Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Лабораторная работа №1 по дисциплине “Вычислительная математика”

Вариант 7

Выполнил:
Думцев Виктор Сергеевич

Преподаватель:
Машина Екатерина Алексеевна

Санкт-Петербург 2025

Оглавление

Цель работы

Создать программу способную решать системы линейных уравнений, которые можно привести к диагональному преобладанию.

Описание метода

Метод простых итераций заключается в расчёте коэффицентов C_ij и D_i матрицы с диагональным преобладанием. Исходная система преобразуется путём выражения x_i через остальные члены i-й строки и b_i(i-й элемент вектора свободных членов), делении всей строки на коэффициенты a_ii, в результате чего в каждой строке все члены, кроме выраженных обладают коэффицентами < 1(благодаря диагональному преобладанию). Затем выбирается исходное приближение вектора неизвестных(оно может быть любым, но от близости исходного приближения к настоящему вектору неизвестных зависит скорость сходимости метода). На каждой следующей итерации результат становится всё ближе к истинному, благодаря тому,что значения, сильно большие нежели истинные, становятся меньше, а сильно меньшие - становятся больше. Значения близкие к истинным изменяются всё слабее и слабее в силу того,что начинают удовлетворять равенствам со всё меньшей погрешностью, в итоге "притягиваясь" к решению системы, где изменения с каждой итерацией всё меньше. Критерием остановки является выполнение условия,что разница между текущими и предыдущими значениями меньше выбранной точности.

Расчётные формулы

Рабочая формула метода простой итерации:

$$x_i^{(k+1)} = \frac{b_i}{a_{ii}} - \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{a_{ij}}{a_{ii}} x_j^k , \qquad i = 1, 2, \dots, n$$

где k – номер итерации.

$$\|C\|_1 = \max_{1 \le i \le n} \sum_{j=1}^{n} |c_{ij}| < 1$$

$$\max_{1 \le i \le n} \left| x_i^{(k)} - x_i^{(k-1)} \right| \le \varepsilon$$

Листинг программы

```
import numpy as np
import traceback
import pandas as pd


class LinearEquationSystem:


    def __init__(self, matrix, vector, precision):
        self.matrix = matrix
        self.vector = vector
        self.precision = precision
        self.__solution = None
        self.initial_value = vector
        self.__precision_vector = []
        self.__iter_counter = 0
        self.__norm = 0
```

```
        if not self.ensure_diagonal():
            print("Matrix can't be transformed to neccessary condition.")
            raise SystemExit
        print("Matrix is: ", self.matrix)
        self.solve()
        self.calculate_norm()

    @property
    def norm(self):
        return self.__norm

    @property
    def solution(self):
        return self.__solution

    @property
    def iter_counter(self):
        return self.__iter_counter

    @property
    def precision_vector(self):
        return self.__precision_vector

    def is_diag(self):
        # Checking, that for each row its diagonal element is greater than non-diagonal
ones
```

```
        if all([abs(self.matrix.iloc[i, i]) >= sum([abs(elem) for idx, elem in
enumerate(self.matrix.iloc[i]) if idx != i]) for i in range(len(self.vector))]):
            return True
        return False


    def ensure_diagonal(self):
        if self.is_diag():
            return True
        new_vector = np.zeros(len(self.vector))
        new_matrix = [[] for _ in range(len(self.vector))]
        for i in range(len(self.vector)):
            row = self.matrix.iloc[i].tolist()
            row = [abs(x) for x in row]
            max_element = max(row)
            max_el_index = row.index(max_element)
            if new_matrix[max_el_index] != []:
                return False
            new_matrix[max_el_index] = self.matrix.iloc[i]
            new_vector[max_el_index] = self.vector[i]
        self.vector = new_vector
        self.matrix = pd.DataFrame(data=new_matrix)
        return self.is_diag()


    def calculate_norm(self):
        parameters_sums = []
        for i in range(len(self.vector)):
```

```
        current_row_sum = 0
        a_ii = self.matrix.iloc[i,i]
        for j in range(len(self.vector)):
            if i == j:
                continue
            current_row_sum += abs(self.matrix.iloc[i,j] / a_ii)
        parameters_sums.append(current_row_sum)
    self.__norm = max(parameters_sums)

def solve(self):
    x_old = self.initial_value
    x_new = []
    iter_counter = 0
    precision_vector = []
    precision_not_satisfied = True
    while precision_not_satisfied:
        n = len(self.vector)

        for i in range(n):
            a_ii = self.matrix.iloc[i, i]
            x_i = self.vector[i] / a_ii

            for j in range(0, n):
                if i == j:
                    continue
```

```
                x_i -= self.matrix.iloc[i, j] * x_old[j] / a_ii
            x_new.append(x_i)

        iter_counter += 1
        precision_vector = [abs(x_new[i] - x_old[i])
                            for i in range(len(x_old))]
        if max(precision_vector) < self.precision:
            precision_not_satisfied = False
        x_old = x_new
        x_new = []
    self.__solution = x_old
    self.__precision_vector = precision_vector
    self.__iter_counter = iter_counter

@staticmethod
def read_matrix_from_file(filename):
    matrix_rows = []
    try:
        with open(filename, 'r') as file:
            mat_dim = int(file.readline().strip())
            for _ in range(mat_dim):
                matrix_rows.append([float(x)
                                    for x in file.readline().strip().split()])
            prec = float(file.readline().strip())
            if prec <= 0:
```

```
                print("Precision must be greater than zero")
                raise Exception
        except:
            print("An error occured during file reading. Check file permissons,filename and make sure matrix format is correct")
            raise SystemExit
        matrix, vector = LinearEquationSystem.array_to_matrix(matrix_rows)
        return matrix, vector, prec


    @staticmethod
    def array_to_matrix(matrix_rows):
        try:
            matrix = pd.DataFrame(data=matrix_rows)
            if matrix.shape[0] + 1 != matrix.shape[1]:
                raise Exception
        except:
            print(
                "Matrix has to be square, and B vector must be present as an additional column")
            raise SystemExit
        # Get the last column of matrix, which is B vector.
        vector = np.array(matrix.iloc[:, -1])
        # Remove it, so we're left with a square matrix
        matrix = matrix.iloc[:, :-1]
        return (matrix, vector)
```

```
@staticmethod
def read_matrix_from_console():
    matrix_rows = []
    while True:
        try:
            n = int(input("Please, enter matrix dimentionality: "))
            if n < 2:
                print("Matrix dimentionality must be more or equal to 2")
                raise Exception
            for _ in range(n):
                matrix_rows.append([int(x) for x in input().split()])
            prec = float(input("Please, enter the precision: "))
            if prec <= 0:
                print("Precision must be greater than zero")
                raise Exception
            matrix, vector = LinearEquationSystem.array_to_matrix(
                matrix_rows)
            return matrix, vector, prec
        except KeyboardInterrupt:
            raise SystemExit
        except:
            print("Error encountered, try to check if your data is valid.")
            matrix_rows = []
```

```
def main():
    from_file = input("Would you like to get matrix from file?(y/n): ")
    match from_file:
        case "y":
            filename = input("Type the filename(with extention): ")
            system = LinearEquationSystem(
                *LinearEquationSystem.read_matrix_from_file(filename))
        case _:
            print("Type in the matrix, row by row, separating numbers with spaces")
            system = LinearEquationSystem(
                *LinearEquationSystem.read_matrix_from_console())
    print("Norm is: ", system.norm)
    print("X vector is: ",system.solution)
    print("Precision vector: ", system.precision_vector)
    print("Iteration count is: ", system.iter_counter)



if __name__ == "__main__":
    main()
```

Примеры и результаты работы программы

Пример ввода 1:

```
3
2 -1 3 9
1 -3 -2 0
```

```
3 2 -1 -1
0.1
```

Результат:

```
Norm is:  1.0
X vector is:  [np.float64(0.9997003359016061), np.float64(-1.0683436924807301), np.float64(2.0686433565791242)]
Precision vector:  [np.float64(0.08782765260402536), np.float64(0.09801912881326391), np.float64(0.010191476209238548)]
Iteration count is:  35
```

Пояснения:

Норма равна единице, потому что для каждой строки сумма модулей коэффицентов C_ij = 0.9999999... В силу особенностей хранения чисел с плавающей точкой это значение округляется до 1.

Пример ввода 2:

```
6
1 2 16 3 4 5 115
1 16 2 3 4 5 101
16 1 2 3 4 5 86
1 2 3 16 4 5 128
1 2 3 4 16 5 140
1 2 3 4 5 16 151
0.1
```

Результат:

```
Norm is:  0.9375
```

```
X vector is:  [np.float64(1.046361816247455), np.float64(2.046361816247455),
np.float64(3.046361816247455), np.float64(4.046361816247455),
np.float64(5.046361816247455), np.float64(6.046361816247453)]

Precision vector:  [np.float64(0.09581442024473996),
np.float64(0.09581442024473996), np.float64(0.09581442024474063),
np.float64(0.09581442024474018), np.float64(0.09581442024474018),
np.float64(0.09581442024473841)]

Iteration count is:  122
```

Пояснения:

А что тут пояснять? Всё работает.

Вывод

В ходе работы я узнал о численных методах решения систем линейных уравнений и освежил свои знания ООП в языке Python.