**ЛАБОРАТОРНАЯ РАБОТА 7**

## ЧИСЛЕННОЕ РЕШЕНИЕ ЗАДАЧ ДЛЯ ОБЫКНОВЕННЫХ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

Вариант 20

Студент: Маркаров М.Г.

Преподаватель: ст.преп.Крупин Г.В.

**Задача 7.1.** Найти решения краевой задачи

 (7.1)

### ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ

1. Найти аналитическое решение задачи - функцию *u(x)* при f(x)=0 ( отсутствии источника тепла).

(см. **ПРИЛОЖЕНИE 7.B**).

2. Cоставить разностную схему и выписать коэффициенты матрицы системы уравнений и коэффициенты

правой части.

3. Провести расчет по разностной схеме с шагом h=(b-a)/10 и найти вектор приближенного решения *y.*

Для решения системы уравнений можно воспользоваться встроенной процедурой **linalg.solve** модуля **Numpy.**

4. Построить на одном чертеже графики приближенного и аналитического решений.

Определить величину погрешности как .

5. Составить отчет по задаче.

Решение задачи:

−u(x)'' +1.69\*u(x)=0

u(0)=6, u(3)=2

1) Решаем аналитически:

u(x)= e^(-1.3 x) (5.96196 + 0.038041 e^(2.6 x)) из задачи на собственные значения

Решим все остальное используя пакеты numpy и matplotlib:

import numpy as np

import matplotlib.pyplot as plt

a = 0

b = 3

q = 1.69

ua = 6

ub = 2

N = 10

x = np.linspace(a, b, N+1)

def analytical\_solution(x):

return np.exp(-1.3 \* x) \* (5.96196 + 0.038041 \* np.exp(2.6 \* x))

def num\_solution(N, a, b, ua, ub, q):

h = (b - a) / N

A = np.zeros((N+1, N+1)

b = np.zeros(N+1)

A[0, 0] = 1

b[0] = ua

for i in range(1, N):

A[i, i-1] = 1

A[i, i] = -2 - h\*\*2 \* q

A[i, i+1] = 1

A[N, N] = 1

b[N] = ub

y = np.linalg.solve(A, b)

return y

y\_numerical = num\_solution(N, a, b, ua, ub, q)

plt.figure(figsize=(10, 6))

plt.plot(x, y\_numerical, label='Численное решение', marker='o',linestyle='dashed')

plt.plot(x, analytical\_solution(x), label='Аналитическое решение', linestyle='dotted')

plt.xlabel('x')

plt.ylabel('u(x)')

plt.title('Аналитическое и численное решения')

plt.legend()

plt.grid(True)

plt.show()

err = np.abs(analytical\_solution(x) - y\_numerical)

max\_err = np.max(err)

print("Максимальная величина погрешности:", max\_err)

Результат работы программы:

Максимальная величина погрешности: 0.015173601229399258

Изображение выглядит как линия, текст, График, диаграмма

Автоматически созданное описание

Как мы видим решения очень близки, почти незаметно различие.

**Задача 7.2.** Промоделировать стационарное распределение температуры в стержне в зависимости от правой части уравнения (внешнего источника тепла).

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ

Воспользоваться программой из задачи 1 для уравнения с правой частью f(x) из индивидуального варианта.

**Промоделировать распределение температуры в зависимости от источника тепла следующим образом.**

1. Пусть источник мощности  помещается в точку с на отрезке [a,b], то есть плотность источника задается формулой: . Найти приближенное решение задачи при шаге со значениями W и с

из индивидуального варианта.

2. Подобрать мощность  таким образом, чтобы в выбранной точке c значение температуры увеличилось бы примерно в 3 раза.

3. Построить три графика распределения температуры: при *f(x)=0, f(x)* c первоначальной мощностью , и график температуры из п. 2.

4. Для каждого из трех случах ( п. 2) найти тепловую энергию стержня как значение ,где y(x) – приближенное решение задачи.

4. Оформить отчет по задаче.

Решение:

***Сначала детально рассмотрим пункт 2 и лишь потом решим пункты 1-3:Нам надо увеличить температуру в точке с втрое. Тогда мы введем функцию разницы температур от W: temperature\_difference(W):***

def temperature\_difference(W):

x, y = finite\_difference\_heat(N, a, b, ua, ub, q, c, W)

u\_c = np.interp(c, x, y)

return u\_c - ua \* 3

*Конечная точка сигнализирует о необходимости увеличения мощности источника тепла таким образом, чтобы температура в точке c увеличилась примерно в 3 раза. В уравнении u\_c - ua \* 3, u\_c представляет температуру в точке c, а ua представляет температуру в левом краевом условии (точка a). Таким образом, мы сравниваем разницу между текущей температурой в точке c и температурой в левом краевом условии с тройным значением температуры в левом краевом условии. Если температура в точке c увеличивается втрое, это означает, что разница между ними равна двум температурам в левом краевом условии, т.е. (u\_c - ua) = ua \* 2.*

*Данный ход ,неформально говоря, является костылем, однако я пока не вижу как его обойти*

*Вариант 2 : итерация по W=W+dW , dW где-то 0.1 0.01 . В общем прогон множества значений W.(не реализован ввиду простоты),0*

Теперь приступим к реализации:

import numpy as np

import matplotlib.pyplot as plt

# правая часть уравнения теплопроводности

def source\_term(x, c, W):

return W \* np.exp(-(x - c)\*\*2)

# решение уравнения теплопроводности методом конечных разностей

def finite\_difference\_heat(N, a, b, ua, ub, q, c, W):

h = (b - a) / N

x = np.linspace(a, b, N+1)

A = np.zeros((N+1, N+1))

b = np.zeros(N+1)

A[0, 0] = 1

b[0] = ua

for i in range(1, N):

A[i, i-1] = 1

A[i, i] = -2 - h\*\*2 \* q

A[i, i+1] = 1

b[i] = -h\*\*2 \* source\_term(x[i], c, W)

A[N, N] = 1

b[N] = ub

y = np.linalg.solve(A, b)

return x, y

a, b = 0, 3

ua, ub = 6, 2

q = 1.69

N = 100

c = 2.2

# 1. Решение уравнения теплопроводности при отсутствии источника тепла

W\_no\_heat\_source = 0

x\_no\_heat\_source, y\_no\_heat\_source = finite\_difference\_heat(N, a, b, ua, ub, q, c, W\_no\_heat\_source)

# 2. Решение уравнения теплопроводности при источнике тепла с первоначальной мощностью

W\_original\_heat\_source = 18

x\_original\_heat\_source, y\_original\_heat\_source = finite\_difference\_heat(N, a, b, ua, ub, q, c, W\_original\_heat\_source)

# 3. Подбор мощности источника тепла для увеличения температуры в точке c втрое

def temperature\_difference(W):

x, y = finite\_difference\_heat(N, a, b, ua, ub, q, c, W)

u\_c = np.interp(c, x, y)

return u\_c - ua \* 3

def bisection\_method\_for\_W(func, a, b, tol=1e-6, max\_iter=1000):

iter = 0

while (b - a) / 2 > tol and iter < max\_iter:

c = (a + b) / 2

if func(c) == 0:

return c

elif func(c) \* func(a) < 0:

b = c

else:

a = c

iter += 1

return (a + b) / 2

W\_lower = 0

W\_upper = 100

W\_adjusted = bisection\_method\_for\_W(temperature\_difference, W\_lower, W\_upper)

x\_adjusted, y\_adjusted = finite\_difference\_heat(N, a, b, ua, ub, q, c, W\_adjusted)

# 4. Построение графиков

plt.figure(figsize=(10, 6))

# График при отсутствии источника тепла

plt.plot(x\_no\_heat\_source, y\_no\_heat\_source, label='Отсутствие источника тепла')

# График при первоначальной мощности источника тепла

plt.plot(x\_original\_heat\_source, y\_original\_heat\_source, label=f'Источник тепла (W={W\_original\_heat\_source})')

# График при скорректированной мощности источника тепла

plt.plot(x\_adjusted, y\_adjusted, label=f'Скорректированная мощность (W={W\_adjusted})')

# Отметим точку c на графиках при оригинальной и скорректированной мощностях

u\_c\_original = np.interp(c, x\_original\_heat\_source, y\_original\_heat\_source)

u\_c\_adjusted = np.interp(c, x\_adjusted, y\_adjusted)

plt.scatter(c, u\_c\_original, color='red', label=f'Точка c (W={W\_original\_heat\_source})')

plt.scatter(c, u\_c\_adjusted, color='blue', label=f'Точка c (W={W\_adjusted})')

plt.xlabel('x')

plt.ylabel('Температура')

plt.title('Распределение температуры в стержне')

plt.legend(loc='upper left')

plt.grid(True)

plt.show()

Итого имеем:

Изображение выглядит как линия, График, диаграмма, текст

Автоматически созданное описание

Пункт 4 . Тепловая энергия стержня: интеграл будем брать по формуле Симпсона

Реализация на python:

from scipy.integrate import simps

# 5. Вычисление тепловой энергии стержня через интеграл

def thermal\_energy(x, y):

return simps(y\*\*2, x)

# Тепловая энергия для случая без источника тепла

energy\_no\_heat\_source = thermal\_energy(x\_no\_heat\_source, y\_no\_heat\_source)

print(f"Тепловая энергия без источника тепла: {energy\_no\_heat\_source}")

# Тепловая энергия для случая с первоначальной мощностью источника тепла

energy\_original\_heat\_source = thermal\_energy(x\_original\_heat\_source, y\_original\_heat\_source)

print(f"Тепловая энергия с первоначальной мощностью источника тепла: {energy\_original\_heat\_source}")

# Тепловая энергия для случая с корректированной мощностью источника тепла

energy\_adjusted\_heat\_source = thermal\_energy(x\_adjusted, y\_adjusted)

print(f"Тепловая энергия с корректированной мощностью источника тепла: {energy\_adjusted\_heat\_source}")

Итого:

Тепловая энергия без источника тепла: 16.385352749303593

Тепловая энергия с первоначальной мощностью источника тепла: 76.84041058473377

Тепловая энергия с корректированной мощностью источника тепла: 527.2270135876835

**Задача 7.3.** Дана задача Коши для двух систем обыкновенных дифференциальных уравнений с постоянными коэффициентами:

 ,

 , **(7.2)**

где  и  – заданные матрицы,  - заданные векторы. Исследовать поведение решения систем уравнений на устойчивость.

**ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ**:

1. Используя встроенную функцию **linalg**.**eig**(*M*) (*M* – матрица) модуля **NUMPy** для нахождения собственных чисел матриц *A* и *B*, найти коэффициенты жесткости обеих систем. Установить какая задача является жесткой.

2.Решить обе системы уравнений явным и неявным методами Эйлера с шагом h=0.01

3. Решить систему уравнений, используя встроенные средства Python c шагом h=0.01

4. Для жесткой задачи экспериментально подобрать шаг *h*, при котором графики компонент решения, полученного по неявному методу, визуально совпадают с графиками компонент решения встроенным методом.

5. Построить графики найденных решений.

Решение задачи:

1.Собственные числа- через eigenval:

import numpy as np

Y0 = np.array([7.2, 8, 5.2, 8])

Z0 = np.array([10.4, 5.2, 8.4, 5.2])

A = np.array([[-9.769, -0.19, 1.4, -1.102],

[-0.259, -7.833, -1.166, 0.967],

[-0.945, 1.179, -2.007, 6.836],

[-1.501, 0.967, -6.76, -2.391]])

B = np.array([[-59.993, -19.683, -22.495, 138.156],

[110.216, -50.679, -55.251, -57.296],

[-79.09, 34.35, -268.27, 26.665],

[39.72, -130.06, -67.731, -187.06]])

eigenvalues\_A = np.linalg.eig(A)[0]

print("Собственные числа матрицы A:")

print(eigenvalues\_A)

eigenvalues\_B = np.linalg.eig(B)[0]

print("\nСобственные числа матрицы B:")

print(eigenvalues\_B)

max\_abs\_real\_A = np.max(np.abs(np.real(eigenvalues\_A)))

min\_abs\_real\_A = np.min(np.abs(np.real(eigenvalues\_A)))

max\_abs\_real\_B = np.max(np.abs(np.real(eigenvalues\_B)))

min\_abs\_real\_B = np.min(np.abs(np.real(eigenvalues\_B)))

stiffness\_A = max\_abs\_real\_A / min\_abs\_real\_A

stiffness\_B = max\_abs\_real\_B / min\_abs\_real\_B

print("Число жесткости для матрицы A:", stiffness\_A)

print("Число жесткости для матрицы B:", stiffness\_B)

Имеем:

Собственные числа матрицы A:

[ -2.00005221+7.00023619j -2.00005221-7.00023619j

-10.00017505+0.j -7.99972054+0.j ]

Собственные числа матрицы B:

[ -3.00071223+89.99983032j -3.00071223-89.99983032j

-280.00028777+70.00010026j -280.00028777-70.00010026j]

Число жесткости для матрицы A: 4.999957008579366

Число жесткости для матрицы B: 93.31127623938136

Система Z’=B\*Z является жесткой т.к SB>>1 = 93

***Сначала решим обе системы через odeint из пакета SciPy(п.3)***

***Шаг odeint мы ограничим как hmax=0.01***

odeint использует адаптивные методы интегрирования по времени для численного решения системы ОДУ. Он автоматически адаптирует размер временного шага в зависимости от требуемой точности решения.**Но мы не дадим ему адаптировать шаг.**

Реализация:

import numpy as np

from scipy.integrate import odeint

import matplotlib.pyplot as plt

Z0 = np.array([10.4, 5.2, 8.4, 5.2])

B = np.array([[-59.993, -19.683, -22.495, 138.156],

[110.216, -50.679, -55.251, -57.296],

[-79.09, 34.35, -268.27, 26.665],

[39.72, -130.06, -67.731, -187.06]])

# Вектор-функция Z

def Z\_func(Z, t):

return np.dot(B, Z)

h = 0.01

steps = 1000

t = np.linspace(0, h \* steps, steps + 1)

# Решение системы уравнений с использованием odeint

Z\_solution = odeint(Z\_func, Z0, t, hmax=h)

# Построение графиков компонент вектор-функции Z

plt.figure(figsize=(12, 10))

for i in range(4):

plt.subplot(2, 2, i+1)

plt.plot(t, Z\_solution[:, i], label='Z{}'.format(i+1))

plt.title('Компонента Z{}'.format(i+1))

plt.legend()

plt.tight\_layout()

plt.show() Это строит нам графики всех компонент вектор функции Z(t)={Z1(t),Z2(t),Z3(t),Z4(t)}.

Для Y(t) ситуация та же самая( в смысле реализации)Изображение выглядит как текст, линия, диаграмма, Параллельный

Автоматически созданное описание

То же самое получаем для системы с Z(t):

Изображение выглядит как текст, линия, диаграмма, График

Автоматически созданное описание

Теперь реализуем явный метод Эйлера для системы с Y:

import numpy as np

import matplotlib.pyplot as plt

Y0 = np.array([7.2, 8, 5.2, 8])

A = np.array([[-9.769, -0.19, 1.4, -1.102],

[-0.259, -7.833, -1.166, 0.967],

[-0.945, 1.179, -2.007, 6.836],

[-1.501, 0.967, -6.76, -2.391]])

h = 0.01

steps = 1000

def explicit\_euler\_Y(A, Y0, h, steps):

Y\_values = [Y0]

for \_ in range(steps):

Y\_next = Y\_values[-1] + h \* np.dot(A, Y\_values[-1])

Y\_values.append(Y\_next)

return np.array(Y\_values)

Y\_explicit = explicit\_euler\_Y(A, Y0, h, steps)

plt.figure(figsize=(12, 10))

for i in range(4):

plt.subplot(2, 2, i+1)

plt.plot(Y\_explicit[:, i])

plt.title('Явный метод Эйлера для Y{}'.format(i+1))

plt.tight\_layout()

plt.show()

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание

Для системы с Z(t): то есть жесткой мы ожидаем что явный метод сломается:

import numpy as np

import matplotlib.pyplot as plt

Z0 = np.array([10.4, 5.2, 8.4, 5.2])

B = np.array([[-59.993, -19.683, -22.495, 138.156],

[110.216, -50.679, -55.251, -57.296],

[-79.09, 34.35, -268.27, 26.665],

[39.72, -130.06, -67.731, -187.06]])

h = 0.01

steps = 1000

def explicit\_euler\_Z(B, Z0, h, steps):

Z\_values = [Z0]

for \_ in range(steps):

Z\_next = Z\_values[-1] + h \* np.dot(B, Z\_values[-1])

Z\_values.append(Z\_next)

return np.array(Z\_values)

Z\_explicit = explicit\_euler\_Z(B, Z0, h, steps)

# Вывод графиков

plt.figure(figsize=(12, 10))

for i in range(4):

plt.subplot(2, 2, i+1)

plt.plot(Z\_explicit[:, i])

plt.title('Явный метод Эйлера для Z{}'.format(i+1))

plt.tight\_layout()

plt.show()

Изображение выглядит как текст, диаграмма, линия, График

Автоматически созданное описание

Видно что что-то не так ведь осцилляции совсем не там.

Теперь неявный метод Эйлера: нам нужно решать нелинейное уравнение на каждом шаге поэтому подключаем scipy.fsolve

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import fsolve

Y0 = np.array([7.2, 8, 5.2, 8])

A = np.array([[-9.769, -0.19, 1.4, -1.102],

[-0.259, -7.833, -1.166, 0.967],

[-0.945, 1.179, -2.007, 6.836],

[-1.501, 0.967, -6.76, -2.391]])

h = 0.01

steps = 1000

def implicit\_euler\_Y(A, Y0, h, steps):

Y\_values = [Y0]

for \_ in range(steps):

Y\_next = fsolve(lambda Y: Y - Y\_values[-1] - h \* np.dot(A, Y), Y\_values[-1])

Y\_values.append(Y\_next)

return np.array(Y\_values)

Y\_implicit = implicit\_euler\_Y(A, Y0, h, steps)

plt.figure(figsize=(12, 10))

for i in range(4):

plt.subplot(2, 2, i+1)

plt.plot(Y\_implicit[:, i])

plt.title('Неявный метод Эйлера для Y{}'.format(i+1))

plt.tight\_layout()

plt.show()

Изображение выглядит как текст, линия, диаграмма, Параллельный

Автоматически созданное описание

Теперь для Z(t):

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import fsolve

Z0 = np.array([10.4, 5.2, 8.4, 5.2])

B = np.array([[-59.993, -19.683, -22.495, 138.156],

[110.216, -50.679, -55.251, -57.296],

[-79.09, 34.35, -268.27, 26.665],

[39.72, -130.06, -67.731, -187.06]])

h = 0.01

steps = 1000

def implicit\_euler\_Z(B, Z0, h, steps):

Z\_values = [Z0]

for \_ in range(steps):

Z\_next = fsolve(lambda Z: Z - Z\_values[-1] - h \* np.dot(B, Z), Z\_values[-1])

Z\_values.append(Z\_next)

return np.array(Z\_values)

Z\_implicit = implicit\_euler\_Z(B, Z0, h, steps)

plt.figure(figsize=(12, 10))

for i in range(4):

plt.subplot(2, 2, i+1)

plt.plot(Z\_implicit[:, i])

plt.title('Неявный метод Эйлера для Z{}'.format(i+1))

plt.tight\_layout()

plt.show()

Изображение выглядит как текст, линия, диаграмма, График

Автоматически созданное описание

***Примечание: вообще неявный метод очень хорошо работает для жестких систем но у нас очень большое число осцилляций так что даже он тут не может до конца дать хорошее решение. (Im(lambda)>>50 очень быстро колеблящиеся синусы). Но при менее активно осциллирующих функциях неявный метод Эйлера даст очень хорошее решение по сравнению с явным.***

П.4 Подбор шага чтобы решения были похожи при odeint и неявном методе Эйлера для жесткой системы:

Нам понадобится fsolve и odeint из SciPy. fsolve для неявного метода Эйлера. odeint как встроенная функция. Переберем штуки 4 шагов и оценим визуально:

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import fsolve

from scipy.integrate import odeint

Z0 = np.array([10.4, 5.2, 8.4, 5.2])

B = np.array([[-59.993, -19.683, -22.495, 138.156],

[110.216, -50.679, -55.251, -57.296],

[-79.09, 34.35, -268.27, 26.665],

[39.72, -130.06, -67.731, -187.06]])

def implicit\_euler\_Z(B, Z0, h, steps):

Z\_values = [Z0]

for \_ in range(steps):

Z\_next = fsolve(lambda Z: Z - Z\_values[-1] - h \* np.dot(B, Z), Z\_values[-1])

Z\_values.append(Z\_next)

return np.array(Z\_values)

def system(Z, t):

return np.dot(B, Z)

h\_values\_to\_try = [0.001, 0.0005, 0.0001, 0.00005]

for h in h\_values\_to\_try:

steps = int(1 / h)

t = np.linspace(0, 1, steps+1)

Z\_implicit = implicit\_euler\_Z(B, Z0, h, steps)

Z\_builtin = odeint(system, Z0, t)

plt.figure(figsize=(12, 10))

for i in range(4):

plt.subplot(2, 2, i+1)

plt.plot(t, Z\_implicit[:, i], label=f'Неявный, h={h}')

plt.plot(t, Z\_builtin[:, i], label='Встроенный')

plt.title(f'Сравнение графиков для Z{i+1}')

plt.legend()

plt.tight\_layout()

plt.show()

На выход мы получим 4 блока в каждом из которых 4 графика ( по графику на компоненту):

Шаг 0.001- исходный

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание

Видно что близко но не совсем

Шаг 0.0005

Изображение выглядит как текст, снимок экрана, линия, График

Автоматически созданное описание

Уже ближе но все равно есть ощутимые различия

Шаг 0.0001

Изображение выглядит как текст, снимок экрана, линия, График

Автоматически созданное описание

Почти прекрасно, но все же проверим последний шаг h=0.00005:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Автоматически созданное описание

Различия почти не видны (если не зумить)

То есть при шаге h=5e^-5 почти не отличить.

***Заключение:***

***Был проведен вычислительный эксперимент, заключающийся в исследовании жестких систем , влиянии жесткости системы на работу численных методов. Минусом эксперимента было большое число колебаний в системе с Z(t) и это повлияло на сравнение того как работает явный и неявный метод Эйлера для жесткой системы( при равном шаге), но из теоретический знаний можно сказать что явный метод будет работать не хорошо( кроме необоснованно малых шагов) в отличии от неявного метода.***