БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №1

Вариант 4

**Построение разностных схем**

**Выполнила:**

Старостина Ангелина

3 курс 7 группа

**Преподаватель:**

Будник Анатолий Михайлович

Минск, 2023

**Содержание**

Постановка задачи…………………………….………………………………………....3

Аппроксимация на минимальном шаблоне...…….…………………………………....4

Интегро-интерполяционный метод.…………………………………………..………..9

Вариационно-разностный метод.………………………………………...…..………..13

**Постановка задачи**

Дана третья краевая задача для ОДУ второго порядка следующего вида:

Где

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| f(x) | k(x) | q(x) |  |  |  |  |
|  |  |  | 2 | 2 | 0 | -3sin1 |

**Аппроксимация на минимальном шаблоне**

**Алгоритм решения**

Строим аппроксимацию на сетке .

Аппроксимация уравнения:

Вычислим :

Аппроксимация первого начального условия:

где

Аппроксимация второго начального условия:

Коэффициенты получим из погрешности:

Следовательно:

Схема в индексной форме:

Приведем уравнение к виду удобному для определения коэффициентов метода разностной прогонки:

Следовательно

Решение системы найдем методом прогонки. Алгоритм:

где

Из граничных условий получаем коэффициенты:

Аналогично из первого граничного условия получаем:

Метод прогонки проверяем на устойчивость по формулам:

**Листинг программы**

import math  
import numpy as np  
  
h = 0.1  
N = int(1/h)  
k0 = 2  
g0 = 2  
k1 = 0  
g1 = -3 \* math.sin(1)  
y = np.zeros(N+1)  
alpha = np.zeros(N)  
beta = np.zeros(N)  
  
def x(num):  
 return num \* h  
  
def f(i):  
 return 4 \* math.cos(x(i)) - 2 \* x(i) \* math.sin(x(i))  
  
def k(x):  
 return 4 - x \*\* 2  
  
def q(x):  
 return x\*\*2  
  
def dk(x):  
 return -2\*x  
  
def calculate\_k0():  
 return k0 \* (1 - (h / 2) \* (dk(0) / k(0))) + h / 2 \* q(0)  
  
def calculate\_g0():  
 return g0 \* (1 - (h / 2) \* (dk(0) / k(0))) + h / 2 \* f(0)  
  
def calculate\_k1():  
 return k1 \* (1 + (h / 2) \* (dk(1) / k(1))) + h / 2 \* q(1) *#?*def calculate\_g1():  
 return g1 \* (1 + (h / 2) \* (dk(1) / k(1))) + h / 2 \* f(N) *#?*def B(num):  
 xi = x(num)  
 return k(xi) / (h \*\* 2) + dk(xi) / (2 \* h)  
  
  
def A(num):  
 xi = x(num)  
 return k(xi) / (h \*\* 2) - dk(xi) / (2 \* h)  
  
  
def C(num):  
 xi = x(num)  
 return 2 \* k(xi) / h\*\*2 + q(xi)  
  
  
def find\_alpha():  
 alpha[0] = k(0) / (k0 \* h + k(0))  
 for i in range(1, N):  
 alpha[i] = B(i) / (C(i) - A(i) \* alpha[i - 1])  
  
  
def find\_beta():  
 beta[0] = g0 \* h / (k0 \* h + k(0))  
 for i in range(1, N):  
 beta[i] = (f(i) + beta[i-1] \* A(i)) / (C(i) - A(i) \* alpha[i-1])  
  
  
def sweep\_method():  
 find\_alpha()  
 find\_beta()  
 k2\_sweep = k(1) / (k1 \* h + k(1))  
 nu2 = g1 \* h / (k1 \* h + k(1))  
 y[N] = (nu2 + k2\_sweep \* beta[N-1]) / (1 - alpha[N-1] \* k2\_sweep)  
 for i in reversed(range(0, N)):  
 y[i] = alpha[i] \* y[i+1] + beta[i]  
  
  
def print\_1000():  
 for i in range(int(N/100)):  
 print(y[i\*100])  
  
def test\_stab():  
 stab = []  
 B\_ = np.zeros(N)  
 for i in range(1, N):  
 B\_[i] = B(i - 1)  
 A\_ = np.zeros(N)  
 for i in range(1, N):  
 A\_[i] = A(i - 1)  
 C\_ = np.zeros(N)  
 for i in range(1, N):  
 C\_[i] = C(i - 1)  
 for i in range(N):  
 stab.append(abs(C\_[i]) >= abs(A\_[i]) + abs(B\_[i]))  
 stab.append(alpha[0] <= 1)  
 stab.append(k(1) / (h \* (k(1) / h + k1)) <= 1)  
 stab.append(abs(k(1) / (h \* (k(1) / h + k1)) + abs(alpha[0]) < 2))  
 for i in range(len(stab)):  
 if stab[i] == False:  
 print("Достаточное условие устойчивости метода прогонки не выполнено")  
 return -1  
 print("Достаточное условие устойчивости метода прогонки выполнено")  
 return 0  
  
  
test\_stab()  
g1 = calculate\_g1()  
g0 = calculate\_g0()  
k1 = calculate\_k1()  
k0 = calculate\_k0()  
sweep\_method()  
print(y)

**Вывод программы**

Достаточное условие устойчивости метода прогонки выполнено

[0.99897722 0.99392608 0.97892466 0.9541221 0.91976547 0.87619727

0.82385194 0.7632515 0.69500025 0.61977865 0.53833634]

**Интегро-интерполяционный метод**

**Алгоритм решения**

Разностная схема имеет вид:

где коэффициенты вычисляются по формулам (интегралы вычисляем по формуле средних прямоугольников):

Приведем уравнение к виду удобному для определения коэффициентов метода разностной прогонки:

Следовательно

Решение системы найдем методом прогонки.

Из граничных условий получаем коэффициенты:

**Листинг программы**

import math  
import numpy as np  
  
h = 0.1  
N = int(1/h)  
k0 = 2  
g0 = 2  
k1 = 0  
g1 = -3 \* math.sin(1)  
y = np.zeros(N+1)  
alpha = np.zeros(N)  
beta = np.zeros(N)  
a = np.zeros(N+1)  
  
def x(num):  
 return num \* h  
  
def f(i):  
 if i == N:  
 return 4 \* math.cos(1 - h / 4) - (2 - h / 2) \* math.sin(1 - h / 4)  
 elif i == 0:  
 return 4 \* math.cos(h / 4) - (h / 2) \* math.sin(h / 4)  
 else:  
 return 4 \* math.cos(x(i)) - 2 \* x(i) \* math.sin(x(i))  
  
def k(x):  
 return 4 - x \*\* 2  
  
def q(x):  
 return x\*\*2  
  
def calculate\_a():  
 for i in range(N+1):  
 a[i] = 4 - ((x(i) + x(i-1)) / 2) \*\* 2  
  
def d(i):  
 if i == 0:  
 return h\*\*2 / 16  
 if i == N:  
 return (1 - h / 4)\*\*2  
 else:  
 return x(i)\*\*2  
  
def B(i):  
 return a[i+1] / h\*\*2  
  
  
def A(i):  
 return a[i] / h\*\*2  
  
  
def C(i):  
 return (a[i+1] + a[i]) / h\*\*2 + d(i)  
  
  
def find\_alpha():  
 alpha[0] = a[0] / (h \* (k0 + h/2 \* d(0) + a[0]/h))  
 for i in range(1, N):  
 alpha[i] = B(i) / (C(i) - A(i) \* alpha[i - 1])  
  
  
def find\_beta():  
 beta[0] = (g0 + h/2\*f(0)) / (k0 + h/2 \* d(0) + a[0]/h)  
 for i in range(1, N):  
 beta[i] = (f(i) + beta[i-1] \* A(i)) / (C(i) - A(i) \* alpha[i-1])  
  
  
def sweep\_method():  
 find\_alpha()  
 find\_beta()  
 k2\_sweep = a[N] / (h \* (k1 + h / 2 \* d(N) + a[N] / h))  
 nu2 = (g1 + h / 2 \* f(N)) / (k1 + h / 2 \* d(N) + a[N] / h)  
 y[N] = (nu2 + k2\_sweep \* beta[N-1]) / (1 - alpha[N-1] \* k2\_sweep)  
 for i in reversed(range(0, N)):  
 y[i] = alpha[i] \* y[i+1] + beta[i]  
  
  
def test\_stab():  
 stab = []  
 B\_ = np.zeros(N)  
 for i in range(1, N):  
 B\_[i] = B(i - 1)  
 A\_ = np.zeros(N)  
 for i in range(1, N):  
 A\_[i] = A(i - 1)  
 C\_ = np.zeros(N)  
 for i in range(1, N):  
 C\_[i] = C(i - 1)  
 for i in range(N):  
 stab.append(abs(C\_[i]) >= abs(A\_[i]) + abs(B\_[i]))  
 stab.append(alpha[0] <= 1)  
 stab.append(a[N] / (h \* (k1 + h / 2 \* d(N) + a[N] / h)) <= 1)  
 stab.append(abs(k(1) / (h \* (k(1) / h + k1)) + abs(alpha[0]) < 2))  
 for i in range(len(stab)):  
 if stab[i] == False:  
 print("Достаточное условие устойчивости метода прогонки не выполнено")  
 return -1  
 print("Достаточное условие устойчивости метода прогонки выполнено")  
 return 0  
  
  
test\_stab()  
calculate\_a()  
sweep\_method()  
print(y)

**Вывод программы**

Достаточное условие устойчивости метода прогонки выполнено

[1.0014641 0.99653814 0.98165633 0.95696865 0.92272319 0.87926367

0.8270261 0.76653447 0.69839557 0.62329312 0.54198105]

**Вариационно-разностный метод**

**Алгоритм решения**

Разностная схема:

Где коэффициенты вычисляются по формулам (интегралы вычисляем по формуле трапеций):

Решение системы найдем методом прогонки. Коэффициенты будут аналогичны коэффициентам интегро-интерполяционного метода.

**Листинг программы**

import math  
import numpy as np  
  
h = 0.1  
N = int(1/h)  
k0 = 2  
g0 = 2  
k1 = 0  
g1 = -3 \* math.sin(1)  
y = np.zeros(N+1)  
alpha = np.zeros(N)  
beta = np.zeros(N)  
a = np.zeros(N+1)  
  
def x(num):  
 return num \* h  
  
def f(i):  
 if i == N:  
 return (4 \* math.cos(1) - 2 \* math.sin(1))  
 elif i == 0:  
 return 4  
 else:  
 return 4 \* math.cos(x(i)) - 2 \* x(i) \* math.sin(x(i))  
  
def k(x):  
 return 4 - x \*\* 2  
  
def q(x):  
 return x\*\*2  
  
def calculate\_a():  
 for i in range(N+1):  
 a[i] = (8 - x(i-1)\*\*2 - x(i)\*\*2) / 2  
  
def d(i):  
 if i == 0:  
 return 0  
 if i == N:  
 return 1  
 else:  
 return x(i) \*\* 2  
  
def B(i):  
 return a[i+1] / h\*\*2  
  
def A(i):  
 return a[i] / h\*\*2  
  
def C(i):  
 return a[i+1] / h\*\*2 + a[i] / h\*\*2 + d(i)  
  
def find\_alpha():  
 alpha[0] = a[0] / (h \* (k0 + h/2 \* d(0) + a[0]/h))  
 for i in range(1, N):  
 alpha[i] = B(i) / (C(i) - A(i) \* alpha[i - 1])  
  
  
def find\_beta():  
 beta[0] = (g0 + h/2\*f(0)) / (k0 + h/2 \* d(0) + a[0]/h)  
 for i in range(1, N):  
 beta[i] = (f(i) + beta[i-1] \* A(i)) / (C(i) - A(i) \* alpha[i-1])  
  
def sweep\_method():  
 find\_alpha()  
 find\_beta()  
 k2\_sweep = a[N] / (h \* (k1 + h / 2 \* d(N) + a[N] / h))  
 nu2 = (g1 + h / 2 \* f(N)) / (k1 + h / 2 \* d(N) + a[N] / h)  
 y[N] = (nu2 + k2\_sweep \* beta[N-1]) / (1 - alpha[N-1] \* k2\_sweep)  
 for i in reversed(range(0, N)):  
 y[i] = alpha[i] \* y[i+1] + beta[i]  
  
def print\_1000():  
 for i in range(int(N/100)):  
 print(y[i\*100])  
  
def test\_stab():  
 stab = []  
 B\_ = np.zeros(N)  
 for i in range(1, N):  
 B\_[i] = B(i - 1)  
 A\_ = np.zeros(N)  
 for i in range(1, N):  
 A\_[i] = A(i - 1)  
 C\_ = np.zeros(N)  
 for i in range(1, N):  
 C\_[i] = C(i - 1)  
 for i in range(N):  
 stab.append(abs(C\_[i]) >= abs(A\_[i]) + abs(B\_[i]))  
 stab.append(alpha[0] <= 1)  
 stab.append(a[N] / (h \* (k1 + h / 2 \* d(N) + a[N] / h)) <= 1)  
 stab.append(abs(k(1) / (h \* (k(1) / h + k1)) + abs(alpha[0]) < 2))  
 for i in range(len(stab)):  
 if stab[i] == False:  
 print("Достаточное условие устойчивости метода прогонки не выполнено")  
 return -1  
 print("Достаточное условие устойчивости метода прогонки выполнено")  
 return 0  
  
test\_stab()  
calculate\_a()  
sweep\_method()  
print(y)

**Вывод программы**

Достаточное условие устойчивости метода прогонки выполнено

[0.99793756 0.99282805 0.97775542 0.95286836 0.9184133 0.87473189

0.82225741 0.76151034 0.69309286 0.61768254 0.536025]

**Вывод**

В качестве точного решения возьмем решение интегро-интерполяционного метода с . Получаем: [1.00000016 0.99500433 0.98006675 0.95533666 0.92106117 0.87758274 0.82533579 0.76484237 0.69670689 0.62161015 0.54030248]

Погрешность методов:

1. [0.00102294 0.00107825 0.00114209 0.00121456 0.0012957 0.00138547 0.00148385 0.00159087 0.00170664 0.0018315 0.00196614]
2. [-0.00146395 -0.00153381 -0.00158958 -0.00163199 -0.00166202 -0.00168093 -0.00169031 -0.0016921 -0.00168868 -0.00168297 -0.00167856]
3. [0.0020626 0.00217628 0.00231133 0.0024683 0.00264787 0.00285085 0.00307838 0.00333203 0.00361403 0.00392761 0.00427748]

Все методы имеют второй порядок точности о чем свидетельствует погрешность решений.