**Лабораторная работа №3**

**«Метод сеток решения граничной задачи для ОДУ»**

Вариант 10

Выполнил студент 3 курса 2 группы ФПМИ

Сараев Владислав Максимович

Минск, 2020

**Постановка задачи**

Дана линейная граничная задача. Необходимо построить для граничной задачи разностную схему второго порядка аппроксимации на минимальном шаблоне и с помощью метода прогонки с шагом найти ее численное решение. Проверить выполняются ли достаточные условия корректности и устойчивости прогонки. Сравнить найденное численное решение с точным решением . В одной системе координат построить график функции и график полученного численного решения.

Граничная задача:

**Краткие теоретические сведения**

Имеем разностную задачу

Выбрав равномерную сетку узлов , разностное уравнение запишем в виде

Т.к. невязка разностного уравнения на решении дифференциального уравнения равна , то разностное уравнение аппроксимирует дифференциальное со вторым порядком.

Рассмотрим второе граничное условие

Разностное условие будем искать в виде

Сеточные коэффициенты введены, т.к. разностное граничное условие аппроксимирует дифференциальное с первым порядком, а нам необходимо получить второй порядок аппроксимации.

Тогда

Отсюда следует, что, выбрав

=

получим разностное граничное условие

Получили схему второго порядка аппроксимации

Система в индексной форме имеет вид

Метод прогонки:

Имеем трехдиагональную систему

*c*0*y*0 *–* *b*0*y*1                                                 = *f*0,

*–a*1*y*0 + *c*1*y*1 *– b*1*y*2                                      = *f*1,

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

*–aiyi–*1+ *ciyi* *– biyi+*1              = *fi*,

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

*–aNyN–*1 + *cNyN* = *fN*.

Верхняя двухдиагональная матрица

*y*0 – α1 *y*1                                              = β1,

*y*1 *–* α2 *y*2                                   = β2,

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

*yi* – α*i+*1*yi+*1              = β*i+*1,

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

*yN* = β*N+*1.

Коэффициенты для прямой прогонки вычисляются по формулам:

α1*=*,β1*=*,

α*i+*1*=*,β*i+*1*=*, *i=* 1, 2, ... , *N*–1,

β*N+*1*=*;

Для обратной *yN*=β*N+*1, *yi=*α*i+*1*yi+*1*+*β*i+*1, *i=* *N*–1, ...,1, 0

Если элементы *aij*, 1 ≤ *i , j* ≤ *n*, некоторой матрицы *A* удовлетворяют условиям

**, 1≤*i*≤*n*

то говорят, что матрица *A* обладает свойством диагонального преобладания по строкам.

Диагональное преобладание является достаточным условием корректности прогонки.

**Листинг**

# coding=utf-8

**import** numpy **as** np

**from** matplotlib **import** pyplot **as** plot

**import** time

**from** math **import** fabs

# Искомая функция

**def** u**(**x**):**

**return** 1 **/** **(**x **+** 1**)**

# Функция при u'

**def** p**(**x**):**

**return** **-**3 **\*** **(**x **+** 1**)** **\*\*** 2

# Функция при u

**def** q**(**x**):**

**return** 2 **/** **(**x **+** 1**)** **\*\*** 2

# Правая часть

**def** f**(**x**):**

**return** **-**3

**def** check**(**matrix**):**

**for** i **in** **range(**1**,** **len(**matrix**[**1**])** **-** 1**):**

**if** fabs**(**matrix**[**0**][**i **-** 1**])** **+** fabs**(**matrix**[**2**][**i**])** **-** fabs**(**matrix**[**1**][**i**])** **>** 0**:**

**raise** **ValueError(**"Incorrect matrix"**)**

**def** tridiagonal\_matrix\_algorithm**(**matrix**,** f**):**

check**(**matrix**)**

n **=** **len(**f**)**

matrix**[**2**][**0**]** **/=** matrix**[**1**][**0**]**

f**[**0**]** **/=** matrix**[**1**][**0**]**

matrix**[**1**][**0**]** **=** 1

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

coeff **=** matrix**[**1**][**i**]** **-** matrix**[**2**][**i **-** 1**]** **\*** matrix**[**0**][**i **-** 1**]**

matrix**[**2**][**i**]** **/=** coeff

f**[**i**]** **=** **(**f**[**i**]** **-** f**[**i **-** 1**]** **\*** matrix**[**0**][**i **-** 1**])** **/** coeff

matrix**[**1**][**i**]** **=** 1

matrix**[**0**][**i **-** 1**]** **=** 0

f**[**n **-** 1**]** **=** **(**f**[**n **-** 1**]** **-** f**[**n **-** 2**]** **\*** matrix**[**0**][**n **-** 2**])** **/** **(**matrix**[**1**][**n **-** 1**]** **-** matrix**[**2**][**n **-** 2**]** **\*** matrix**[**0**][**n **-** 2**])**

matrix**[**1**][**n **-** 1**]** **=** 1

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

f**[**i**]** **-=** f**[**i **+** 1**]** **\*** matrix**[**2**][**i**]**

matrix**[**2**][**i**]** **=** 0

# y(x0)

y0 **=** 1

# шаг

h **=** 0.01

# сигма 1

sigma1 **=** **-**0.5

# ню 1

nu1 **=** **-**0.5

# границы

L **=** 0

R **=** 1

# узлы

dots **=** np**.**linspace**(**L**,** R**,** **int((**R **-** L**)** **/** h**)** **+** 1**)**

# значения функций в узлах

q\_values **=** **[**q**(**x**)** **for** x **in** dots**]**

p\_values **=** **[**p**(**x**)** **for** x **in** dots**]**

f\_values **=** **[**f**(**x**)** **for** x **in** dots**]**

# создание матрицы из трех диагоналях

matrix **=** np**.**array**([**np**.**zeros**(len(**dots**)** **-** 2**),** np**.**zeros**(len(**dots**)** **-** 1**),** np**.**zeros**(len(**dots**)** **-** 2**)])**

# создание правой части

b **=** np**.**zeros**(len(**dots**)** **-** 1**)**

# заполнение матрицы и правой части

matrix**[**1**][**0**]** **=** **-**2 **/** **(**h **\*\*** 2**)** **-** q\_values**[**1**]**

matrix**[**2**][**0**]** **=** 1 **/** **(**h **\*\*** 2**)** **+** p\_values**[**1**]** **/** **(**2 **\*** h**)**

b**[**0**]** **=** **-**f\_values**[**1**]** **-** 1 **/** **(**h **\*\*** 2**)** **+** p\_values**[**1**]** **/** **(**2 **\*** h**)**

**for** i **in** **range(**1**,** **len(**b**)** **-** 1**):**

matrix**[**0**][**i **-** 1**]** **=** 1 **/** **(**h **\*\*** 2**)** **-** p\_values**[**i **+** 1**]** **/** **(**2 **\*** h**)**

matrix**[**1**][**i**]** **=** **-**2 **/** **(**h **\*\*** 2**)** **-** q\_values**[**i **+** 1**]**

matrix**[**2**][**i**]** **=** 1 **/** **(**h **\*\*** 2**)** **+** p\_values**[**i **+** 1**]** **/** **(**2 **\*** h**)**

b**[**i**]** **=** **-**f\_values**[**i **+** 1**]**

N **=** **len(**b**)** **-** 1

matrix**[**0**][**N **-** 1**]** **=** **-**1 **/** h

matrix**[**1**][**N**]** **=** 1 **/** h **+** sigma1 **+** h **/** 2 **\*** p\_values**[**N **+** 1**]** **\*** sigma1 **+** h **\*** q\_values**[**N **+** 1**]** **/** 2

b**[**N**]** **=** nu1 **+** h **\*** f\_values**[**N **+** 1**]** **/** 2 **+** h **\*** p\_values**[**N **+** 1**]** **\*** nu1 **/** 2

**try:**

# метод прогонки

tridiagonal\_matrix\_algorithm**(**matrix**,** b**)**

# полученное решение

y **=** **[**y0**]**

**for** i **in** b**:**

y**.**append**(**i**)**

u\_values **=** **[**u**(**x**)** **for** x **in** dots**]**

# вывод результата

plot**.**plot**(**dots**,** y**,** label**=**"Approximation"**)**

plot**.**plot**(**dots**,** u\_values**,** label**=**"Function"**)**

plot**.**legend**()**

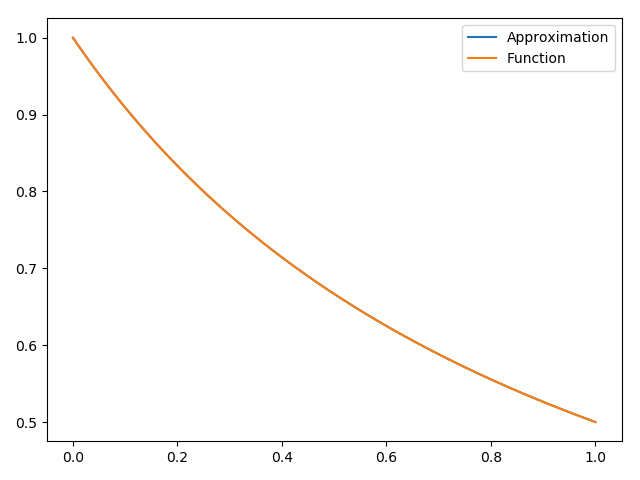
plot**.**show**()**

**print(**"Max delta: " **+** **str(max([abs(**y**[**i**]** **-** u\_values**[**i**])** **for** i **in** **range(len(**y**))])))**

**except** **ValueError** **as** e**:**

**print(**e**.**message**)**

**Результаты**

****

= 2.3369184327104442e-05

Условия для корректности метода прогонки соблюдаются.

**Выводы**

При решении граничной задачи, с помощью разностных схем можно получить СЛАУ с трехдиагональной матрицей, которая быстро решается с помощью метода прогонки. Если использовать разностные схемы второго порядка, то полученное решения является достаточно точным.