

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теории функций и стохастического анализа

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ (БАЗОВОЙ) ПРАКТИКЕ

студента 4 курса 451 группы

направления 38.03.05 — Бизнес-информатика

механико-математического факультета

Чайковского Петра Ильича

Место прохождения: завод "Тантал"

Сроки прохождения: с 29.06.2019 г. по 26.07.2019 г.

Оценка:

Руководитель практики от СГУ

доцент, к. ф.-м. н.

Н. Ю. Агафонова

Руководитель практики от организации

ведущий программист

Д. Э. Кнутов

Саратов 2019

Тема практики: «Правила оформления курсовых и дипломных работ»

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 Постановка задачи. Описание метода её решения.....	5
2 Вычисление сложности проблемы.	8
3 Программа решения на языке C++.	10
ЗАКЛЮЧЕНИЕ	16
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	17

ВВЕДЕНИЕ

Работа с матрицами и системами линейных уравнений представляет особенно большие проблемы, если пользоваться неэффективными методами решения. Поиск решений систем линейных уравнений составляет обязательную часть решения некоторых прикладных задач, поэтому на практике возникает необходимость составления эффективных методов решения систем линейных уравнений, поиска определителей, нахождения собственного значения и собственного вектора матрицы. Одним из таких эффективных методов решения является НВП-разложение.

1 Постановка задачи. Описание метода её решения.

Для того, чтобы более корректно поставить задачу вычисления определителя невырожденной матрицы и описать метод её решения, необходимо ввести несколько обязательных определений.

Определение. НВ-разложением (НВ - по первым буквам слов "нижняя" и "верхняя") $(m \times n)$ - матрицы A , $m \leq n$, называется представление её в виде $A = LU$, где L - нормированная нижняя треугольная $(m \times n)$ - матрица, а U - верхняя треугольная $(m \times n)$ - матрица.

Уравнение $A\mathbf{x} = \mathbf{b}$, где A - это $(n \times n)$ - матрица, \mathbf{x} - n -мерный вектор-столбец неизвестных, а \mathbf{b} - n -мерный вектор-столбец, можно решить, сначала НВ-разложив A в произведение LU в предположении, что это возможно. Затем представим $A\mathbf{x} = \mathbf{b}$ в виде $LU\mathbf{x} = \mathbf{b}$. Чтобы получить решение \mathbf{x} , решим сначала $L\mathbf{y} = \mathbf{b}$ относительно \mathbf{y} , а затем $U\mathbf{x} = \mathbf{y}$ относительно \mathbf{x} .

Трудность применения метода заключается в том, что у матрицы A может не быть НВ-разложения, даже если она невырожденна. Однако если матрица A невырожденна, то найдётся такая матрица перестановки P , что AP^{-1} имеет НВ-разложение.

Изложим алгоритм, который по любой невырожденной матрице A находит такие матрицы L, U и P , что $A = LUP$. Матрицы L, U и P образуют НВП-разложение (НВП - по первым буквам слов "нижняя" "верхняя" и "перестановка") матрицы A :

Вход. Невырожденная $(n \times n)$ - матрица M , n - степень числа 2.

Выход. Матрицы L, U и P , для которых $M = LUP$, причём L - нормированная нижняя треугольная матрица, U - верхняя треугольная и P - матрица перестановки.

Метод. Вызываем процедуру **МНОЖИТЕЛЬ**(M, n, n), где **МНОЖИТЕЛЬ** - рекурсивная процедура, описанная ниже.

Каждый рекурсивный вызов процедуры **МНОЖИТЕЛЬ** происходит на $(m \times p)$ - подматрице A $(n \times n)$ - матрицы M . При каждом вызове m есть степень числа 2 и $m \leq p \leq n$. Выходом этой процедуры являются три матрицы L, U и P .

Опишем псевдокод процедуры **МНОЖИТЕЛЬ**:

```
1 procedure МНОЖИТЕЛЬ( $A, m, p$ ):  
2 if  $m = 1$  then  
3   begin  
4     пусть  $L = [1]$  (т.е.  $L$  - нормированная  $(1 \times 1)$  - матрица);  
5     найти, если можно, столбец с матрицы  $A$  с ненулевым  
6       элементом, и пусть  $P$  будет  $(p \times p)$  - матрицей,  
7       переставляющей столбцы 1 и  $s$ ;  
8     пусть  $U = AP$ ;  
9     return  $(L, U, P)$   
10  end  
11 else  
12  begin  
13    разбить  $A$  на  $((m/2) \times p)$  - матрицы  $B$  и  $C$   
14    вызвать МНОЖИТЕЛЬ( $B, m/2, p$ ), чтобы получить  $L_1, U_1, P_1$ ;  
15    вычислить  $D = CP_1^{-1}$ ;  
16    вычислить  $G = D - FE^{-1}U_1$ ;  
17    пусть  $G'$  - самые правые  $p - m/2$  столбцов матрицы  $G$ ;  
18    вызвать МНОЖИТЕЛЬ( $G', m/2, p - m/2$ ) и получить  $L_2, U_2$  и  $P_2$ ;  
19    пусть  $P_3$  будет  $(p \times p)$  - матрицей перестановки, у которой в  
20      левом верхнем углу стоит  $I_{m/2}$ , а в правом нижнем  $P_2$ ;  
21    вычислить  $H = U_1P_3^{-1}$ ;  
22    пусть  $L$  - это  $(m \times m)$  - матрица, состоящая из  $L_1, O_{m/2}, FE^{-1}$  и  $L_2$ ;  
23    пусть  $U$  - это  $(m \times p)$  - матрица, у которой в верхней части  
24      стоит  $H$ , а в нижней  $O_{m/2}$  и  $U_2$ ;  
25    пусть  $P$  - произведение  $P_3P_1$ ;  
26    return  $(L, U, P)$   
27  end
```

Листинг 1: Описание процедуры **МНОЖИТЕЛЬ**.

Разобравшись с формулировкой необходимых определений и демонстрации псевдокода обязательной процедуры, перейдём к рассмотрению одной из основных теорем, следствие из которой даёт оценку сложности НВП-разложения любой невырожденной $(n \times n)$ - матрицы.

Как впоследствии будет видно, задача вычисления определителя невырожденной матрицы может быть сведена к задаче умножения двух матриц.

Теорема 1. Пусть для каждого n можно умножить две $(n \times n)$ - матрицы за такое время $M(n)$, что при некотором $\varepsilon > 0$ неравенство $M(2m) \geq$

$2^{2+\varepsilon}M(m)$ выполняется для всех m (если проще, то здесь требуется, чтобы значение $M(n)$ было заключено между $n^{2+\varepsilon}$ и n^3 . Может оказаться, что $M(n) = kn^2 \log n$ для некоторой постоянной k ; тогда условие теоремы не удовлетворяется.). Тогда найдётся такая постоянная k , что алгоритм нахождения матриц L, U и P , таких что $A = LUP$, тратит не более $kM(n)$ времени для любой невырожденной матрицы.

Мы не будем приводить доказательство, но упомянем очень важное следствие, развитие идеи которого приведёт нас к оценке временной сложности исходной задачи.

Следствие. НВП-разложение любой невырожденной $(n \times n)$ - матрицы можно найти за $O(n^{2,81})$ шагов.

2 Вычисление сложности проблемы.

В этом разделе мы продемонстрируем тот факт, что задача нахождения определителя невырожденной $(n \times n)$ - матрицы может быть сведена к задаче умножения двух матриц. Это означает, что любое улучшение асимптотической сложности умножения матриц приведёт к улучшению асимптотической сложности исходной задачи.

Теорема 2. Пусть $M(n)$ - время, требуемое для умножения двух $(n \times n)$ - матриц. Если $8M(m) \geq M(2m) \geq 4M(m)$ для всех m , то найдётся такая постоянная c , что обращение любой невырожденной верхней (нижней) треугольной $(n \times n)$ - матрицы A можно вычислить за время $cM(n)$.

Теорема 3. Пусть $\varepsilon > 0$ и $a \geq 1$. Пусть $M(n)$ - время, требуемое для умножения двух матриц, и $M(2m) \geq 2^{2+\varepsilon} M(m)$ для некоторого $\varepsilon > 0$. Тогда матрицу, обратную к данной невырожденной матрице, можно найти за время $O(M(n))$.

Доказательство. Пусть A - невырожденная $(n \times n)$ - матрица. В силу теоремы 1 можно найти НВП-разложение $A = LUP$ за время $O(M(n))$. Тогда $A^{-1} = P^{-1}U^{-1}L^{-1}$. Матрицу P^{-1} можно вычислить за $O(n)$ шагов. Матрицы U^{-1} и L^{-1} существуют, и их можно аналогично вычислить за $O(M(n))$ шагов в силу теоремы 2. Аналогично за $O(M(n))$ шагов можно вычислить произведение $P^{-1}U^{-1}L^{-1}$. \square

И, наконец, теорема, дающая ответ на вопрос о временной сложности поставленной нами задачи:

Теорема 4. Если функция $M(n)$ та же, что и в теореме 3, и A есть $(n \times n)$ - матрица, то $\det(A)$ можно вычислить за $O(M(n))$ шагов.

Доказательство. Применим вышеупомянутый алгоритм, чтобы найти НВП-разложение матрицы A . Если он не срабатывает из-за того, что в строке 5 не удалось найти ненулевой столбец или в строке 16 не существует E^{-1} , то матрица A вырожденна, и $\det(A) = 0$. В противном случае пусть $A = LUP$. Тогда $\det(A) = \det(L) \det(U) \det(P)$. Найдём $\det(L)$ и $\det(U)$, вычислив произведения их диагональных элементов. Так как L - нормированная нижняя треугольная матрица, то $\det(L) = 1$. Так как U - верхняя треугольная, то можно вычислить $\det(U)$ за $O(n)$ шагов. Поскольку P - матрица перестановки, то $\det(P) = \pm 1$ в зависимости от того, представляет P четную или нечётную

перестановку. Вопрос о чётности или нечётности перестановки можно выяснить, построив её из $(1, 2, \dots, n)$ с помощью транспозиций. Потребуется не более $n - 1$ транспозиций, и их число можно сосчитать во время выполнения. \square

Таким образом, при помощи следующего следствия, мы получаем ответ на вопрос о временной сложности алгоритма для поставленной задачи вычисления определителя невырожденной матрицы:

Следствие. Определитель $(n \times n)$ - матрицы можно вычислить за $O(n^{2,81})$ шагов.

3 Программа решения на языке C++.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  #include <iostream>
5  #include <algorithm>
6  #include <vector>
7
8  using namespace std;
9
10 void inputMatrix (double ** A,  int n)
11 {
12     int i, j;
13
14     cout << "Введите вашу матрицу:\n";
15
16     for (i = 1; i <= n; ++i)
17         for (j = 1; j <= n; ++j)
18             cin >> A[i][j];
19 }
20
21 void LUP_Decomposition (double ** A, double ** L,
22                         double ** U, int * P, int n)
23 {
24     int i, j, k, difK;
25     double temp, difTemp;
26
27     for (i = 1; i <= n; ++i)
28         P[i] = i;
29
30     for (k = 1; k <= n; ++k)
31     {
32         int p = 0;
33         for (i = k; i <= n; i++)
34         {
35             if (fabs (A[i][k]) > p)
36             {
37                 p = fabs (A[i][k]);
38                 difK = i;
```

```

39     }
40 }
41
42 if (p == 0)
43     cout << "Единичная матрица\n";
44 else swap (P[k], P[difK]);
45
46 for (i = 1; i <= n; i++)
47     swap (A[k][i], A[difK][i]);
48
49     for (i = k + 1; i <= n; ++i)
50     {
51         temp = A[i][k] / A[k][k];
52         A[i][k] = round (temp * 100) / 100;
53
54         for (j = k + 1; j <= n; ++j)
55         {
56             temp = A[i][k] * A[k][j];
57             difTemp = round (temp * 100) / 100;
58             A[i][j] = A[i][j] - difTemp;
59         }
60     }
61 }
62
63 for (i = 1; i <= n; ++i)
64     for (j = 1; j <= n; ++j)
65         if (i == j)
66             L[i][j] = 1;
67         else
68             if (i < j)
69                 L[i][j] = 0;
70             else L[i][j] = A[i][j];
71
72 for (i = 1; i <= n; ++i)
73     for (j = 1; j <= n; ++j)
74         if (i <= j)
75             U[i][j] = A[i][j];
76         else U[i][j] = 0;
77
78     for (i = 1; i <= n; ++i)

```

```

79     P[i] -= 1;
80 }
81
82 void print (int * x, int n)
83 {
84     int i, j;
85
86     for (i = 1; i <= n; ++i)
87         cout << x[i] << "\t";
88 }
89
90 void printL (double ** L, int n)
91 {
92     int i, j;
93
94     for (i = 1; i <= n; i++)
95     {
96         for (j = 1; j <= n; ++j)
97             cout << L[i][j] << "\t";
98
99         cout << "\n";
100     }
101 }
102
103 vector<vector<double>> getMinor (int columnIdx,
104                                vector<vector<double>> matrix)
105 {
106     int i, j;
107     vector<vector<double>> minor;
108
109     for (i = 1; i < matrix.size(); ++i)
110     {
111         vector<double> row;
112
113         for (j = 0; j < matrix.size(); ++j)
114             if (j != columnIdx)
115                 row.push_back (matrix[i][j]);
116
117         minor.push_back (row);
118     }

```

```

119
120     return (minor);
121 }
122
123 int computeDet (vector<vector<double>> matrix)
124 {
125     if (matrix.size() == 1)
126         return (matrix[0][0]);
127
128     int det = 0, multiplier = 1, i;
129
130     for (i = 0; i < matrix.size(); ++i)
131     {
132         int elt = matrix[0][i];
133
134         if (elt != 0)
135             det += multiplier * elt * computeDet (getMinor (i, matrix));
136
137         multiplier *= -1;
138     }
139
140     return (det);
141 }
142
143 void printMatrix (vector<vector<double>> matrix)
144 {
145     int i, j;
146
147     for (i = 0; i < matrix.size(); ++i)
148     {
149         for (j = 0; j < matrix.size(); ++j)
150             cout << matrix[i][j] << " ";
151
152         cout << "\n";
153     }
154 }
155
156 void convertArrayToVector (double ** U,
157                             vector<vector<double>>& U_vector,
158                             int dimension)

```

```

159 {
160     int i, j;
161
162     for (i = 1; i <= dimension; ++i)
163     {
164         vector<double> row;
165
166         for (j = 1; j <= dimension; ++j)
167             row.push_back (U[i][j]);
168
169         U_vector.push_back (row);
170     }
171 }
172
173 vector<vector<double>> formIdentity (int * P, int dimension)
174 {
175     int i, j;
176     vector<vector<double>> permutatedIdentity;
177
178     for (i = 0; i < dimension; ++i)
179     {
180         vector<double> row (dimension, 0);
181         permutatedIdentity.push_back (row);
182     }
183
184     for (i = 1; i <= dimension; ++i)
185         permutatedIdentity[i - 1][P[i]] = 1;
186
187     return (permutatedIdentity);
188 }
189
190 int main()
191 {
192     int n, i = 0;
193     double ** L = NULL;
194     double ** U = NULL;
195     int * P = NULL;
196     double ** A = NULL;
197
198     cout << "Введите размерность матрицы:\n";

```

```

199     cin >> n;
200
201     L = new double * [n + 1];
202     for (i = 1; i <= n; ++i)
203         L[i] = new double[n + 1];
204
205     U = new double * [n + 1];
206     for (i = 1; i <= n; ++i)
207         U[i] = new double[n + 1];
208
209     A = new double * [n + 1];
210     for (i = 1; i <= n; ++i)
211         A[i] = new double[n + 1];
212
213     P = new int[n];
214
215     inputMatrix (A, n);
216     LUP_Decomposition (A, L, U, P, n);
217     cout << "L:\n";
218     printL (L, n);
219     cout << "U:\n";
220     printL (U, n);
221     cout << "P:\n";
222     print (P, n);
223
224     vector<vector<double>> U_vector;
225
226     cout << "\nОпределитель заданной вами матрицы равен: ";
227
228     convertArrayToVector (U, U_vector, n);
229     cout << computeDet (U_vector)
230         * computeDet (formIdentity (P, n)) << "\n";
231
232     return (EXIT_SUCCESS);
233 }

```

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы было показано, что задача нахождения определителя матрицы может быть сведена к поиску НВП-разложения заданной матрицы, и дальнейшему поиску произведения определителей матриц из этого НВП-разложения. Такой метод является более быстрым, нежели иные способы нахождения значения определителя (метод Лапласа или другие способы). Применение НВП-разложения не ограничивается задачей нахождения определителя, оно также может быть использовано, например, для решения систем линейных уравнений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. А. Ахо, Дж.Хопкрофт, Дж.Ульман. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979, 536 стр.