Университет ИТМО

Факультет: ПИиКТ

Дисциплина: Вычислительная математика

Лабораторная работа №1

Вариант “Метод простых итераций”

Выполнили: **Кудлаков Роман**

Группа: **P3231**

Преподаватель: **Перл Ольга Вячеславовна**

Санкт-Петербург, 2021

Метод простой итерации, называемый также **методом** последовательного приближения, - это математический алгоритм нахождения значения неизвестной величины путем постепенного ее уточнения.
Первым действием пытаемся привести матрицу к диагональному преобладанию. Для этого можно использовать перестановки строк и столбцов, а также матричные преобразования.
Если диагональное преобразование было получено, то теперь в каждой строке разделим все элементы на элемент, стоящий на диагонали.
Теперь перенесем все элементы из левой части уравнения в правую, кроме диагональных элементов.
Далее, чтобы воспользоваться методом нужно проверить выполняется ли одно из достаточных условий сходимости в правой части уравнения (столбец свободных членов не учитывается):

1. $max \sum_{j=1}^{n} \left|a_{ij}\right| < 1\,,\ \text{при}\ 1 \le i \le n$
2. $max \sum_{i=1}^{n} \left|a_{ij}\right| < 1\,,\ \text{при}\ 1 \le j \le n$
3. $\sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}^2} < 1$

На следующем шаге обозначим наше начальное приближение присвоив ему столбец свободных членов.
Теперь подставим в правую часть каждого уравнения значение из начального приближения и получим новые значения неизвестных. Дальше будем подставлять уже новые, только что полученные значения, в уравнения.
Так будем продолжать делать до тех пор, пока не получим точность меньше, чем данная.

Расчет точности на i-том шаге:

$$max \sum_{i=1}^{n} |x_i - x_{i-1}|$$

Листинг кода

```
class ProblemAnswer {
public:
    ProblemAnswer(vector<double> unknownVariables, int
numOfIterations, vector<double> inaccuracy) {
        setUnknownVariables(unknownVariables);
        setNumOfIterations(numOfIterations);
        setInaccuracy(inaccuracy);
    }
    ProblemAnswer(int numOfIterations){
        setNumOfIterations(numOfIterations);
    }
```

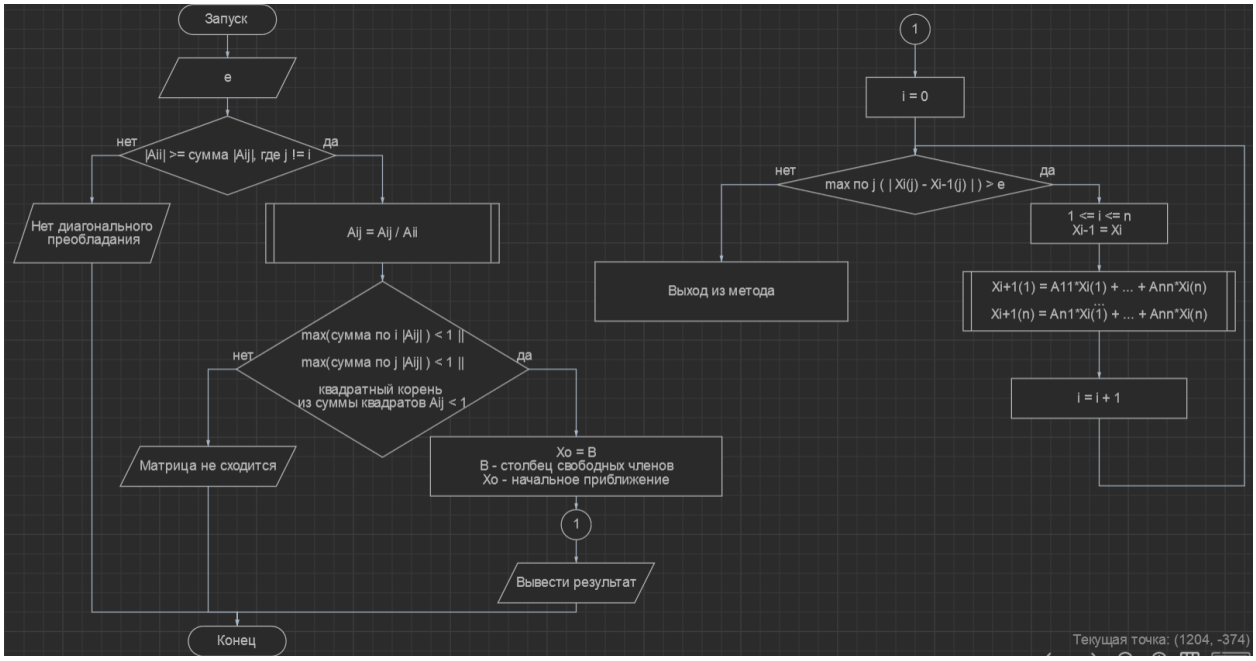
```
    void setUnknownVariables(vector<double> &unknownVariables) {
        this->unknownVariables = unknownVariables;
    }
    void setNumOfIterations(int numOfIterations) {
        this->numOfIterations = numOfIterations;
    }
    void setInaccuracy(vector<double> inaccuracy) {
        this->inaccuracy = inaccuracy;
    }
    vector<double> getUnknownVariables() {
        return unknownVariables;
    }
    int getNumOfIterations() {
        return numOfIterations;
    }
    vector<double> getInaccuracy() {
        return inaccuracy;
    }
    void printAnswer(int precision) {
        talkToUser("Number of Iterations:");
        cout << this->numOfIterations << "\n";
        talkToUser("UnknownVariables:");
        printRow(this->unknownVariables, precision);
        talkToUser("InaccuracyOfResult:");
        printRow(this->inaccuracy, precision);
    }
private:
    vector<double> unknownVariables;
    int numOfIterations;
    vector<double> inaccuracy;
};


int findMaxAbsInRow(vector<double> &row) {
    double max = abs(row[0]);
    int pos = 0;
    for (int i = 0; i < row.size() - 1; ++i) {
        if (max < abs(row[i])) {
            max = abs(row[i]);
            pos = i;
        }
    }
    return pos;
}

void swapColumns(vector<vector<double>> &matrix, int firstColumnNum,
int secondColumnNum) {
    if (firstColumnNum == secondColumnNum) return;
    for (int i = 0; i < matrix.size(); ++i) {
        double temp = matrix[i][firstColumnNum];
        matrix[i][firstColumnNum] = matrix[i][secondColumnNum];
        matrix[i][secondColumnNum] = temp;
    }
}

bool tryToMakeDiagonalPredominance(vector<vector<double>> &matrix) {

    for (int thisRow = 0; thisRow < matrix.size() - 1; ++thisRow) {
```

```
        int columnNum = findMaxAbsInRow(matrix[thisRow]);
        if (matrix[thisRow][columnNum] == 0 || columnNum < thisRow)
return false;
        swapColumns(matrix, thisRow, columnNum);
      }
    return true;
}

void divideByDiagonal(vector<vector<double>> &matrix) {
    for (int i = 0; i < matrix.size(); ++i) {
        for (int j = 0; j < matrix[i].size(); ++j) {
            if (i == j) { continue; }
            matrix[i][j] /= matrix[i][i];
        }
        matrix[i][i] = 0;
    }
}

double countConvergeCoefByRows(vector<vector<double>> &matrix) {
    double max = 0;

    for (int i = 0; i < matrix.size(); ++i) {
        double sum = 0;
        for (int j = 0; j < matrix.size(); ++j) {
            sum += matrix[i][j];
        }
        if (max < sum) max = sum;
    }
    if (max < 1) return max;
    return 0;
}

double countConvergeCoefByColumns(vector<vector<double>> &matrix) {
    double max = 0;

    for (int j = 0; j < matrix.size(); ++j) {
        double sum = 0;
        for (int i = 0; i < matrix.size(); ++i) {
            sum += matrix[i][j];
        }
        if (max < sum) max = sum;
    }
    if (max < 1) return max;
    return 0;
}

double countConvergeCoefByEachValue(vector<vector<double>> &matrix) {
    double sum = 0;
    for (int i = 0; i < matrix.size(); ++i) {
        for (int j = 0; j < matrix.size(); ++j) {
            sum += pow(matrix[i][j], 2);
        }
    }
    sum = sqrt(sum);
    if (sum < 1) return sum;
    return 0;
}
```

```
double countConvergeCoef(vector<vector<double>> &matrix) {
    double convergeCoef = 0;
    convergeCoef = countConvergeCoefByRows(matrix);
    if (convergeCoef) return convergeCoef;
    convergeCoef = countConvergeCoefByColumns(matrix);
    if (convergeCoef) return convergeCoef;
    convergeCoef = countConvergeCoefByEachValue(matrix);
    return convergeCoef;
}

vector<double> makeIteration(vector<vector<double>> &matrix,
vector<double> &prevResults) {
    vector<double> results;
    for (int i = 0; i < matrix.size(); ++i) {
        double sum = 0;
        for (int j = 0; j < matrix.size(); ++j) {
            sum += matrix[i][j] * prevResults[j];
        }
        sum = (-sum) + matrix[i][matrix.size()];
        results.push_back(sum);
    }
    return results;
}

double countPrecision(vector<double> &prevResults, vector<double>
&currentResults) {
    double max = abs(prevResults[0] - currentResults[0]);
    for (int i = 1; i < prevResults.size(); ++i) {
        if (max < abs(prevResults[0] - currentResults[0])) {
            max = abs(prevResults[0] - currentResults[0]);
        }
    }
    return max;
}

vector<double> countInaccuracyVector(vector<double> &result,
vector<double> &prevStep, double convergeCoef) {
    vector<double> inaccuracyVector;
    for (int i = 0; i < result.size(); ++i) {
        double inaccuracy = convergeCoef / (1 - convergeCoef) *
abs(result[i] - prevStep[i]);
        inaccuracyVector.push_back(inaccuracy);
    }
    return inaccuracyVector;
}

ProblemAnswer simpleOperationMethod(vector<vector<double>> &matrix,
int precision) {

    double convergeCoef = countConvergeCoef(matrix);
    if (!convergeCoef) {
        ProblemAnswer answer(0);
        return answer;
    }

    vector<double> currentResults(matrix.size(), 0);
```

```
    for (int i = 0; i < matrix.size(); ++i) {
        currentResults[i] = matrix[i][matrix.size()];
    }
    int numOfIterations = 0;
    double currentPrecision = 2;
    vector<double> previousResults;
    while (currentPrecision >= pow(0.1, precision)) {
        previousResults = currentResults;
        currentResults = makeIteration(matrix, previousResults);
        currentPrecision = countPrecision(previousResults,
currentResults);
        ++numOfIterations;
    }
    vector<double> inaccuracy = countInaccuracyVector(currentResults,
previousResults, convergeCoef);

    ProblemAnswer answer(currentResults, numOfIterations, inaccuracy);
    return answer;
}
```

Блок-схема алгоритма



Тесты и результаты

Тест 1.

```
Matrix
|    6.33     0.12     -2.2      1.6    -1.36    14.21 |
|    1.22   -8.567    1.566     3.11      2.4      -11 |
|    1.04    -3.63    -7.75     1.17        1       13 |
|     2.1    -2.34    -1.15     7.51    -1.11       23 |
|    1.14     2.13    -4.25    -0.13    -8.63     21.4 |
Diagonal Predominance
|    6.33     0.12     -2.2      1.6    -1.36    14.21 |
|    1.22   -8.567    1.566     3.11      2.4      -11 |
|    1.04    -3.63    -7.75     1.17        1       13 |
|     2.1    -2.34    -1.15     7.51    -1.11       23 |
|    1.14     2.13    -4.25    -0.13    -8.63     21.4 |
Number of Iterations:
19
UnknownVariables:
|  0.5045    1.789   -2.119     3.01  -0.9736 |
InaccuracyOfResult:
|0.002591 0.0009665 0.001656  0.00252 0.001515 |
TOTAL:
|-0.0003142 -0.0001558 -0.0002495 -0.0003688 -0.0002485 |
```

Тест 2.

```
Matrix
|   4.394   -1.758     -4.9    3.887   -20.99    2.445    41.31 |
|   0.782    -3.37   -1.689   -16.14   -2.707    7.091    55.52 |
|  -4.041   -4.639   -3.256   -1.705    3.478   -18.36   -33.52 |
|  -3.067    22.07   -2.923    4.125   -4.559    6.789    33.15 |
|   1.442   -2.398   -17.63   -3.801   -2.002     5.34   -46.17 |
|  -28.73    0.295   -4.396   -5.794   -7.715     8.79    49.72 |
Diagonal Predominance
|  -20.99    3.887    2.445   -1.758     -4.9    4.394    41.31 |
|  -2.707   -16.14    7.091    -3.37   -1.689    0.782    55.52 |
|   3.478   -1.705   -18.36   -4.639   -3.256   -4.041   -33.52 |
|  -4.559    4.125    6.789    22.07   -2.923   -3.067    33.15 |
|  -2.002   -3.801     5.34   -2.398   -17.63    1.442   -46.17 |
|  -7.715   -5.794     8.79    0.295   -4.396   -28.73    49.72 |
Number of Iterations:
13
UnknownVariables:
|  -3.619   -3.364   0.4933    1.649    3.641  -0.4697 |
InaccuracyOfResult:
|0.001336 0.007257  4.3e-05 0.004341 0.006501 0.008559 |
TOTAL:
|  -124.9   -87.64    80.62   -110.7    24.26    9.315 |
```

Вывод.

В ходе лабораторной работы изучен метод простых итераций. Данный метод используется только для СЛАУ, в которых выполняется один из достаточных признаков сходимости. Сложность данного метода $O(N^2*k)$, где k - количество итераций.

Плюсы данного метода в том, что можно получить результат максимально близкий к точному, но чем больше точность, тем больше итераций данному методу надо будет совершить. Также на больших размерах матриц метод простых итераций будет быстрее нежели прямые методы.

Минусы метода в том, что на маленьких размерах матрицы он работает очень долго по сравнению с прямыми методами. Обязательно должен выполняться хотя бы один признак сходимости, таким образом прямые методы могут находить решения в большем количестве матриц, чем итерационные методы.