**ЛАБОРАТОРНАЯ РАБОТА 8**

## ЧИСЛЕННОЕ РЕШЕНИЕ ЗАДАЧ ТЕПЛОПРОВОДНОСТИ

Вариант 20

Студент: Маркаров М.Г.

Преподаватель: ст.преп.Крупин Г.В.

**Задача 8.1**  Промоделировать установление процесса теплопроводности в стержне. Для этого рассмотреть две задачи: начально-краевую (8.1)

(8.1)

и краевую (8.2)

(8.2)

Очевидно, что при .

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Найти аналитическое решение задачи **(8.2)** см. **ПРИЛОЖЕНИЕ 8 B**

2. Cоставить явную разностную схему и программу, реализующую нахождение приближенного решения задачи (8.1).

3. Взять шаг разностной схемы и выбрать шаг из условия устойчивости для явной разностной схемы. Для начальной отладки принять число слоев по времени .

4. Найти массивы решений задач 8.1 и 8.2.

5. Произвести анимацию процесса установления ( см. приложенный файл)

6. Установить момент времени TN, при котором приближенное решение задачи (8.1) визуально совпадает с решением задачи (8.2)

7. В отчет по задаче включить: нахождение аналитического решения задачи (8.2), программу, реализующую явную разностную схему, профили температуры в моменты времени 0, ТN/4, TN/2, TN.

Приступим к решению задачи 8.1:

f(x)=x^3 + 3

l=1

ua=3

ub=-3

1. Аналитическое решение -u’’=f(x) u(0)=3 , u(1)=-3:

u(x)= (60 - 89 x - 30 x^2 - x^5)/20 используя пакет Wolfram Alpha

2,3. Реализуем явную разностную схему для решения начально краевой задачи: и учтем условие устойчивости tau<= h^2 / 2

import numpy as np

def explicit\_scheme\_solver(l, T, ua, ub, phi, f, h, tau, K):

N = int(l / h) + 1

M = int(T / tau) + 1

U = np.zeros((M, N))

for i in range(N):

U[0, i] = phi(i \* h)

U[:, 0] = ua

U[:, -1] = ub

for n in range(0, M - 1):

for i in range(1, N - 1):

U[n + 1, i] = U[n, i] + tau / h\*\*2 \* (U[n, i - 1] - 2 \* U[n, i] + U[n, i + 1]) + tau \* f(i \* h)

return U

# Test

l = 1

ua = 3

ub = -3

phi = lambda x: ua + (ub - ua) / l \* x

f = lambda x: x\*\*3 + 3

h = l / 10

tau =0.5\* h\*\*2 /2 # берем как половину от tau\_max=h^2 /2 на всякий случай

K = 10

T=K\*tau

U = explicit\_scheme\_solver(l, T, ua, ub, phi, f, h, tau, K)

print(U)

Результатом будет матрица, где каждая строка это решения в момент времени n\*tau, где n-тый слой это по времени.

Имеем послойное решение:

[[ 3. 2.4 1.8 1.2 0.6 0.

-0.6 -1.2 -1.8 -2.4 -3. ]

[ 3. 2.4075025 1.80752 1.2075675 0.60766 0.0078125

-0.59196 -1.1916425 -1.79122 -2.3906775 -3. ]

[ 3. 2.41313375 1.8150475 1.21514625 0.615335 0.01564375

-0.5838975 -1.18325875 -1.78241 -2.38382125 -3. ]

[ 3. 2.41783125 1.82211375 1.22273625 0.623025 0.02349375

-0.5758125 -1.17484875 -1.774195 -2.37819062 -3. ]

[ 3. 2.42194656 1.82871875 1.23022031 0.63073 0.0313625

-0.567705 -1.16656875 -1.76657734 -2.37332156 -3. ]

[ 3. 2.42565547 1.83492109 1.23753984 0.6384207 0.03925

-0.55961406 -1.15849746 -1.75948125 -2.36898262 -3. ]

[ 3. 2.42906051 1.84077937 1.24467287 0.64606781 0.04713916

-0.5515789 -1.15066506 -1.75283064 -2.36503912 -3. ]

[ 3. 2.4322276 1.84634303 1.25161573 0.65364691 0.05500431

-0.54363092 -1.14307741 -1.74656137 -2.36140472 -3. ]

[ 3. 2.43520206 1.85165235 1.25837285 0.66113847 0.06281865

-0.53579374 -1.13572928 -1.74062122 -2.3580202 -3. ]

[ 3. 2.43801662 1.8567399 1.26495163 0.66852711 0.07055801

-0.52808453 -1.12861088 -1.73496798 -2.35484291 -3. ]

[ 3. 2.44069578 1.86163201 1.27136007 0.67580096 0.07820215

-0.52051548 -1.12171107 -1.72956744 -2.35184095 -3. ]]

4,5,6:

Создадим анимацию и посмотрим на поведение U(x,t) из Н-К задачи и найдем время TN при котором она наложится на решение чисто краевой задачи. Как мы знаем краевая задача в нашем случае это при больших t в Н-К задаче.

Реализация:

import numpy as np

from matplotlib import pyplot as plt

from matplotlib.animation import FuncAnimation

from IPython.display import HTML

l=1

Ua = 3

Ub =-3

def f(x):

return x\*\*3 + 3

def F(x, t):

return f(x) \* (1 - np.exp(-t))

def phi(x):

return Ua + x \* (Ub - Ua)/l

N = 10

h = l / 10

tau = h\*\*2 / 2

def U\_a(x):

return (60 - 89 \* x - 30 \* x\*\*2 - x\*\*5) / 20

def find\_u(N, K):

U = np.zeros([K+1, N+1])

#первая кордината -- t

#вторая координата -- х

for j in range(K+1):

U[j, 0] = Ua

U[j, N] = Ub

for i in range(N+1):

U[0, i] = phi(i \* h)

g = tau / h\*\*2

for j in range(K):

for i in range(1, N):

U[j+1, i] = (g \* U[j, i - 1] + (1 - 2 \* g) \* U[j, i] + g \* U[j, i + 1]+ tau \* F( i \* h, j \* tau))

return U

K = 500

U = find\_u(N, K)

fig, ax = plt.subplots(1, 1, figsize = (6, 5))

x\_data = np.linspace(0, l, N+1, endpoint = True)

line1, = ax.plot(x\_data, U[0])

line2, = ax.plot(x\_data, U\_a(x\_data))

def update1(frame):

if frame % 10 == 0:

print(f"{100 \* frame/K:.0f}%", end=" ")

line2.set\_ydata(U\_a(x\_data))

line1.set\_ydata(U[frame])

ax.set\_xlabel(f"t={frame \* tau:.2f}")

return line1, line2

animation1 = FuncAnimation(fig, update1, frames = range(K+1),

interval = int(5000 / K), blit = False)

HTML(animation1.to\_jshtml())

Число слоев K- подбирается чтобы численное решение Н-К задачи прижималось к решению чисто краевой. В нашем случае это 500. То есть на времени t=500tau решения наложились.

Анимация записана и будет воспроизведена при требовании\*

Имеем: TN=2.50

Построим профили температуры при t=0;TN/4;TN/2;TN

Примечание: мы будем строить при TN/40; TN/20 т.к при половине и четверти и самом TNпочти не видно отличий из-за поведения функции

import numpy as np

import matplotlib.pyplot as plt

def explicit\_scheme\_solver(l, T, ua, ub, phi, f, h, tau, K):

N = int(l / h) + 1

M = int(T / tau) + 1

U = np.zeros((M, N))

for i in range(N):

U[0, i] = phi(i \* h)

U[:, 0] = ua

U[:, -1] = ub

for n in range(0, M - 1):

for i in range(1, N - 1):

U[n + 1, i] = U[n, i] + tau / h\*\*2 \* (U[n, i - 1] - 2 \* U[n, i] + U[n, i + 1]) + tau \* f(i \* h)

return U

# Test parameters

l = 1

ua = 3

ub = -3

phi = lambda x: ua + (ub - ua) / l \* x

f = lambda x: x\*\*3 + 3

h = l / 10

tau = h\*\*2 / 2

K = 500

T = K \* tau

# Compute solution

U = explicit\_scheme\_solver(l, T, ua, ub, phi, f, h, tau, K)

# Time at which we want to plot profiles

TN = 2.5

# Find corresponding time steps

n0 = int(0 / tau)

n1 = int(TN / (40 \* tau))

n2 = int(TN / (20 \* tau))

n3 = int(TN / tau)

# Plot temperature profiles

plt.plot(np.linspace(0, l, U.shape[1]), U[n0], label=f"t = 0")

plt.plot(np.linspace(0, l, U.shape[1]), U[n1], label=f"t = {TN/40:.2f}")

plt.plot(np.linspace(0, l, U.shape[1]), U[n2], label=f"t = {TN/20:.2f}")

plt.plot(np.linspace(0, l, U.shape[1]), U[n3], label=f"t = {TN:.2f}")

plt.title("Temperature Profiles at Different Time Points")

plt.xlabel("Position")

plt.ylabel("Temperature")

plt.legend()

plt.grid(True)

plt.show()

print(U[n0])

print(U[n1])

print(U[n2])

print(U[n3])

Изображение выглядит как линия, График, текст, диаграмма

Автоматически созданное описание

**Задача 8.2.** Найти приближенное решение начально-краевой задачи для одномерного уравнения теплопроводности

(8.3)

### ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ

1. Cоставить неявную разностную схему.

2. Выписать коэффициенты матрицы системы уравнений и коэффициенты правой части.

3. Составить подпрограмму нахождения решения задачи .

4. Найти решение разностной схемы*и*  для шагов и с шагом по времени . Найти решение для 50 временных слоев.

5. Вычислить величину погрешности на 50-ом слое по времени

6. На одном чертеже построить профили температур в моменты времени .

Сначала разберемся с первыми тремя пунктами:

То есть мы будем решать системы методом прогонки и выведем матрицу коэффицентов левой и правой частей:

import numpy as np

from matplotlib import pyplot as plt

def k\_function(x):

return 0.5 + x\*\*2

def phi\_function(x):

return np.sin(np.pi \* x / 4)

def f\_function(x, t):

return 0

def g1\_function(t):

return t / 2

def g2\_function(t):

return 0

def implicit\_scheme\_coefficients(l, T, N, M):

h = l / N

tau = T / M

alpha = tau / h\*\*2

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

B = np.zeros(N-1)

x\_values = np.linspace(h, l-h, N-1)

t\_values = np.linspace(tau, T, M)

for i in range(N-1):

if i == 0:

A[i, i] = 1 + alpha \* (k\_function(x\_values[i] + h/2) + k\_function(x\_values[i] - h/2))

A[i, i+1] = -alpha \* k\_function(x\_values[i] + h/2)

B[i] = phi\_function(x\_values[i]) + alpha \* (k\_function(x\_values[i] + h/2) + k\_function(x\_values[i] - h/2)) \* g1\_function(t\_values[-1])

elif i == N-2:

A[i, i-1] = -alpha \* k\_function(x\_values[i] - h/2)

A[i, i] = 1 + alpha \* (k\_function(x\_values[i] + h/2) + k\_function(x\_values[i] - h/2))

B[i] = phi\_function(x\_values[i]) + alpha \* (k\_function(x\_values[i] + h/2) + k\_function(x\_values[i] - h/2)) \* g2\_function(t\_values[-1])

else:

A[i, i-1] = -alpha \* k\_function(x\_values[i] - h/2)

A[i, i] = 1 + alpha \* (k\_function(x\_values[i] + h/2) + k\_function(x\_values[i] - h/2))

A[i, i+1] = -alpha \* k\_function(x\_values[i] + h/2)

B[i] = phi\_function(x\_values[i])

return A, B

def tridiagonal\_solver(A, B):

# Прямой проход - вычисление прогоночных коэффициентов

n = len(B)

alpha = [0] \* n

beta = [0] \* n

alpha[0] = A[0][0]

beta[0] = B[0]

for i in range(1, n):

alpha[i] = A[i][i] - A[i][i-1] \* A[i-1][i] / alpha[i-1]

beta[i] = B[i] - A[i][i-1] \* beta[i-1] / alpha[i-1]

# Обратный проход - нахождение решения

x = [0] \* n

x[-1] = beta[-1] / alpha[-1]

for i in range(n-2, -1, -1):

x[i] = (beta[i] - A[i][i+1] \* x[i+1]) / alpha[i]

return x

# Test parameters

l = 4

T = 1

N = 10

M = 10

# Compute coefficients

A, B = implicit\_scheme\_coefficients(l, T, N, M)

# Print coefficients

print("Matrix A:")

print(A)

print("\nVector B:")

print(B)

x\_solution = tridiagonal\_solver(A, B)

print("Solution:", x\_solution)

Результат:

Matrix A:

[[ 1.875 -0.5375 0. 0. 0. 0. 0. 0. 0. ]

[-0.5375 2.475 -0.9375 0. 0. 0. 0. 0. 0. ]

[ 0. -0.9375 3.475 -1.5375 0. 0. 0. 0. 0. ]

[ 0. 0. -1.5375 4.875 -2.3375 0. 0. 0. 0. ]

[ 0. 0. 0. -2.3375 6.675 -3.3375 0. 0. 0. ]

[ 0. 0. 0. 0. -3.3375 8.875 -4.5375 0. 0. ]

[ 0. 0. 0. 0. 0. -4.5375 11.475 -5.9375 0. ]

[ 0. 0. 0. 0. 0. 0. -5.9375 14.475 -7.5375]

[ 0. 0. 0. 0. 0. 0. 0. -7.5375 17.875 ]]

Vector B:

[0.74651699 0.58778525 0.80901699 0.95105652 1. 0.95105652

0.80901699 0.58778525 0.30901699]

Solution: [0.5850885778209194, 0.652137840073073, 0.7592221773969474, 0.7921306322022366, 0.7457866173017939, 0.6371596457578265, 0.4880820946690511, 0.32010326725495203, 0.1522682725207915]

Это как решение на каком-то слое.

4. Решения с двумя разными шагами на 50-и слоях+графики

*Примечание: для облегчения задачи не будем каждый раз применять метод прогонки, хотя можно было.*

import numpy as np

from matplotlib import pyplot as plt

def implicit\_scheme(fi, g1, g2, f, k, l, h, tau, M):

N = int(l / h)

x\_values = np.linspace(0, l, N)

t\_values = np.linspace(0, 50\*tau,M)

U = np.zeros((M, N))

U[:, 0] = g1(t\_values)

U[:, -1] = g2(t\_values)

U[0, :] = fi(x\_values)

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

A[0,0] = 1

for i in range(1, N - 1):

gamma = k(x\_values[i])\*tau / h\*\*2

A[i, i - 1] = gamma

A[i, i] = -(1+2\*gamma)

A[i, i + 1] = gamma

A[N-1,N-1] = 1

for j in range(1, M):

F = np.zeros(N)

for i in range(1,N):

F[i] = -U[j-1,i] - tau \* f(x\_values[i], t\_values[j])

F[0] = g1(t\_values[j])

F[-1] = g2(t\_values[j])

U[j,:] = np.linalg.solve(A, F)

time\_instances = [0,9,19, 29, 39, 49] # костыль

fig, ax = plt.subplots(figsize = (12,10))

ax.plot(x\_values, U[time\_instances[0], :], label=f"Time {time\_instances[0]}\*τ")

for i in range (1,len(time\_instances)):

ax.plot(x\_values, U[time\_instances[i], :], label=f"Time {time\_instances[i]+1}\*τ")

ax.set\_title(f'Temperature profiles h = {h}')

ax.set\_xlabel('x', fontsize=12)

ax.set\_ylabel('Temperature', fontsize=12)

ax.legend()

plt.grid(True)

plt.show()

return U

def k(x):

return 0.5 + x\*\*2

def phi(x):

return np.sin(np.pi \* x / 4)

def f(x, t):

return 0

def g1(t):

return t / 2

def g2(t):

return 0

#Test

l = 4

T = 1

N = 10

M = 50

tau = 0.05

h1 = l / 20

h2 = l / 40

U\_h1 = implicit\_scheme(phi, g1, g2, f, k, l, h1, tau, M)

U\_h2 = implicit\_scheme(phi, g1, g2, f, k, l, h2, tau, M)

Результат:

Изображение выглядит как линия, График, диаграмма, скат

Автоматически созданное описание

Изображение выглядит как линия, График, диаграмма, скат

Автоматически созданное описание

5. Погрешности:

U\_norm = 0

for i in range (N):

U\_norm += (U\_h1[-1,i]-U\_h2[-1,2\*i])\*\*2

error = np.sqrt(U\_norm)

print(f"\nDelta profile 50 = {error}\n")

Результат:

Delta profile 50 = 0.04148816588331743

**Задача 8.3.**  Промоделировать процесс теплопроводности для составного стержня

Рассмотреть два случая конфигурации:

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Задать входные данные из задачи 8.2.

2. Воспользовавшись программой задачи 8.2 , получить решение на 50-ом слое по времени.

3.Сравнить полученное решение с решением задачи 8.2.

Решение задачи:

import numpy as np

from matplotlib import pyplot as plt

def implicit\_scheme(fi, g1, g2, f, k, l, h, tau, M, N):

x\_values = np.linspace(0, l, N)

t\_values = np.linspace(0, 50\*tau,M)

U = np.zeros((M, N))

U[:, 0] = g1(t\_values)

U[:, -1] = g2(t\_values)

U[0, :] = fi(x\_values)

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

A[0,0] = 1

for i in range(1, N - 1):

gamma = k(x\_values[i])\*tau / h\*\*2

A[i, i - 1] = gamma

A[i, i] = -(1+2\*gamma)

A[i, i + 1] = gamma

A[N-1,N-1] = 1

for j in range(1, M):

F = np.zeros(N)

for i in range(1,N):

F[i] = -U[j-1,i] - tau \* f(x\_values[i], t\_values[j])

F[0] = g1(t\_values[j])

F[-1] = g2(t\_values[j])

U[j,:] = np.linalg.solve(A, F)

return U

def k(x):

return 0.5 + x\*\*2

def phi(x):

return np.sin(np.pi \* x / 4)

def f(x, t):

return 0

def g1(t):

return t / 2

def g2(t):

return 0

def k1(x):

if x <= l/3:

k1 = k(x)

elif x > l/3 and x <= 2\*l/3:

k1 = 10\*k(x)

elif x > 2\*l/3 and x <= l:

k1 = k(x)

return k1

def k2(x):

if x <= l/3:

k2 = k(x)

elif x > l/3 and x <= 2\*l/3:

k2 = 0.1\*k(x)

elif x > 2\*l/3 and x <= l:

k2 = k(x)

return k2

#Test

l = 4

M = 50

tau = 0.05

h2 = l / 40

h=h2

N = int(l / h)

U\_k = implicit\_scheme(phi, g1, g2, f, k, l, h, tau, M, N)

U\_k1 = implicit\_scheme(phi, g1, g2, f, k1, l, h, tau, M, N)

U\_k2 = implicit\_scheme(phi, g1, g2, f, k2, l, h, tau, M, N)

x\_values = np.linspace(0, l, N)

fig, ax = plt.subplots(figsize=(12,10))

ax.plot(x\_values, U\_k[M-1, :], label=f" k", color='b')

ax.plot(x\_values, U\_k1[M-1, :], label=f"k1", color='r')

ax.plot(x\_values, U\_k2[M-1, :], label=f" k2",color='black')

ax.axvline(x=l/3, color='g', linestyle='dashed')

ax.axvline(x=2\*l/3, color='g', linestyle='dashed')

ax.set\_title(f'Temperature profiles at k')

ax.set\_xlabel('x', fontsize=12)

ax.set\_ylabel('Temperature', fontsize=12)

ax.legend()

plt.grid(True)

plt.show()

Изображение выглядит как линия, График, диаграмма, скат

Автоматически созданное описание