

# Практическая задача, пункт а

Натальченко Александр

11 ноября 2020 г.

## Содержание

1. Постановка задачи
2. Краткое описание алгоритма
3. Результаты
4. Код программы

## 1 Постановка задачи

12. ПРАКТИЧЕСКАЯ ЗАДАЧА 3 курс 5 семестр

Решить систему

$$\left\{ \begin{array}{lcl} c_1 x_1 + d_1 x_2 + e_1 x_3 & = & f_1 \\ b_2 x_1 + c_2 x_2 + d_2 x_3 + e_2 x_4 & = & f_2 \\ a_3 x_1 + b_3 x_2 + c_3 x_3 + d_3 x_4 + e_3 x_5 & = & f_3 \\ a_4 x_2 + b_4 x_3 + c_4 x_4 + d_4 x_5 + e_4 x_6 & = & f_4 \\ \dots & \dots & \dots \\ a_m x_{m-2} + b_m x_{m-1} + c_m x_m + d_m x_{m+1} + e_m x_{m+2} & = & f_m \\ \dots & \dots & \dots \\ a_{n-1} x_{n-3} + b_{n-1} x_{n-2} + c_{n-1} x_{n-1} + d_{n-1} x_n & = & f_{n-1} \\ a_n x_{n-2} + b_n x_{n-1} + c_n x_n & = & f_n \end{array} \right.$$

$$b_{i+1} = d_i = 1, i = 1, \dots, n-1; a_{i+2} = e_i = 0, i = 1, \dots, n-2.$$

**а. прямым методом (методом Гаусса):**  
вывести на печать решение и невязки;

**б. итерационным методом (методом Зейделя) с точностью  $\varepsilon = 10^{-4}$ :**  
вывести на печать решение;

**с. определить число обусловленности матрицы системы  $\mu = \|A\| \cdot \|A^{-1}\|$**

(УКАЗАНИЕ: найти  $\lambda_{\max}$  степенным методом, затем  $\lambda_{\min}$ , используя сдвиг спектра матрицы, для определения числа обусловленности воспользоваться евклидовой нормой).

## 2 Краткое описание алгоритма

Сначала задаём значения элементов расширенной матрицы системы (с. 21-30), в матрице  $A_0$  мы будем хранить её, а в матрице  $A$  будем её преобразовывать.

Прямым ходом метода Гаусса (с. 34-65) мы «обнуляем» элементы под нижней диагональю матрицы, затем обратным ходом (с. 67-73) «обнуляем» — под верхней. Причём в обратном ходе вовсе не обязательно (с. 71) преобразовывать верхнюю часть матрицы, так как это ни на что не влияет. Достаточно работать только с последним столбцом (с. 70). Затем получаем вектор-решение (с. 75-81), вектор-невязку (с. 83-92) и считаем отношение их норм (с. 95-100).

### 3 Результаты

```

Terminal
x_1 = 0.08333333
x_2 = 0.16666667
x_3 = 0.25000000
x_4 = 0.33333333
x_5 = 0.41666667
x_6 = 0.50000000
x_7 = 0.58333333
x_8 = 0.66666667
x_9 = 0.75000000
x_10 = 0.83333333
x_11 = 0.91666667
x_12 = 1.00000000
x_13 = 1.08333331
x_14 = 1.16666685
x_15 = 1.24999814
x_16 = 1.33335174
x_17 = 1.41648441
x_18 = 1.50180413
x_19 = 1.56547433
x_20 = 1.84345257

Terminal
r_1 = -1.38777878e-16
r_2 = 2.49800181e-16
r_3 = 6.10622664e-16
r_4 = 4.44089210e-16
r_5 = -2.22044605e-16
r_6 = 1.11022302e-15
r_7 = -1.33226763e-15
r_8 = -4.44089210e-16
r_9 = -1.66533454e-15
r_10 = 2.22044605e-16
r_11 = 2.22044605e-16
r_12 = -1.33226763e-15
r_13 = -2.22044605e-15
r_14 = -1.77635684e-15
r_15 = -6.66133815e-16
r_16 = -1.99840144e-15
r_17 = 0.00000000e+00
r_18 = -2.88657986e-15
r_19 = -1.33226763e-15
r_20 = -3.55271368e-15

Отношение нормы невязки к норме решения 1.46179490e-15
Press <RETURN> to close this window...

```

Мы получили решение и невязки, и вычислили отношение нормы невязки к норме решения:

$$\frac{\|r\|}{\|x^*\|} = 1.4618 \cdot 10^{-15}$$

Поскольку число обусловленности матрицы системы, которое требовалось рассчитать в другом пункте, равно  $\mu = 1.48 \sim 1$ , то следует ожидать, что отношение нормы невязки к норме решения по порядку совпадёт с машинным эpsilonом:

$$\frac{\|r\|}{\|x^*\|} \sim \varepsilon_m$$

Для типа double в языке C++ величина машинного epsilon составляет  $\varepsilon = 2^{-52}$

И действительно,

$$\frac{1.4618 \cdot 10^{-15}}{2^{-52}} \approx 6.6,$$

значит, как и ожидалось,

$$\frac{\|r\|}{\|x^*\|} \approx 7\varepsilon_m.$$

## 4 Код программы (C++)

```
1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4 #include <iomanip>
5
6 const double c_i = 10; // Значения элементов матрицы на главной диагонали
7 const double d_i = 1; // Значения элементов матрицы на соседних диагоналях
8 const size_t N = 20; // Размер матрицы
9 const double epsilon = 1e-10;
10
11 using namespace std;
12
13 inline bool equals0(double x) {
14     return(fabs(x) < epsilon);
15 }
16
17 int main() {
18     vector<vector<double> > A_0(N), A(N);
19
20     // Задаём значения элементов матрицы
21     for (size_t i = 0; i < N; i++) {
22         A_0[i].resize(N + 1);
23         A[i].resize(N + 1);
24         A_0[i][i] = c_i;
25         if (i < N - 1)
26             A_0[i][i + 1] = d_i;
27         if (i > 0)
28             A_0[i][i - 1] = d_i;
29         A_0[i][N] = i + 1;
30     }
31
32     A = A_0;
33
34     //Прямой ход метода Гаусса
35     for (size_t i = 0; i < N; i++) {
36
37         //Если диагональный элемент равен нулю с точностью 1e-10,
38         //пытаемся переставить строки
39         if (equals0(A[i][i])) {
40             vector<double> tmp(N + 1);
41             tmp = A[i];
42             for (size_t j = i + 1; j < N; j++) {
43                 if (!equals0(A[j][i])) {
44                     A[i] = A[j];
45                     A[j] = tmp;
46                 }
47             }
48         }
49
50         if (equals0(A[i][i])) {
51             return 0;
52         } else { //Если диагональный элемент не равен нулю
53             //Делим элементы строки на диагональный элемент
54             for (size_t j = N; j >= i && j <= N; j--) {
```

```

55         A[i][j] /= A[i][i];
56     }
57
58     // Вычитаем эту строку из нижних с коэффициентами A[i][j]
59     for (size_t j = i + 1; j < N; j++) {
60         for (size_t k = N; k >= i && k + 1 != 0; k--) {
61             A[j][k] -= A[i][k]*A[j][i];
62         }
63     }
64 }
65 }
66
67 //Обратный ход метода Гаусса
68 for (size_t j = N - 1; j + 1 != 0; j--) {
69     for (size_t i = j - 1; i + 1 != 0; i--) {
70         A[i][N] -= A[j][N]*A[i][j];
71         //A[i][j] = 0;
72     }
73 }
74
75 //Выводим решение
76 vector<double> X(N);
77 for (size_t i = 0; i < N; i++) {
78     X[i] = A[i][N];
79     cout << fixed << setprecision(8)
80         << "x_" << i + 1 << " = " << X[i] << endl;
81 }
82
83 //Выводим невязки
84 vector<double> r(N);
85 for (size_t i = 0; i < N; i++) {
86     r[i] = A_0[i][N];
87     for (size_t j = 0; j < N; j++) {
88         r[i] -= A_0[i][j]*X[j];
89     }
90     cout << "r_" << i + 1 << " = " << scientific
91         << r[i] << endl;
92 }
93 cout << endl;
94
95 //Считаем квадраты норм
96 double NX = 0, Nr = 0;
97 for (size_t i = 0; i < N; i++) {
98     NX += X[i]*X[i];
99     Nr += r[i]*r[i];
100 }
101
102 cout << "Отношение нормы невязки к норме решения " << sqrt(Nr/NX) << endl;
103 return 0;
104 }

```