переходом на следующий

# временной слой

y\_2[:], y\_1[:] = y\_1, y

return y, x, t, E

def viz(

dx,dt,L,a,f,psi,fi,mu1,mu2,T, # Параметры задачи

umin, umax, # Интервал для отображения u

animate=True, # Расчет с анимацией

solution = None

):

class PlotMatplotlib:

def \_\_call\_\_(self, u, x, t, n):

"""Функция user\_action для солвера."""

if n == 0:

plt.ion()

self.lines = plt.plot(x, u, 'r-')

plt.xlabel('x'); plt.ylabel('u')

plt.axis([0, L, umin, umax])

plt.legend(['t=%f' % t[n]], loc='lower left')

else:

self.lines[0].set\_ydata(u)

plt.legend(['t=%f' % t[n]], loc='lower left')

plt.draw()

if t[n] == 0:

time.sleep(2)

else:

time.sleep(0.2)

plt.savefig('frame\_%04d.png' % n) # для генерации видео

# Удаляем старые кадры

for filename in glob.glob('frame\_\*.png'):

os.remove(filename)

# Вызываем солвер и выполняем расчет

plot = PlotMatplotlib()

if animate:

Action = plot

else:

Action = None

u, x, t, E = solveRS(

dx,dt,L,a,f,psi,fi,mu1,mu2,T, Action,solution)

# Генерируем видео файлы

cmd = 'ffmpeg -r 24 -i frame\_%04d.png -c:v libx264 movie\_5.mp4'

os.system(cmd)

return E

def test1():

gamma = 0.25

L = math.pi

a = 1

N = 40 # Используем грубую сетку

dx = L/N

dt = dx\*gamma/a

T = 7

def u\_exact(x, t):

return 1.5\*np.sin(x+t)-0.5\*np.cos(x+t)-1.5\*np.sin(x-t)-0.5\*np.cos(x-t)+np.cos(x)\*((np.e)\*\*(-3\*t))

def psi(x):

return 0

def fi(x):

return 0

def mu1(t):

return 0

def mu2(t):

return 0

def f(x, t):

return 10\*math.e\*\*(-3\*t)\*math.cos(x)

umax = 5

umin = -umax

E = viz(dx,dt,L,a,f,psi,fi,mu1,mu2,T, umin, umax,

animate=True, solution = u\_exact)

return E, dt, dx

def test2():

L = math.pi

gamma = 1

a = 1

N = 160

dx = L/N

dt = dx\*gamma/a

T = 4

def u\_exact(x, t):

return -1/3\*np.cos(4\*(x+t))-1/3\*np.cos(4\*(x-t))+2/3\*np.cos(2\*t)\*np.cos(4\*x)

def psi(x):

return 0

def fi(x):

return 0

def mu1(x):

return 0

def mu2(x):

return 0

def f(x, t):

return 8\*math.cos(2\*t)\*math.cos(4\*x)

umax = 5

umin = -umax

E = viz(dx,dt,L,a,f,psi,fi,mu1,mu2,T, umin, umax,

animate=True, solution = u\_exact)

return E, dt, dx

def text\_impuls\_1():

L = 2

gamma = 1

a = 1

N = 100

dx = L/N

dt = dx\*gamma/a

T = 10

def psi(x):

return 0

def fi(x):

return 0

def mu1(x):

return 0

def mu2(x):

return 0

eps = 0.2

ksi = 1/2

def f(x, t):

return g(x,ksi,eps)

def g(x,ksi,eps):

if ksi - eps <= x and x <= ksi + eps:

return 2\*eps

else:

return 0

umax = 5

umin = -umax

E = viz(dx,dt,L,a,f,psi,fi,mu1,mu2,T, umin, umax,

animate=True)

return E, dt, dx

def text\_impuls\_2():

L = 2

gamma = 1

a = 1

N = 50

dx = L/N

dt = dx\*gamma/a

T = 7

def psi(x):

return 0

def fi(x):

return 0

def mu1(x):

return 0

def mu2(x):

return 0

eps = 0.2

ksi = 1

def f(x, t):

return g(x,ksi,eps)

def g(x,ksi,eps):

if ksi - eps < x and x < ksi + eps:

return 2\*eps

else:

return 0

umax = 5

umin = -umax

E = viz(dx,dt,L,a,f,psi,fi,mu1,mu2,T, umin, umax,

animate=True)

return E, dt, dx

def text\_impuls\_3():

L = 2

gamma = 0.5

a = 1

N = 50

dx = L/N

dt = dx\*gamma/a

T = 10

def psi(x):

return 0

def fi(x):

return 0

def mu1(x):

return 0

def mu2(x):

return 0

eps = 0.02

ksi = 2/3

def f(x, t):

return g(x,ksi,eps)

def g(x,ksi,eps):

if ksi - eps < x and x < ksi + eps:

return 2\*eps

else:

return 0

umax = 8

umin = -1

E = viz(dx,dt,L,a,f,psi,fi,mu1,mu2,T, umin, umax,

animate=True)

return E, dt, dx

def text\_impuls\_4():

L = 2

gamma = 0.5

a = 1

N = 50

dx = L/N

dt = dx\*gamma/a

T = 10

def psi(x):

return 0

def fi(x):

return 0

def mu1(x):

return 0

def mu2(x):

return 0

eps = 0.02

ksi = 2/3

def f(x, t):

return g(x,ksi,eps)

def g(x,ksi,eps):

if ksi - eps < x and x < ksi + eps:

return 2\*eps

else:

return 0

umax = 5

umin = -umax

E = viz(dx,dt,L,a,f,psi,fi,mu1,mu2,T, umin, umax,

animate=True)

return E, dt, dx

E, dt, dx = text\_impuls\_3()

print('Погрешность = ',E)

print('Шаг по времени = ', dt)

print('Шаг по пространству = ', dx)