Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1
Задание №3, часть II по курсу «Численные методы»

Студент: С. М. Власова
Преподаватель: И. Э. Иванов
Группа: М8О-306Б
Дата:
Оценка:
Подпись:

Москва, 2020

Задание №1.3

Реализовать метод Зейделя в виде програмы, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.
Вариант: 12

$$\begin{cases} 14 \cdot x_1 - 4 \cdot x_2 - 2 \cdot x_3 + 3 \cdot x_4 = 38 \\ -3 \cdot x_1 + 23 \cdot x_2 - 6 \cdot x_3 - 9 \cdot x_4 = -195 \\ -7 \cdot x_1 - 8 \cdot x_2 + 21 \cdot x_3 - 5 \cdot x_4 = -27 \\ -2 \cdot x_1 - 2 \cdot x_2 + 8 \cdot x_3 + 18 \cdot x_4 = 142 \end{cases}$$

1 Описание метода решения

Метод простых итераций довольно медленно сходится. Для его ускорения существует **метод Зейделя**, заключающийся в том, что при вычислении компонента x_i^{k+1} вектора неизвестных на $(k+1)$-й итерации используются x_1^{k+1}, $x_2^{k+1}, \ldots, x_{i-1}^{k+1}$, уже вычисленные на $(k+1)$-й итерации.
Значения остальных компонент $x_{i+1}^{k+1}, x_{i+2}^{k+1}, \ldots, x_n^{k+1}$ берутся из предыдущей итерации.
Так же, как и в методе простых итераций, строится эквивалентная СЛАУ и за начальное приближение принимается вектор правых частей $x^{(0)} = (\beta_1 \beta_2 \ldots \beta_n)$.
Тогда метод Зейделя для известного вектора $(x_1^k x_2^k \ldots x_n^k)^T$ на $k-$й итерации имеет вид:

$$\begin{cases} x_1^{k+1} = \beta_1 + \alpha_{11} \cdot x_1^k + \alpha_{12} \cdot x_2^k + \ldots + \alpha_{1n} \cdot x_n^k \\ x_2^{k+1} = \beta_2 + \alpha_{21} \cdot x_1^{k+1} + \alpha_{22} \cdot x_2^k + \ldots + \alpha_{2n} \cdot x_n^k \\ x_3^{k+1} = \beta_3 + \alpha_{31} \cdot x_1^{k+1} + \alpha_{32} \cdot x_2^{k+1} + \alpha_{33} \cdot x_3^k + \ldots + \alpha_{3n} \cdot x_n^k \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n^{k+1} = \beta_n + \alpha_{n1} \cdot x_1^{k+1} + \alpha_{n2} \cdot x_2^{k+1} + \ldots + \alpha_{nn-1} \cdot x_n^{k+1} + \alpha_{nn} \cdot x_n^k \end{cases}$$

Из этой системы видно, что $x^{k+1} = \beta + B \cdot x^{k+1} + C \cdot x^k$, где B — нижняя треугольная матрица с диагональными элементами, равными нулю, а C — верхняя треугольная матрица с диагональными элементами, отличными от нуля, $\alpha = B + C$.

Следовательно,

$$(E - B) \cdot x^{k+1} = C \cdot x^k + \beta,$$

откуда

$$x^{k+1} = (E - B)^{-1} \cdot C \cdot x^k + (E - B)^{-1} \cdot \beta.$$

Таким образом, метод Зейделя является методом простых итераций с матрицей правых частей $\alpha = (E-B)^{-1} \cdot C$ и вектором правых частей $(E-B)^{-1} \cdot \beta$ и, следовательно, сходимость и погрешность метода Зейделя можно исследовать с помощью формул, выведенных для метода простых итераций, в которых вместо матрицы α подставлена матрица $(E - B)^{-1} \cdot C$, а вместо вектора правых частей — вектор $((E - B)^{-1} \cdot \beta$.

Для практических вычислений важно, что в качестве достаточных условий сходимости метода Зейделя могут быть использованы условия, приведенные для метода простых итераций ($||\alpha|| < 1$ или диагональное преобладание матрицы A).
В случае выполнения этих условий для оценки погрешности на $k-$й итерации можно

использовать выражение

$$\varepsilon^k = \frac{||C||}{1 - ||\alpha||} \cdot ||x^{(k)} - x^{(k-1)}||.$$

2 Протокол

Входные данные я храню в файле *data3*: первая строка — размерность матрицы, на следующих строках — матрица СЛАУ, коэффициенты вектора правых частей и заданная точность.
Выходные данные я записываю в файл *zeydres*.
Скриншот консоли:

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ g++ zeyd.cpp -o zeyd
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat data3
4
14 -4 -2 3
-3 23 -6 -9
-7 -8 21 -5
-2 -2 8 18
38 -195 -27 142
0.01
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ ./zeyd < data3 > zeydres
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat zeydres
Число итераций метода Зейделя: 6
  x_1 = -1.00018
  x_2 = -6.0001
  x_3 = -2.00009
  x_4 = 8.00001
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$
```

Точное решение СЛАУ:

$$x^* = \begin{pmatrix} -1 \\ -6 \\ -2 \\ 8 \end{pmatrix}$$

Можно видеть, что программа корректно вычислила решение системы с точностью $\varepsilon = 0.01$. Ей потребовалось 6 итераций, что на 3 итерации меньше метода простых итераций.

$$x = \begin{pmatrix} -1.00018 \\ -6.0001 \\ -2.00009 \\ 8.00001 \end{pmatrix}$$

Попробуем найти оценку погрешности, при которой алгоритм найдет приблизительное решение, совпадающее с точным решением. Для этого будем уменьшать точность.

Пусть $\varepsilon = 0.0001$.
Скриншот консоли:

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat data3
4
14 -4 -2 3
-3 23 -6 -9
-7 -8 21 -5
-2 -2 8 18
38 -195 -27 142
0.0001
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ ./zeyd < data3 > zeydres
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat zeydres
Число итераций метода Зейделя: 10
  x_1 = -1
  x_2 = -6
  x_3 = -2
  x_4 = 8
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$
```

Можно видеть, что найденное решение СЛАУ совпало с точным решением. Таким образом, метод Зейделя находит точное решение при меньшем значении оценки погрешности. Для достижения такого результата алгоритму потребовалось 10 итераций.

$$x^* = \begin{pmatrix} -1 \\ -6 \\ -2 \\ 8 \end{pmatrix} = x$$

3 Исходный код

Т.к. от листинга кода предыдущего метода код метода Зейделя отличается только одной функцией, я приведу только ее.

Листинг 1: Метод Зейделя

```
int ZMethod(std::vector<std::vector<double>>& Matrix, std::vector<double>& X, double eps)
{
    int n = Matrix.size();
    int it_count = 0;
    double norma_c, norma_alpha = 0;
    double kaf = 0;
    double sum_alpha, sum_c = 0;
    std::vector<double> X_1(n, 0);  //extra vector, storing the result of previous iteration

    for( int i = 0; i < n; i++)   //leading matrix to x = beta + alpha * x
    {
        if( Matrix[i][i] == 0)
            Swap(Matrix, i);
        for( int j = 0; j <= n; j++)
        {
            if(j == i)
                continue;
            else
            {
                if( j != n)
                {
                    Matrix[i][j] = (-1)* Matrix[i][j]/ Matrix[i][i];
                    sum_alpha += fabs(Matrix[i][j]);
                    if( j > i)
                        sum_c += fabs(Matrix[i][j]);
                }
                else
                    Matrix[i][j] /= Matrix[i][i];
            }
        }
        if( sum_alpha > norma_alpha)    //norma of Alpha matrix
        {
            norma_alpha = sum_alpha;
            sum_alpha = 0;
        }
        if( sum_c > norma_c ) //norm of the upper triangular matrix
        {
```

```
38              norma_c = sum_c;
39              sum_c = 0;
40          }
41          Matrix[i][i] = 0;
42      }
43      norma_c /= (1 - norma_alpha);    //coefficient estimate of error
44      for(int i = 0; i < n; i++)
45          X_1[i] = Matrix[i][n];
46
47      while(true)
48      {
49          for(int i = 0; i < n; i++)
50          {
51              X[i] = 0;
52              for(int j = 0; j < n; j++)
53              {
54                  if( j < i)
55                  {
56                      X[i] += Matrix[i][j]*X[j];
57                  }
58                  else
59                  {
60                      X[i] += Matrix[i][j]* X_1[j];
61                  }
62              }
63              X[i] += Matrix[i][n];
64              if( kaf < fabs(X[i] - X_1[i]))
65              kaf = fabs(X[i] - X_1[i]);
66          }
67          it_count++;
68          if(eps > kaf*norma_c) //estimate of error
69              break;
70          X_1 = X;
71          kaf = 0;
72      }
73      return it_count;
74  }
```

4 Выводы

Выполнив вторую часть третьего задание первой лабораторной работы, я познакомилась с итерационным методом поиска решений СЛАУ — методом Зейделя. Он является ускоренной модификацией метода простых итераций. Это достигается за счет ускорения сходимости вычисленных решений — при вычислении компонент приближенных векторов-значений на каждой итерации учитываются уже вычисленные на этой итерации компоненты, таким образом, ускоряя процесс сходимости.