БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
Факультет прикладной математики и информатики

**Лабораторная работа №2**  
по теме  
«Метод прогонки численного решения систем линейных алгебраических уравнений трехдиагональной матрицы»

Выполнила   
Молочко Екатерина  
2 курс 7 группа

Преподаватель  
Горбачева Юлия Николаевна

Минск   
2021

Постановка задачи

1. Найти решение системы линейных алгебраических уравнений с трёх диагональной матрицей порядка
2. Оценить точность решения системы линейных алгебраических уравнений

Теоретические сведения

# Достаточное условие сходимости:

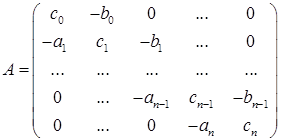
Пусть матрица исходной трехдиагональной системы удовлетворяют условию:

И в неравенствах А и Б хотя бы одно неравенство строгое, то метод прогонки применим и устойчив, т.е

устойчивость

# Алгоритм решения трех диагональной системы линейный алгебраических уравнений методом прогонки

Имеется трёхдиагональная матрица вида



И пусть требуется найти решение следующей системы:

Изображение выглядит как текст

Автоматически созданное описание

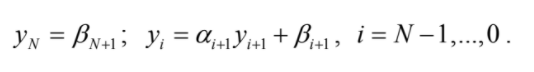
Суть алгоритма:

1. Вычисляем прогоночные коэффициенты по формулам

Изображение выглядит как текст

Автоматически созданное описание

2. Вычисляем решение по формулам



Так как значения yi находятся последовательно от (i +1)-го к i -му, то данные формулы называют формулами правой прогонки.

Аналогично выводятся формулы для левой прогонки.

Пусть коэффициенты системы удовлетворяют условиям

Изображение выглядит как текст

Автоматически созданное описание

причем хотя бы в одном из условий (3.14) – (3.15) выполняется строгое неравенство (другими словами, матрица A имеет диагональное преобладание по строкам, причем хотя бы в одном случае строгое).

Тогда для алгоритма (3.6) – (3.8) метода прогонки имеют место неравенства

Изображение выглядит как текст

Автоматически созданное описание

гарантирующие корректность и устойчивость метода.

Листинг программы

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

**def** solve\_tridiagonal\_matrix**(**\_A**,** \_B**,** \_C**,** \_F**):**

degree **=** \_F**.**shape**[**0**]**

alpha **=** np**.**zeros**(**degree**)**

beta **=** np**.**zeros**(**degree**)**

alpha**[**0**]** **=** \_B**[**0**]** **/** \_C**[**0**]**

beta**[**0**]** **=** \_F**[**0**]** **/** \_C**[**0**]**

**for** i **in** **range(**1**,** degree**):**

tmp **=** \_C**[**i**]** **-** \_A**[**i**]** **\*** alpha**[**i **-** 1**]**

alpha**[**i**]** **=** \_B**[**i**]/**tmp

beta**[**i**]** **=** **(**\_F**[**i**]** **+** \_A**[**i**]** **\*** beta**[**i **-** 1**])/**tmp

beta**[-**1**]** **=** **(**\_F**[-**1**]** **+** \_A**[-**1**]** **\*** beta**[-**2**])** **/** **(**\_C**[-**1**]** **-** \_A**[-**1**]** **\*** alpha**[-**2**])**

# back substitution

solution **=** np**.**zeros**(**degree**)**

solution**[-**1**]** **=** beta**[-**1**]**

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

solution**[**i**]** **=** alpha**[**i**]** **\*** solution**[**i **+** 1**]** **+** beta**[**i**]**

**return** solution

k **=** **int(input(**"Input degree of a matrix: "**))**

A **=** np**.**random**.**randint**(-**100**,** 100**,** k**)**

A**[**0**]** **=** 0

A **\*=** **-**1

**print(**"Low diagonal \n"**,** A**)**

B **=** np**.**random**.**randint**(-**100**,** 100**,** k**)**

B **\*=** **-**1

B**[**k **-** 1**]** **=** 0

**print(**"High diagonal \n"**,** B**)**

C **=** np**.**zeros**(**k**)**

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

C**[**i**]** **=** np**.**random**.**randint**(abs(**A**[**i**])** **+** **abs(**B**[**i**])** **+** 7**,** **abs(**A**[**i**])** **+** **abs(**B**[**i**])** **+** 14**)**

C**[**0**]** **=** np**.**random**.**randint**(abs(**B**[**0**])** **+** 7**,** **abs(**B**[**0**])** **+** 14**)**

C**[-** 1**]** **=** np**.**random**.**randint**(abs(**A**[-** 1**])** **+** 7**,** **abs(**A**[-** 1**])** **+** 14**)**

**print(**"Main diagonal \n"**,** C**)**

Y **=** np**.**zeros**(**k**)** ##создание вектора неизвестных

**for** i **in** **range(len(**Y**)):**

Y**[**i**]** **+=** i **+** 1

**print(**'\nVector of real unknowns: \n'**,** Y**)**

F **=** np**.**zeros**(**k**)** ##генерация столбца свободных членов

F**[-**1**]** **=** **(-**1**)** **\*** A**[-**1**]** **\*** Y**[-**2**]** **+** C**[-**1**]** **\*** Y**[-**1**]**

F**[**0**]** **=** C**[**0**]** **\*** Y**[**0**]** **-** B**[**0**]** **\*** Y**[**1**]**

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

F**[**i**]** **=** **(-**1**)** **\*** A**[**i**]** **\*** Y**[**i**-**1**]** **+** C**[**i**]\***Y**[**i**]** **-** B**[**i**]\***Y**[**i**+**1**]**

**print(**'\nVector of constant terms: \n'**,** F**)**

Y\_ **=** solve\_tridiagonal\_matrix**(**A**,** B**,** C**,** F**)**

**print(**"\nVector of computed unknowns: "**)**

format\_string **=** "{:.16f}"

**for** i **in** Y\_**:**

**print(**format\_string**.format(**i**))**

# вычисление максимум-норм векторов

v\_norm\_sub **=** np**.**array**(**np**.abs(**np**.**subtract**(**Y**,** Y\_**)))**

v\_norm\_unk **=** np**.abs(**Y**)**

# вычиление погрешности

uncertainty **=** v\_norm\_sub**[**np**.**argmax**(**v\_norm\_sub**)]** **/** v\_norm\_unk**[**np**.**argmax**(**v\_norm\_unk**)]**

**print(**"\nUncertainty is"**,** format\_string**.format(**uncertainty**))**

Результаты:

A = (88, -75, 73, -91, 4, -74, -92, 91, -95)  
B = (-89, -12, 23, -5, 24, 72, -61, -6, 36 )  
C = (99.0, 113.0, 109.0, 90.0, 122.0, 84.0, 145.0, 107.0, 134.0, 104.0)  
F = (277.0, 174.0, 385.0, 166.0, 830.0, -20.0, 1947.0, 1554.0, 118.0, 1895.0)

Y – вектор неизвестных  
Y = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
Y\_ – вектор-результат, вычисленный с помощью компьютера

Y\_ = (1.0000000000000002, 1.9999999999999998, 3.0000000000000000, 4.0000000000000000, 5.0000000000000000, 5.9999999999999982, 6.9999999999999991, 8.0000000000000018, 9.0000000000000018, 9.9999999999999982)  
  
Относительная погрешность, вычисленная по формуле = 0.0000000000000003

Вывод:

Метод прогонки препятствует накоплению вычислительной погрешности. Об этом свидетельствует малое значение относительной погрешности.