БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

**Лабораторная работа №1**

**Вариант 12**

Выполнил: Белоушко Степан

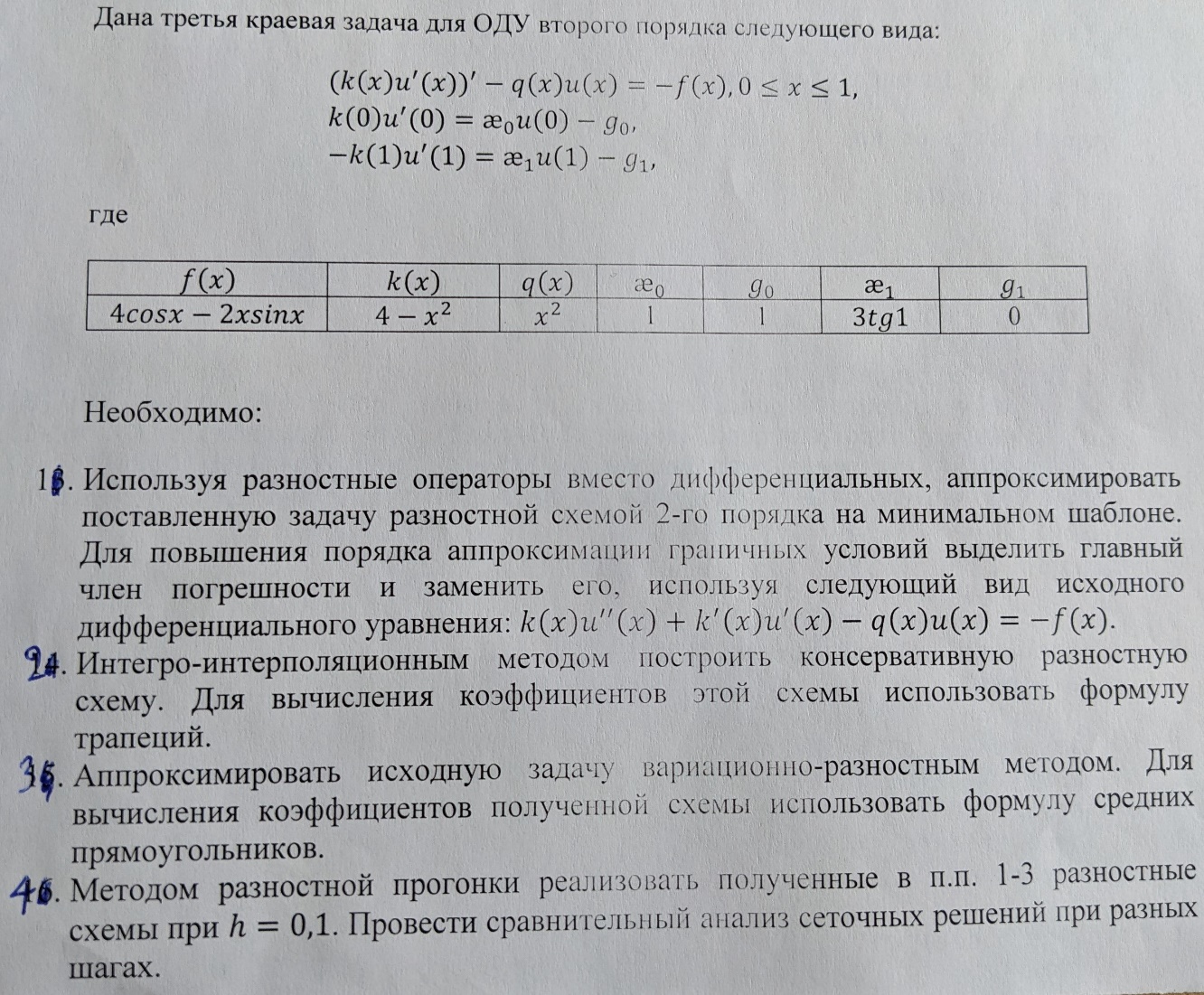
3 курс 7 группа

Преподаватель:

Будник Анатолий Михайлович

Минск, 2023

**Постановка задачи**



**Аппроксимация исходной задачи разностной схемой**

Так как в задаче содержится вторая производная u(x), то минимальный шаблон имеет вид . Аппроксимация исходного уравнения примет вид:

Для граничных условий минимальный шаблон будет иметь вид

Аппроксимация будет иметь вид:

Однако у такой аппроксимации погрешность . Необходимо повысить порядок аппроксимации. Для этого выберем соответствующие коэффициенты Погрешность примет вид:

Соответственно, при

Погрешность

Аналогично для второго граничного условия получим:

Получаем разностную схему для исходной задачи:

**Реализация**

Для вычисления значений функции в узлах сетки необходимо решить СЛАУ:

Решать буду методом прогонки.

**Листинг программы**

def sweep\_method(e: np.ndarray, d: np.ndarray, c: np.ndarray, b: np.ndarray) -> np.ndarray:

n = d.shape[0]

x = np.zeros\_like(d)

for k in range(1, n):

d[k] = d[k] - e[k-1]\*c[k]/d[k-1]

b[k] = b[k] - b[k-1]\*c[k]/d[k-1]

x[n-1] = b[n-1]/d[n-1]

for k in reversed(range(n-1)):

x[k] = (b[k] - e[k]\*x[k+1])/d[k]

return x

def f(x):

return 4\*np.cos(x) - 2\*x\*np.sin(x)

def k(x):

return 4 - x\*\*2

def dk(x):

return -2\*x

def q(x):

return x\*\*2

kappa\_0 = 1

g\_0 = 1

kappa\_1 = 3\*np.tan(1)

g\_1 = 0

def solve\_approx(k, dk, q, f, kappa\_0, g\_0, kappa\_1, g\_1, h=0.1):

X = np.arange(0, 1 + h/2, h)

n = X.shape[0]

D = q(X)

PHI = f(X)

KAPPA\_0 = kappa\_0 \* (1 - (h/2) \* (dk(0)/k(0))) + h/2 \* q(0)

G\_0 = g\_0 \* (1 - (h/2) \* (dk(0)/k(0))) + h/2 \* f(0)

KAPPA\_1 = kappa\_1\*(1 + (h/2) \* (dk(1)/ k(1))) + h/2 \* q(1)

G\_1 = g\_1\*(1 + (h/2) \* (dk(1)/ k(1))) + h/2 \* f(1)

diag\_el = np.zeros(n)

diag\_el[0] = (-k(0)/h - KAPPA\_0)

diag\_el[-1] = (-k(1)/h - KAPPA\_1)

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

diag\_el[i] = -(2\*k(X[i]))/(h\*\*2) - D[i]

up\_diag = np.zeros(n)

up\_diag[0] = k(0)/h

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

up\_diag[i] = k(X[i])/(h\*\*2) + dk(X[i])/(2\*h)

low\_diag = np.zeros(n)

low\_diag[-1] = k(1) / h

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

low\_diag[i] = k(X[i])/(h\*\*2) - dk(X[i])/(2\*h)

neodnorodnost = np.zeros(n)

neodnorodnost[0] = -G\_0

neodnorodnost[-1] = -G\_1

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

neodnorodnost[i] = -PHI[i]

return X, sweep\_method(up\_diag, diag\_el, low\_diag, neodnorodnost)

**Интегро-интерполяционный метод**

Будем рассматривать наше уравнение как уравнение стационарного распределения тепла в стержне. Для него справедлив закон сохранения тепла (уравнение баланса) на отрезке :

*,*

Где – тепловой поток, – коэффициент теплопроводности, – температура.

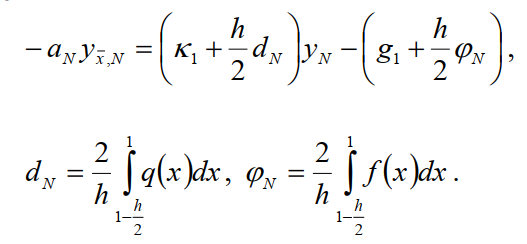
Аппроксимируем дифференциальное уравнение на сетке. Для отрезка

Введём коэффициенты:

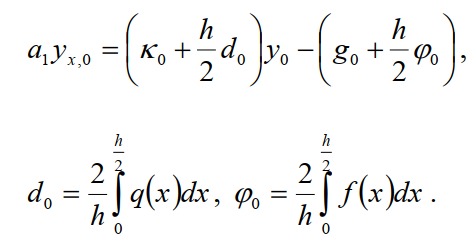
Получим разностную задачу:

Перейдём к аппроксимации граничных условий. Уравнение баланса для отрезка [1 - , 1]

Из граничных условий имеем , из полученного ранее . А также, заменяя в первом интеграле на интерполяционный многочлен на отрезке [1 - , 1] получаем следующую аппроксимацию граничного условия:



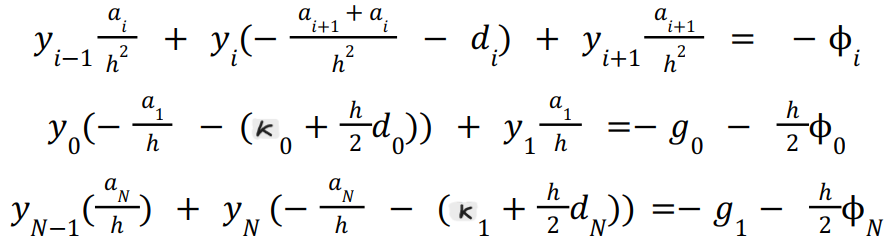
Аналогично получаем:



**Реализация**

Для вычисления интегралов по условию надо использовать формулу трапеций, при этом коэффициенты будут иметь вид:

В результате получаем СЛАУ:



**Код**

def solve\_balance(k, q, f, kappa\_0, g\_0, kappa\_1, g\_1, h=0.1):

X = np.arange(0, 1 + h/2, h)

n = X.shape[0]

A = np.array([2 \* k(x - h) \* k(x) / (k(x) + k(x + h)) for x in X])

PHI = np.array([1/2 \* (f(x - 0.5 \* h) + f(x + 0.5 \* h)) for x in X])

PHI[0] = 1/2 \* (f(h / 2) + f(0))

PHI[-1] = 1/2 \* (f(1) - f(1 - h / 2))

D = np.array([1/2 \* (q(x - 0.5 \* h) + q(x + 0.5 \* h)) for x in X])

D[0] = 1/2 \* (q(h / 2) + q(0))

D[-1] = 1/2 \* (q(1) - q(1 - h / 2))

diag\_el = np.zeros(n)

diag\_el[0] = -A[1]/h - (kappa\_0 + h/2 \* D[0])

diag\_el[-1] = -A[-1]/h - (kappa\_1 + h/2 \* D[-1])

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

diag\_el[i] = -(A[i] + A[i+1])/(h\*\*2) - D[i]

up\_diag = np.zeros(n)

up\_diag[0] = A[1]/h

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

up\_diag[i] = A[i+1]/(h\*\*2)

low\_diag = np.zeros(n)

low\_diag[-1] = A[-1]/h

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

low\_diag[i] = A[i]/(h\*\*2)

neodnorodnost = np.zeros(n)

neodnorodnost[0] = -g\_0 - h \* PHI[0]/2

neodnorodnost[-1] = -g\_1 - h\*PHI[-1]/2

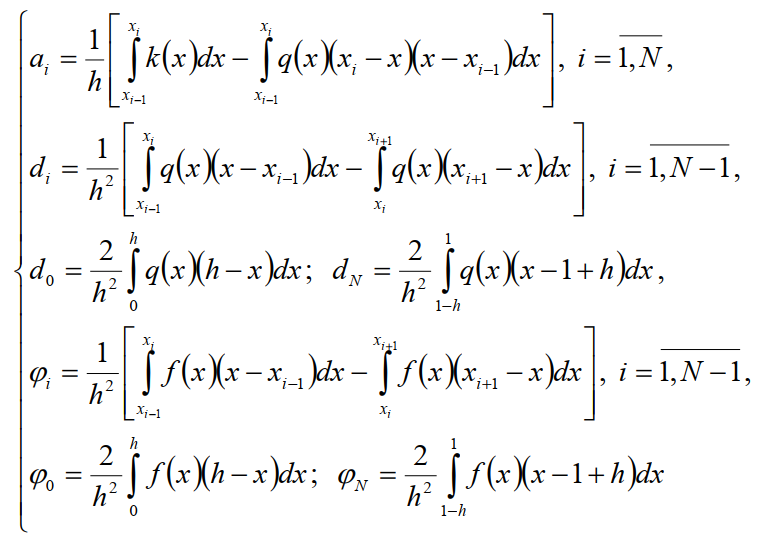
for i in range(1, n - 1):

neodnorodnost[i] = -PHI[i]

return X, sweep\_method(up\_diag, diag\_el, low\_diag, neodnorodnost)

**Вариационно-разностный метод**

Формулы этого метода будут такими же, изменятся только коэффициенты:



**Реализация**

По условию, для вычисления интегралов надо использовать формулу средних прямоугольников. Тогда коэффициенты примут вид:

**Код**

def solve\_ritz(k, q, f, kappa\_0, g\_0, kappa\_1, g\_1, h=0.1):

X = np.arange(0, 1 + h/2, h)

n = X.shape[0]

A = np.array([k(x-h/2) - q(x-h/2)\*h/4 for x in X])

PHI = np.array([(f(x-h/2) + f(x+h/2)) / 2 for x in X])

PHI[0] = f(h/2)

PHI[-1] = f(1-h/2)

D = np.array([(q(x-h/2) + q(x+h/2)) / 2 for x in X])

D[0] = q(h/2)

D[-1] = q(1-h/2)

diag\_el = np.zeros(n)

diag\_el[0] = -A[1]/h - (kappa\_0 + h/2 \* D[0])

diag\_el[-1] = -A[-1]/h - (kappa\_1 + h/2 \* D[-1])

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

diag\_el[i] = -(A[i] + A[i+1])/(h\*\*2) - D[i]

up\_diag = np.zeros(n)

up\_diag[0] = A[1]/h

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

up\_diag[i] = A[i+1]/(h\*\*2)

low\_diag = np.zeros(n)

low\_diag[-1] = A[-1]/h

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

low\_diag[i] = A[i]/(h\*\*2)

neodnorodnost = np.zeros(n)

neodnorodnost[0] = -g\_0 - h \* PHI[0]/2

neodnorodnost[-1] = -g\_1 - h\*PHI[-1]/2

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

neodnorodnost[i] = -PHI[i]

return X, sweep\_method(up\_diag, diag\_el, low\_diag, neodnorodnost)

**Сравнительный анализ полученных решений**

Полученные решения:

Аппроксимация разностной схемой: [0.99987896 0.99487593 0.97992289 0.9551693 0.92086265 0.87734593 0.82505425 0.76451047 0.69631998 0.62116468 0.53979614]

Интегро-интерполяционный метод: [0.98463114 0.97927492 0.96411195 0.93938933 0.90545312 0.86274725 0.81181191 0.75328182 0.6878849 0.61644199 0.53986884]

Вариационно-разностный метод: [1.00087406 0.99590836 0.98101577 0.95634034 0.9221203 0.87868548 0.82645356 0.76592538 0.6976789 0.62236216 0.54068456]

В качестве точного решения возьму решение, полученное аппроксимацией разностной схемой при шаге 10-6:

[1.0000007565603757 0.9950048993857485 0.9800672841488035 0.9553371666557199 0.9210616380184248 0.8775831603744514 0.8253361673381864 0.7648426908555206 0.69670716278941 0.6216103708682184 0.5403026557978144]

Нормы погрешностей:

Погрешность 1: 0.00029527583545757574

Погрешность 2: 0.013295678280938427

Погрешность 3: 0.001019491849045262

Все методы имеют второй порядок аппроксимации, однако во втором и третьих методах используются интегралы, которые вычисляются приближенно, что даёт погрешность, потому и точность у них меньше. При уменьшении шага погрешность уменьшается, потому в качестве точного решения и использовал первый метод с шагом 10-6.