БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ  
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

Лабораторная работа №1

«Способы построения и исследования разностных схем»  
по дисциплине «Численные методы математической физики»

Вариант 9

Выполнил: Прудникович В.В. студент 3 курс 7 группы

Проверил: Будник А.М. Доцент кафедры вычислительной математики ФПМИ

**Постановка задачи**

Дана третья краевая задача для ОДУ второго порядка следующего вида:

где

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 2 |  |  | 1 | 1 |  | 0 |

Необходимо:

1. Используя разностные операторы вместо дифференциальных, аппроксимировать поставленную задачу разностной схемой 2-го порядка на минимальном шаблоне. Для повышения порядка аппроксимации граничных условий выделить главный член погрешности и заменить его, используя следующий вид исходного дифференциального уравнения:
2. Интегро-интерполяционным методом построить наилучшую консервативную разностную схему, когда интегралы в коэффициентах этой схемы вычисляются точно.
3. Аппроксимировать исходную задачу вариационно-разностным методом. Для вычисления коэффициентов полученной схемы использовать формулу средних прямоугольников.
4. Методом разностной прогонки реализовать полученные в п.п. 1-3 разностные схемы при Провести сравнительный анализ сеточных решений при разных шагах.
5. **Аппроксимация исходной задачи с помощью РС**
   1. **Выбор шаблона и РС**

Для аппроксимации задачи, содержащей 2-ую производную минимальным будет шаблон, содержащий (2+1) точку.

Граничные условия содержат производные первого порядка, следовательно минимальным шаблоном для них будет являться 2-хточечный шаблон. Учтем, что в шаблоне следует использовать точки, принадлежащие области определения.

Тогда:

Распишем выражение для производной :

Тогда исходное уравнение примет вид:

Его можно аппроксимировать следующим образом:

аппроксимирует с порядком .

аппроксимирует также с порядком

Тогда порядок аппроксимации уравнения заданной схемой равен

* 1. **Повышение порядка аппроксимации**

Аппроксимируем правое граничное условие и рассмотрим для него .

Для нахождения порядка аппроксимации найдем значение выражение и используем представление функции в ряд Тейлора.

*, =>*

Для повышения порядка аппроксимации заменим главный член выражения (\*), заменив коэффициенты на соответственно.

Тогда, выражение (\*) примет вид:

Для получения порядка аппроксимации , нам необходимо занулить выражения, находящиеся в скобках. Исходя из этого наши коэффициенты равны:

= .

При рассмотрении погрешности аппроксимации левого граничного условия изменятся рассматриваемая точка (1 => 0), а также выражение для первой разностной производной.

Следовательно, для увеличения порядка аппроксимации будут использованы коэффициенты:

* 1. **Итоговая разностная схема и алгоритм метода**

Получаем следующую разностную схему для исходной задачи:

Так как разностная схема выражена неявно, то, для получения коэффициентов , необходимо решить СЛАУ для трёхдиагональной матрицы. Для гарантированной устойчивости и уменьшения времени работы алгоритма будем использовать метод Прогонки.

**1.4 Листинг программы**

**import numpy as np  
def f(x):  
 return 2 \* np.cos(x) - np.sin(x)  
def k(x):  
 return 2 - x  
def PROG(e, d, c, b):  
 n = len(d)  
 y = [0 for \_ in range (n)]  
 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]  
 y[n - 1] = b[n - 1] / d[n - 1]  
 for k in reversed(range(n - 1)):  
 y[k] = (b[k] - e[k] \* y[k + 1]) / d[k]  
 return y  
dk = -1  
ae = [1, np.math.tan(1)]  
g = [1, 0]  
h = 0.1  
X = np.arange(0, 1 + h / 2, h)  
n = len(X)  
Q = X  
F = f(X)  
k\_xi\_05 = np.array([k(x - 0.5 \* h) for x in X])  
ae\_0 = ae[0] \* (1 - (h / 2) \* (-1) / k(0))  
ae\_1 = ae[1] \* (1 + (h / 2) \* (-1) / k(1)) + h / 2  
g\_0 = g[0] \* (1 - (h / 2) \* (-1) / k(0)) + (h / 2) \* f(0)  
g\_1 = g[1] \* (1 + (h / 2) \* (-1) / k(1)) + (h / 2) \* f(1)  
mid = [0 for i in range(n)]  
mid[0] = (-k(0) / h - ae\_0)  
mid[-1] = (-k(1) / h - ae\_1)  
for i in range(1, n - 1):  
 mid[i] = -(k\_xi\_05[i] + k\_xi\_05[i + 1]) / (h \*\* 2) - Q[i]  
 up = [0 for i in range(n)]  
 up[0] = k(0) / h  
for i in range(1, n - 1):  
 up[i] = k\_xi\_05[i + 1] / (h \*\* 2)  
low = [0 for i in range(n)]  
low[-1] = k(1) / h  
for i in range(1, n - 1):  
 low[i] = k\_xi\_05[i] / (h \*\* 2)  
neodn = [0 for i in range(n)]  
neodn[0] = -g\_0  
neodn[-1] = -g\_1  
for i in range(1, n - 1):  
 neodn[i] = -F[i]  
print("Узлы: ", X)  
Y = PROG(up, mid, low, neodn)  
print("Значения в Y в узлах", Y)**

1. **Интегро-интерполяционный метод**
   1. **Физическая интерпретация. Подсчёт коэффициентов.**

­Рассматриваемое нами уравнение можно трактовать как уравнение стационарного распределения температуры в стержне

с коэффициентом теплопроводности

Закон сохранения тепла (уравнение баланса) на отрезке

имеет следующий вид:

где – тепловой поток.

Вычислим значения для коэффициентов:

Тогда, запишем получившиеся выражения для наших коэффициентов:

Используя полученные коэффициенты, получим разностную схему вида:

**Листинг программы**   
**def f(x):  
 return 2 \* np.cos(x) - np.sin(x)  
def k(x):  
 return 2 - x  
def q(x):  
 return x  
def PROG(e, d, c, b):  
 n = len(d)  
 y = [0 for \_ in range (n)]  
 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]  
 y[n - 1] = b[n - 1] / d[n - 1]  
 for k in reversed(range(n - 1)):  
 y[k] = (b[k] - e[k] \* y[k + 1]) / d[k]  
 return y  
dk = -1  
ae = [1, np.math.tan(1)]  
g = [1, 0]  
h = 0.1  
X = np.arange(0, 1 + h / 2, h)  
n = len(X)  
Q = X  
F = f(X)  
a = [0 for i in range(n)]  
d = [0 for i in range(n)]  
phi = [0 for i in range(n)]  
for i in range(1, n-1):  
 a[i] = h / (np.log(np.abs(X[i] - h -2)) - np.log(np.abs(X[i] -2)))  
 phi[i] = (2 \* np.sin(X[i]+h/2) + np.cos(X[i]+h/2) - 2 \* np.sin(X[i] - h/2) - np.cos(X[i] - h/2)) / h  
 d[i] = ((X[i]+h/2) \*\* 2 - (X[i] - h/2) \*\* 2) / (2 \* h)  
 d[0] = h / 4  
a[-1] = h / np.log((2 - X[-1] + h) / (2 - X[-1]))  
phi[0] = 2 \* (2 \* np.sin(h / 2) + np.cos(h / 2) - 1) / h  
d[-1] = (4 - h) / 4  
phi[-1] = 2 \* (2 \* np.sin(h / 4) \* (2 \* np.cos(1 - h / 4) - np.sin(1 - h / 4))) / h  
print(phi)  
X = np.arange(0, 1 + h / 2, h)  
n = len(X)  
mid = [0 for i in range(n)]  
mid[0] = -a[1] / h - (ae[0] + h / 2 \* d[0])  
mid[-1] = -a[-1] / h - (ae[1] + h / 2 \* d[-1])  
for i in range(1, n - 1):  
 mid[i] = -(a[i] + a[i + 1]) / (h \*\* 2) - d[i]  
up = [0 for i in range(n)]  
up[0] = a[1] / h  
for i in range(1, n - 1):  
 up[i] = a[i + 1] / (h \*\* 2)  
low = [0 for i in range(n)]  
low[-1] = a[-1] / h  
for i in range(1, n - 1):  
 low[i] = a[i] / (h \*\* 2)  
gran = [0 for i in range(n)]  
gran[0] = -g[0] - h \* phi[0] / 2  
gran[-1] = -g[1] - h \* phi[-1] / 2  
for i in range(1, n - 1):  
 gran[i] = -phi[i]  
Y = PROG(up, mid, low, gran)  
print("Значения Y в узлах", Y)**

1. **Вариационно-разностный метод**

**3.1. Выбор разностной схемы.**

Запишем реализацию разностной схемы для интегро-интерполяционного метода:

В вариационно-разностном методе данная схема также имеет место, однако изменятся коэффициенты , , , которые будут вычисляться по следующим формулам:

**3.2. Вычисление коэффицинетов.**

В нашем случае используемые коэффициенты следует считать приближенно, используя формулу средних прямоугольников.

тогда коэффициенты разностной схемы будут иметь следующий вид:

**Листинг программы**

**import numpy as np  
def f(x):  
 return 2 \* np.cos(x) - np.sin(x)  
def k(x):  
 return 2 - x  
def q(x):  
 return x  
def PROG(e, d, c, b):  
 n = len(d)  
 y = [0 for \_ in range (n)]  
 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]  
 y[n - 1] = b[n - 1] / d[n - 1]  
 for k in reversed(range(n - 1)):  
 y[k] = (b[k] - e[k] \* y[k + 1]) / d[k]  
 return y  
dk = -1  
ae = [1, np.math.tan(1)]  
g = [1, 0]  
h = 0.1  
X = np.arange(0, 1 + h / 2, h)  
n = len(X)  
Q = X  
F = f(X)  
  
X = np.arange(0, 1 + h / 2, h)  
n = len(X)  
k\_05 = np.array([k(xi - 0.5 \* h) - pow(h / 2, 2) \* q(xi - 0.5 \* h) for xi in X])  
phi = (f(X - 0.5 \* h) + f(X + 0.5 \* h)) / 2  
phi[0], phi[-1] = f(h / 2), f(1 - h / 2)  
d = (q(X - 0.5 \* h) + q(X + 0.5 \* h)) / 2  
d[0], d[-1] = q(h / 2), q(1 - h / 2)  
mid = np.zeros(n)  
mid[0] = -k\_05[1] / h - (ae[0] + (h / 2) \* d[0])  
mid[-1] = -k\_05[-1] / h - (ae[1] + (h / 2) \* d[-1])  
for i in range(1, n - 1):  
 mid[i] = -(k\_05[i] + k\_05[i + 1]) / (h \*\* 2) - d[i]  
up = np.zeros(n)  
up[0] = k\_05[1] / h  
for i in range(1, n - 1):  
 up[i] = k\_05[i + 1] / (h \*\* 2)  
low = np.zeros(n)  
low[-1] = k\_05[-1] / h  
for i in range(1, n - 1):  
 low[i] = k\_05[i] / (h \*\* 2)  
gran = np.zeros(n)  
gran[0] = -g[0] - h \* phi[0] / 2  
gran[-1] = -g[1] - h \* phi[-1] / 2  
for i in range(1, n - 1):  
 gran[i] = -phi[i]  
print("Узлы: ", X)  
Y = PROG(up, mid, low, gran)  
print("Значения в Y в узлах", Y)**

1. **Сравнительный анализ полученных решений**

**Представим результат работы методов ниже:**

Узлы: [0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1 ]

**Метод с повышением порядка точности**

Значения Y в узлах

[0.9998820544877156, 0.994876009780211, 0.9799199771807804, 0.9551637151216605, 0.9208549301677253, 0.8773367934514069, 0.8250445011791296, 0.7645009127702628, 0.6963113089015496, 0.6211573196470911, 0.5397900796138133]

**Интегро-интерполяционный метод**

Значения Y в узлах

[1.0002180685074464, 0.9952302953355717, 0.9802954852888047, 0.9555633947759224, 0.9212816945043567, 0.8777934803290712, 0.8255338237307511, 0.7650253919699956, 0.6968731738255624, 0.6217583492181318, 0.5404313365805238]

**Вариационно-разностный метод**

Значения Y в узлах

[0.9994246499908118, 0.9945293179331071, 0.9796998886419412, 0.9550841785093427, 0.9209277635076195, 0.8775715415530696, 0.8254483467987332, 0.7650786514595584, 0.6970654009933188, 0.6220880390401456, 0.5408957900918314]

Сразу можно заметить, что методы отличаются друг от друга не более, чем на. Однако стоит сравнить их с точным решением. В качестве точного решения возьмём результат интегро-интерполяционного метода для шага h = 10e-6:

0.0 - 0.9999998741874102

0.1 - 0.9950040338350802

0.2 - 0.980066447946185

0.3 - 0.955336366056194

0.4 - 0.9210608782622832

0.5 - 0.8775824551333569

0.6 - 0.8253355188844914

0.7 - 0.7648421009839007

0.8 - 0.6967066263547809

0.9 - 0.6216098900910093

1.0 - 0.5403022369200055

Рассмотрим разницу между полученными с помощью первого метода значениями и точным решением.

Погрешность:

[0.00011781969969459904, 0.00012802405486922463, 0.00014647076540463821, 0.00017265093453344704, 0.00020594809455787288, 0.0002456616819499491, 0.0002910177053617735, 0.00034118821363793295, 0.00039531745323129197, 0.000452570443918221]

Максимальная погрешность 0.000452570443918221

Тогда можно сделать вывод, что погрешность аппроксимации для первого метода не меньше, чем .

В данном случае метод показал меньшую погрешность, чем требуемая . Максимальная погрешность расположена на правой границе. Её накопление обусловлено использованием новых приближенных значений на каждом шаге.

Погрешность для вариационного метода:

[0.0005752241965983895, 0.00047471590197312175, 0.00036655930424378536, 0.0002521875468513013, 0.00013311475466371014, 1.0913580287263969e-05, 0.00011282791424183092, 0.00023655047565773746, 0.0003587746385379509, 0.00047814894913633665]

Максимальная погрешность 0.0005752241965983895. ()

Погрешность для интерполяционного метода:

Погрешность:

[0.00021819432003622463, 0.00022626150049154248, 0.00022903734261969788, 0.0002270287197284615, 0.00022081624207348494, 0.00021102519571436495, 0.00019830484625971145, 0.00018329098609493055, 0.00016654747078148624, 0.0001484591271224689]

Максимальная погрешность 0.00022903734261969788 (