

Университет ИТМО

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Вычислительная математика»

Отчёт

Лабораторная работа №1

Вариант 10

Выполнил:

Сандов Кирилл Алексеевич

P3213

Преподаватель:

Машина Екатерина Алексеевна

Санкт-Петербург, 2024 г

Цель работы

Разработать программу для решения СЛАУ методом простых итераций. В программе численный метод должен быть реализован в виде отдельной подпрограммы/метода/класса, в который исходные/выходные данные передаются в качестве параметров. Размерность матрицы $n \leq 20$ (задается из файла или с клавиатуры по выбору конечного пользователя). Должна быть реализована возможность ввода коэффициентов матрицы, как с клавиатуры, так и из файла (по выбору конечного пользователя). Для метода простых итераций должно быть реализовано:

- Точность задается с клавиатуры/файла
- Проверка диагонального преобладания (в случае, если диагональное преобладание в исходной матрице отсутствует, сделать перестановку строк/столбцов до тех пор, пока преобладание не будет достигнуто). В случае невозможности достижения диагонального преобладания - выводить соответствующее сообщение.
- Вывод вектора неизвестных: x_1, x_2, \dots, x_n .
- Вывод количества итераций, за которое было найдено решение.
- Вывод вектора погрешностей: $|x_i^{(k)} - x_i^{(k-1)}|$.

Описание метода

Итерационные методы - это методы последовательных приближений. Задается некоторое начальное приближение. Далее с помощью определенного алгоритма проводится один цикл вычислений - итерация. В результате итерации находят новое приближение. Итерации проводятся до получения решения с требуемой точностью.

В методе простых итераций сначала исходную СЛАУ:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1, \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \dots \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases}$$

представляют в виде:

$$\begin{cases} x_1 = \frac{a_{12}}{a_{11}}x_2 + \frac{a_{13}}{a_{11}}x_3 + \dots + \frac{a_{1n}}{a_{11}}x_n - \frac{b_1}{a_{11}} \\ x_2 = \frac{a_{21}}{a_{22}}x_1 + \frac{a_{23}}{a_{22}}x_3 + \dots + \frac{a_{2n}}{a_{22}}x_n - \frac{b_2}{a_{22}} \\ \dots \dots \\ x_n = \frac{a_{n1}}{a_{nn}}x_1 + \frac{a_{n2}}{a_{nn}}x_2 + \dots + \frac{a_{n-1n-1}}{a_{nn}}x_{n-1} - \frac{b_n}{a_{nn}} \end{cases}$$

Обозначив:

$$c_{ij} = \begin{cases} 0, & \text{при } i = j \\ -\frac{a_{ij}}{a_{ii}}, & \text{при } i \neq j \end{cases}$$

$$d_i = \frac{b_i}{a_{ii}} \quad i = 1, 2, \dots, n$$

Получим СЛАУ и запишем в сокращенном виде:

$$x_i = \sum_{j=1}^n c_{ij}x_j + d_i, \quad i = 1, 2, \dots, n$$

Рабочая формула метода простых итераций:

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

С каждой k-ой итерацией решения будут приближаться к действительному решению СЛАУ. Важным критерием этого является то, что матрица коэффициентов СЛАУ должна иметь диагональное преобладание.

Код программы

https://github.com/amphyxs/Computational-Math-2024/tree/main/P3213/Sandow_367527/lab1

Листинг части программы, реализующей сам метод

```
/** Максимальное допустимое N. */
export const MAX_N = 20;

/**
 * Результат решения СЛАН.
 *
 * @property solution - вектор найденных неизвестных.
 * @property iterationsCount - количество итераций.
 * @property errorVectors - векторы погрешностей для каждой итерации.
 */
export interface Result {
  solution: number[];
  iterationsCount: number;
  errorVectors: number[][];
}

/**
 * Получить решение СЛАН с подсчётом кол-ва итераций и погрешностей на каждой итерации.
 *
 * @throws {DiagonallyDominationUnobtainable} - если невозможно достичь диагонального преобладания.
 */
export function solveLinearEquationsSystem(data: InputData): Result {
  if (!obtainDDFrom(data)) {
    throw new DiagonallyDominationUnobtainable();
  }

  bringToNormalCoefficients(data);
  // TODO Решение СЛАН
  let lastVector = data.rightValues;
  let iterations = 0;
  let errorVectors: number[][] = [];
  let maxDelta = 1000000000;
  while (maxDelta > data.epsilon) {
    errorVectors.push([]);
    let curVector = calcCurrentVector(lastVector, data);
    maxDelta = 0;
    for (let i = 0; i < data.size; i++) {
      const delta = Math.abs(lastVector[i] - curVector[i]);
      errorVectors[iterations].push(delta);
      maxDelta = Math.max(maxDelta, delta);
    }
    lastVector = curVector;
    iterations++;
  }

  return {
    solution: lastVector,
    iterationsCount: iterations,
    errorVectors: errorVectors
  };
}

/** Проверить достижимо ли диагональное преобладание матрицы и если да, то привести ее к такой форме */
function obtainDDFrom(input: InputData): boolean {
  let origMatrix = input.matrix;
  let sumAll = origMatrix.reduce((totalSum, row) => {
    return totalSum + row.reduce((rowSum, num) => rowSum + num, 0);
  }, 0);
  const allPermutations = generatePermutations(input.size);
  for (let comb of allPermutations) {
    if (isDiagonallyDominating(origMatrix, comb, sumAll)) {
      rearrangeMatrixToDDForm(input, comb);
      return true;
    }
  }
  return false;
}

/** Сгенерировать все перестановки чисел от 0 до n-1 */
function generatePermutations(n: number): number[][] {
  const permutations: number[][] = [];
  const nums: number[] = [];
```

```

// Заполнение массива числами от 0 до n - 1
for (let i = 0; i < n; i++) {
    nums.push(i);
}
// Рекурсивная функция для генерации перестановок
function generate(nums: number[], index: number): void {
    if (index === nums.length) {
        permutations.push(nums.slice()); // Добавляем текущую перестановку в массив
        return;
    }
    for (let i = index; i < nums.length; i++) {
        // Меняем местами текущий элемент с элементом на позиции index
        [nums[index], nums[i]] = [nums[i], nums[index]];
        generate(nums, index + 1); // Рекурсивный вызов для следующей позиции
        // Возвращаем массив к исходному состоянию перед следующей итерацией
        [nums[index], nums[i]] = [nums[i], nums[index]];
    }
}

generate(nums, 0); // Начинаем генерацию с индекса 0
return permutations;
}

/** Проверить диагональное преобладание матрицы. */
function isDiagonallyDominating(origMatrix: number[][], comb: number[], sumAll: number): boolean {
    let sum = 0;
    for (let i = 0; i < origMatrix.length; i++) {
        sum += origMatrix[i][comb[i]];
    }
    return sum * 2 >= sumAll;
}

/** Переставить строки в матрице и элементы в числах справа в нужном порядке. */
function rearrangeMatrixToDDForm(data: InputData, comb: number[]): void {
    let rearrangedMatrix: number[][] = [];
    let rearrangedValues: number[] = [];
    for (let i = 0; i < data.size; i++) {
        rearrangedMatrix.push([]);
        rearrangedValues.push(0);
    }

    for (let i = 0; i < data.size; i++) {
        const index = comb[i];
        rearrangedMatrix[index] = data.matrix[i];
        rearrangedValues[index] = data.rightValues[i]
    }
    data.matrix = rearrangedMatrix;
    data.rightValues = rearrangedValues;
}

/** Разделить все числа в строке i (в т ч rightValue[i]) на диагональное число matrix[i][i]
 * Диагональные коэффициенты в итоге = 1 */
function bringToNormalCoefficients(data: InputData): void {
    for (let i = 0; i < data.size; i++) {
        const k = data.matrix[i][i];
        for (let j = 0; j < data.size; j++) {
            data.matrix[i][j] /= k;
        }
        data.rightValues[i] /= k;
    }
}

/** Заполнить значениями новый вектор на i-той итерации, опираясь на значения (i-1)-ой итерации и
коэффициенты*/
function calcCurrentVector(lastVector: number[], input: InputData): number[] {
    let curVector: number[] = [];
    for (let i = 0; i < input.size; i++) {
        let val = input.rightValues[i];
        for (let j = 0; j < input.size; j++) {
            if (i !== j) {
                val -= input.matrix[i][j] * lastVector[j];
            }
        }
        curVector.push(val);
    }
    return curVector;
}

```

```
/** Ошибка о том, что невозможно достичь диагонального преобладания исходной матрицы. */  
export class DiagonallyDominationUnobtainable extends Error {  
  constructor() {  
    super();  
    this.name = 'DiagonallyDominationUnobtainable';  
  }  
}
```


Пример работы программы

Входные данные:

3
2 2 10 14
10 1 1 12
2 10 1 13
0.01

Результат:

Вектор неизвестных:

(0.9996, 0.9995, 0.9993)

Количество итераций:

5

Векторы погрешностей:

k	Вектор погрешностей
1	(0.2700, 0.3800, 0.5000)
2	(0.0880, 0.1040, 0.1300)
3	(0.0234, 0.0306, 0.0384)
4	(0.0069, 0.0085, 0.0108)
5	(0.0019, 0.0025, 0.0031)

Вывод

В результате выполнения данной лабораторной работы был изучен метод простых итераций для решения СЛАУ. Основное преимущество данного метода: он прост в реализации на ЭВМ и является универсальным. Однако он имеет и недостатки: в результате получается неточное решение (точность задается пользователем), итераций может быть очень много, матрица коэффициентов СЛАУ должна иметь диагональное преобладание.