Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №1
Задание №4 по курсу «Численные методы»

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

Москва, 2020

Задание №1.4

Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.
Вариант: 12

$$\begin{pmatrix} 7 & 3 & -1 \\ 3 & -7 & -8 \\ -1 & -8 & -2 \end{pmatrix}$$

1 Описание метода решения

Метод вращений Якоби численного решения задач на собственные значения и собственные векторы матриц

Метод вращений Якоби применим только для симметрических матриц $A_{n \cdot n}$ ($A = A^T$) и решает полную проблему *собственных значений* и *собственных векторов* таких матриц.
Он основан на отыскании с помощью итерационных процедур матрицы U в преобразовании подобия $\Lambda = U^{-1} \cdot A \cdot U$, а поскольку для симметрических матриц A матрица преобразования подобия U является ортогональной $U^{-1} = U^T$, то $\Lambda = U^T \cdot A \cdot U$, где Λ — диагональная матрица с собственными значениями на главной диагонали.

$$\Lambda = \begin{pmatrix} \lambda_1 & \dots & 0 \\ \vdots & \ddots & \dots \\ 0 & \dots & \lambda_n \end{pmatrix};$$

U — матрица преобразования, столбцы которой являются собственными векторами матрицы A, соответствующие ее собственным значениям.

Пусть дана симметрическая матрица A. Требуется для нее вычислить с точностью ε все собственные значения и соответствующие им собственные векторы. Алгоритм метода вращения следующий:

Пусть известна матрица $A^{(k)}$ на $k^{\smile}$й итерации, при этом для $k = 0$ $A^{(0)} = A$.

1) Выбирается максимальный по модулю недиагональный элемент $a_{ij}^{(k)}$ матрицы $A^{(k)}$ $|a_{ij}^{(k)}| = \max_{l<m} |a_{lm}^{(k)}|$.

2) Ставится задача найти такую ортогональную матрицу $U^{(k)}$, чтобы в результате преобразования подобия $A^{(k+1)} = U^{(k)T} \cdot A^{(k)} \cdot U^{(K)}$ произошло обнуление элемента $a_{ij}^{(k+1)}$ матрицы $A^{(k+1)}$. В качестве ортогональной матрицы выбирается матрица вращения, имеющая следующий вид:

$$U^k = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccc} & & & i & & & j & & \\ \end{array} \\ \begin{pmatrix} 1 & & & \vdots & & & \vdots & & \\ & \ddots & & \vdots & & & \vdots & 0 & \\ & & 1 & \vdots & & & \vdots & & \\ \cdots\cdots & & & \cos\varphi^{(k)} & \cdots & & -\sin\varphi^{(k)} & \cdots\cdots & \\ & & & \vdots & 1 & & \vdots & & \\ & & & \vdots & & \ddots & \vdots & & \\ & & & \vdots & & & 1 & \vdots & \\ \cdots\cdots & & & \sin\varphi^{(k)} & \cdots & & \cos\varphi^{(k)} & \cdots\cdots & \\ & & & \vdots & & & \vdots & 1 & \\ & 0 & & \vdots & & & \vdots & & \ddots \\ & & & \vdots & & & \vdots & & 1 \end{pmatrix} \end{array} \begin{array}{l} \\ \\ \\ i \\ \\ \\ \\ j \\ \\ \\ \end{array},$$

В матрице вращения на пересечении $i-$й строки и $j-$го столбца находится элемент $u_{ij}^{(k)} = -\sin\varphi^{(k)}$, где $\varphi^{(k)}$ — угол вращения, подлежащий определению. Симметрично относительно главной диагонали ($j-$я строка, $i-$й столбец) расположен элемент $u_{ji}^{(k)} = \sin\varphi^{(k)}$.

Диагональные элементы $u_{ii}^{(k)}$ и $u_{jj}^{(k)}$ равны соответственно $u_{ii}^{(k)} = u_{jj}^{(k)} = \cos\varphi^{(k)}$; другие диагональные элементы $u_{mm}^{(k)} = 1, \quad m = \overline{1,n}, \quad m \neq i \neq j$; остальные элементы в матрице вращения $U^{(k)}$ равны нулю.

Угол вращения $\varphi^{(k)}$ определяется из условия $a_{ij}^{(k+1)} = 0$:

$$\varphi^{(k)} = \frac{1}{2} \cdot arctg \frac{2 \cdot a_{ij}^{(k)}}{a_{ii}^{(k)} - a_{jj}^{(k)}},$$

причем если $a_{ii}^{(k)} = a_{jj}^{(k)}$, то $\varphi^{(k)} = \frac{\pi}{4}$.

3) Строится матрица $A^{(k+1)}$

$$A^{(k+1)} = U^{(k)T} \cdot A^{(k)} \cdot U^{(k)},$$

в которой элемент $a_{ij}^{(k+1)} \approx 0$.

В качестве критерия окончания итерационного процесса используется условие малости суммы квадратов внедиагональных элементов:

$$t(A^{(k+1)}) = (\sum_{l,m;l<m} (a_{lm}^{(k+1)})^2)^{\frac{1}{2}}$$

Если $t(A^{(k+1)}) > \varepsilon$, то итерационный процесс

$$A^{(k+1)} = U^{(k)T} \cdot A^{(k)} \cdot U^{(k)} = U^{(k)T} \cdot U^{(k-1)T} \cdot \ldots \cdot U^{(0)T} \cdot A^{(0)} \cdot U^{(0)} \cdot U^{(1)} \cdot \ldots \cdot U^{(k)}$$

продолжается.

Если $t(A^{(k+1)}) < \varepsilon$, то итерационный процесс останавливается, и в качестве искомых собственных значений принимаются $\lambda_1 \approx a_{11}^{(k+1)}, \lambda_2 \approx a_{22}^{(k+1)}, \ldots, \lambda_n \approx a_{nn}^{(k+1)}$.
Координатными столбцами собственных векторов матрицы A в единичном базисе будут столбцы матрицы $U = U^{(0)} \cdot U^{(1)} \cdot \ldots \cdot U^{(k)}$, т.е.

$$(x^1)^T = (u_{11}u_{21}\ldots u_{n1}), (x^2)^T = (u_{12}u_{22}\ldots u_{n2}), \ldots, (x^n)^T = (u_{1n}u_{2n}\ldots u_{nn}),$$

причем эти собственные векторы будут ортогональны между собой, т.е.
$(x^l, x^m) \approx 0, \quad l \neq m.$

2 Протокол

Входные данные я храню в файле *data4*: первая строка — размерность матрицы, на следующих строках — матрица и заданная точность.
Выходные данные я записываю в файл *res4*.
Скриншот консоли:
$\varepsilon = 0.1$.

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ g++ 1.4.cpp -o 1.4
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat data4
3
7 3 -1
3 -7 -8
-1 -8 -2
0.1
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ ./1.4 < data4 > res4
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat res4
Введите размерность матрицы: Введите матрицу A:
Введите точность:
Число итераций: 4
Собственные значения:
lyambda_1 = 8.57533
lyambda_2 = -13.0509
lyambda_3 = 2.47558

Собственные векторы:
x_1 = 0.868494 0.349851 -0.351174
x_2 = -0.0643213 0.781987 0.619967
x_3 = 0.491509 -0.51585 0.701654
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$
```

Алгоритм нашел собственные значения и собственные векторы за 4 итерации с точностью $\varepsilon = 0.1$.

$\lambda_1 = 8.57533, \quad \lambda_2 = -13.0509, \quad \lambda_3 = 2.47558;$

$$x^1 = \begin{pmatrix} 0.868494 \\ 0.349851 \\ -0.351174 \end{pmatrix}, \quad x^2 = \begin{pmatrix} -0.0643213 \\ 0.781987 \\ 0.619967 \end{pmatrix}, \quad x^3 = \begin{pmatrix} 0.491509 \\ -0.51585 \\ 0.701654 \end{pmatrix}.$$

Попробуем уменьшить точность в 10 раз и посмотрим, как изменится число итераций.
Скриншот консоли:
$\varepsilon = 0.01$.

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ g++ 1.4.cpp -o 1.4
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat data4
3
7 3 -1
3 -7 -8
-1 -8 -2
0.01
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ ./1.4 < data4 > res4
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat res4
Введите размерность матрицы: Введите матрицу A:
Введите точность:
Число итераций: 5
Собственные значения:
lyambda_1 = 8.57605
lyambda_2 = -13.0509
lyambda_3 = 2.47486

Собственные векторы:
x_1 = 0.868494 0.349851 -0.351174
x_2 = -0.0925755 0.81045 0.578447
x_3 = 0.486979 -0.469867 0.736258
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$
```

Алгоритм нашел собственные значения и собственные векторы за 5 итераций, с точностью $\varepsilon = 0.01$.

$\lambda_1 = 8.57605, \quad \lambda_2 = -13.0509, \quad \lambda_3 = 2.47486;$

$$x^1 = \begin{pmatrix} 0.868494 \\ 0.349851 \\ -0.351174 \end{pmatrix}, \quad x^2 = \begin{pmatrix} -0.0925755 \\ 0.81045 \\ 0.578447 \end{pmatrix}, \quad x^3 = \begin{pmatrix} 0.486979 \\ -0.469867 \\ 0.736258 \end{pmatrix}.$$

Рассмотрим еще одно значение $\varepsilon = 0.0001$.

Скриншот консоли:

$\varepsilon = 0.0001$.

```
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ g++ 1.4.cpp -o 1.4
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat data4
3
7 3 -1
3 -7 -8
-1 -8 -2
0.0001
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ ./1.4 < data4 > res4
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$ cat res4
Введите размерность матрицы: Введите матрицу A:
Введите точность:
Число итераций: 6
Собственные значения:
lyambda_1 = 8.57605
lyambda_2 = -13.0509
lyambda_3 = 2.47486

Собственные векторы:
x_1 = 0.873722 0.344737 -0.343172
x_2 = -0.0925755 0.81045 0.578447
x_3 = 0.477536 -0.473632 0.740022
(base) vlasochka@vlasochka-VPCSB11FX:~/Документы/ЧМ$
```

Алгоритм нашел собственные значения и собственные векторы за 6 итераций, с точностью $\varepsilon = 0.0001$.

$\lambda_1 = 8.57605, \quad \lambda_2 = -13.0509, \quad \lambda_3 = 2.47486;$

$$x^1 = \begin{pmatrix} 0.873722 \\ 0.344737 \\ -0.343172 \end{pmatrix}, \quad x^2 = \begin{pmatrix} -0.0925755 \\ 0.81045 \\ 0.578447 \end{pmatrix}, \quad x^3 = \begin{pmatrix} 0.477536 \\ -0.473632 \\ 0.740022 \end{pmatrix}.$$

Можно заметить, что собственные значения, вычисленные с точностью $\varepsilon = 0.01$ и точностью $\varepsilon = 0.0001$, совпадают. Однако, некоторые значения собственных векторов продолжают меняться. Видно, что с уменьшением значения оценки погрешности, сходимость замедляется — когда мы уменьшили ε на один знак после запятой, число итераций выросло на единицу. Далее, уменьшив ε уже на два знака после запятой, мы получили число итераций, увеличенное так же на единицу, т.е. при оценках $\varepsilon = 0.001$ и $\varepsilon = 0.0001$ число итераций будет одинаково.

3 Исходный код

Листинг 1: Метод вращений Якоби

```
#include <iostream>
#include <vector>
#include <cmath>

typedef struct
{
    int i;
    int j;
    double value;
} Max;

void InitE(std::vector<std::vector<double>>& U, int n)
{
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            if(i == j)
                U[i][j] = 1;
            else
                U[i][j] = 0;
}

void MultMatrix(std::vector<std::vector<double>>& F, std::vector<std::vector<double>>& S)
{
    int n = F.size();
    int m = S[0].size();
    double a = 0;
    std::vector<double> H(n);

    for(int i = 0; i < m; i++)
    {
        for(int j = 0; j < m; j++)
        {
            for(int k = 0; k < n; k++)
            {
                a += F[j][k]*S[k][i];
            }
            H[j] = a;
            a = 0;
        }
```

```
40          for(int j = 0; j < n; j++)
41              S[j][i] = H[j];
42      }
43  }
44
45  void MultMatrix(std::vector<std::vector<double>>& F, std::vector<std::vector<double>>& S,
        std::vector<std::vector<double>>& Res)
46  {
47      int n = F.size();
48      double a = 0;
49      for(int i = 0; i < n; i++)
50      {
51          for(int j = 0; j < n; j++)
52          {
53              for(int k = 0; k < n; k++)
54              {
55                  a += F[j][k]*S[k][i];
56              }
57              Res[j][i] = a;
58              a = 0;
59          }
60      }
61  }
62
63  void Transporant(std::vector<std::vector<double>>& Matrix)
64  {
65      double h = 0;
66      int n = Matrix.size();
67      for(int i = 0; i < n; i++)
68          for(int j = i + 1; j < n; j++)
69          {
70              h = Matrix[i][j];
71              Matrix[i][j] = Matrix[j][i];
72              Matrix[j][i] = h;
73          }
74  }
75
76  int JacobiRotations(std::vector<std::vector<double>>& A, std::vector<std::vector<double>>&
        U, double eps)
77  {
78      int n = A.size();
79      double fi = 0;
80      double a = 0;
81      double kr = eps + 1;
```

```
82      int it_c = 0;
83      Max m;
84      std::vector<std::vector<double>> Res(n, std::vector<double>(n, 0));
85      std::vector<std::vector<double>> SV(n, std::vector<double>(n, 0));
86      InitE(SV, n);
87      m.value = 0;
88
89      while(kr > eps)
90      {
91          MultMatrix(SV, U);
92          SV = U;
93          InitE(U, n);
94          m.value = 0;
95          kr = 0;
96          it_c += 1;
97          for(int i = 0; i < n; i++)
98              for(int j = i + 1; j < n; j++)
99                  if( fabs(A[i][j]) > m.value)
100                 {
101                     m.value = fabs(A[i][j]);
102                     m.i = i;
103                     m.j = j;
104                 }
105         fi = atan(2*A[m.i][m.j]/(A[m.i][m.i] - A[m.j][m.j]))/2;
106
107         U[m.i][m.i] = cos(fi);
108         U[m.j][m.j] = cos(fi);
109         U[m.i][m.j] = -sin(fi);
110         U[m.j][m.i] = sin(fi);
111         Transporant(U);
112         MultMatrix(U, A, Res);
113         Transporant(U);
114         MultMatrix(Res, U, A);
115
116         for(int i = 0; i < n; i++)
117             for(int j = i + 1; j < n; j++)
118                 kr += pow(A[i][j], 2);
119         kr = pow(kr, 0.5);
120     }
121     U = SV;
122     return it_c;
123 }
124
125 int main()
```

```cpp
{
    int n;
    double element, eps;
    std::cin >> n;

    std::vector<std::vector<double>> A(n, std::vector<double>(n));
    std::vector<std::vector<double>> U(n, std::vector<double>(n, 0));

    for( int i = 0; i < n; i++)
        for( int j = 0; j < n; j++)
        {
            std::cin >> A[i][j];
            if(i == j)
                U[i][j] = 1;
        }
    std::cin >> eps;
    std::cout << " : " << JacobiRotations(A, U, eps) << std::endl;
    std::cout << " :" << std::endl;
    for(int i = 0; i < n; i++)
        std::cout << "lyambda_" << i + 1 << " = " << A[i][i] << std::endl;
    std::cout << std::endl;
    std::cout << "Eigenvectors of matrix A:" << std::endl;
    for(int i = 0; i < n; i++)
    {
        std::cout << "x_" << i + 1 << " = ";
        for(int j = 0; j < n; j++)
            std::cout << U[j][i] << " ";
        std::cout << std::endl;
    }
    return 0;
}
```

4 Выводы

Выполнив четвертое задание первой лабораторной работы, я познакомилась с методом вращений Якоби, который решает задачу собственных значений и собственных векторов симметрических матриц. Метод базируется на поиске матрицы преобразования, каждый столбец которой является собственным вектором, путем последовательных приближений. Эта матрица переводит исходную матрицу в диагональную, с собственными значениями на главной диагонали. Поиск происходит так, что на каждой итерации метод постепенно «обнуляет» элементы матрицы, расположенные не на главной диагонали, приближая ее к подобному виду. В то же время, чем меньше значение оценки погрешности, тем больше количество итераций и точнее найденная матрица преобразования.