МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

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

Кафедра теоретических основ компьютерной безопасности и криптографии

ВЫЧИСЛЕНИЕ ОПРЕДЕЛИТЕЛЯ С ПОМОЩЬЮ НВП-РАЗЛОЖЕНИЯ

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«СЛОЖНОСТЬ ВЫЧИСЛЕНИЙ»

студента 3 курса 331 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Бородина Артёма Горовича

Преподаватель

доцент, к.ф.-м.н. ___________ А. Н. Гамова

подпись, дата

Саратов 2022

СОДЕРЖАНИЕ

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 $\mathrm{A}\mathbf{x} = \mathrm{b}$, где A - это $(n \times n)$ - матрица, 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 $\mathrm{A}\mathbf{x} = \mathbf{b}$ 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**:

```
procedure МНОЖИТЕЛЬ($A, m, p$):
if $m = 1$ then
  begin
    пусть $L = [1]$ (т.е. $L$ - нормированная $(1 \times 1)$ - матрица);
    найти, если можно, столбец c матрицы $A$ с ненулевым
      элементом, и пусть $P$ будет $(p \times p)$ - матрицей,
      переставляющей столбцы 1 и $c$;
    пусть $U = AP$;
    return $(L, U, P)$
  end
else
  begin
    разбить $A$ на $((m/2) \times p)$ - матрицы $B$ и $C$
    вызвать МНОЖИТЕЛЬ($B, m/2, p$), чтобы получить $L_1, U_1, P_1$;
    вычислить $D = CP_1^{-1}$;
    вычислить $G = D - FE^{-1}U_1$;
    пусть $G'$ - самые правые $p - m/2$ столбцов матрицы $G$;
    вызвать МНОЖИТЕЛЬ($G', m/2, p - m/2$) и получить $L_2, U_2$ и $P_2$;
    пусть $P_3$ будет $(p \times p)$ - матрицей перестановки, у которой в
      левом верхнем углу стоит $I_{m/2}$, а в правом нижнем $P_2$;
    вычислить $H = U_1 P_3^{-1}$;
    пусть $L$ - это $(m \times m)$ - матрица, состоящая из $L_1, O_{m/2}, FE^{-1}$ и $L_2$;
    пусть $U$ - это $(m \times p)$ - матрица, у которой в верхней части
      стоит H, а в нижней $O_{m/2}$ и $U_2$;
    пусть $P$ - произведение $P_3 P_1$;
    return $(L, U, P)$
  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 трегольная 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 $(1, 2, \ldots, n)$ 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++.

Программно реализуем алгоритм построения НВП-разложения матрицы и вычисления её определителя. Будем рассматривать модифицированную версию алгоритма, обрабатывающие также вырожденные матрицы.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

void inputMatrix (double ** A,  int n)
{
  int i, j;

  cout << "Введите вашу матрицу:\n";

  for (i = 1; i <= n; ++i)
  for (j = 1; j <= n; ++j)
  cin >> A[i][j];
}

void LUP_Decomposition (double ** A, double ** L,
                        double ** U, int * P, int n)
{
  int i, j, k, difK;
  double temp, difTemp;

  for (i = 1; i <= n; ++i)
  P[i] = i;

  for (k = 1; k <= n; ++k)
  {
    int p = 0;
    for (i = k; i <= n; i++)
    {
      if (fabs (A[i][k]) > p)
```

```
      {
        p = fabs (A[i][k]);
        difK = i;
      }
    }

  if (p == 0)
  cout << "Единичная матрица\n";
  else swap (P[k], P[difK]);

  for (i = 1; i <= n; i++)
  swap (A[k][i], A[difK][i]);

    for (i = k + 1; i <= n; ++i)
    {
      temp = A[i][k] / A[k][k];
      A[i][k] = round (temp * 100) / 100;

      for (j = k + 1; j <= n; ++j)
      {
        temp = A[i][k] * A[k][j];
        difTemp = round (temp * 100) / 100;
        A[i][j] = A[i][j] - difTemp;
      }
    }
  }

  for (i = 1; i <= n; ++i)
    for (j = 1; j <= n; ++j)
      if (i == j)
        L[i][j] = 1;
      else
        if (i < j)
          L[i][j] = 0;
        else L[i][j] = A[i][j];

  for (i = 1; i <= n; ++i)
    for (j = 1; j <= n; ++j)
      if (i <= j)
        U[i][j] = A[i][j];
```

```
      else U[i][j] = 0;

    for (i = 1; i <= n; ++i)
      P[i] -= 1;
}

void print (int * x, int n)
{
  int i, j;

  for (i = 1; i <= n; ++i)
    cout << x[i] << "\t";
}

void printL (double ** L, int n)
{
  int i, j;

  for (i = 1; i <= n; i++)
  {
    for (j = 1; j <= n; ++j)
      cout << L[i][j] << "\t";

    cout << "\n";
  }
}

vector<vector<double>> getMinor (int columnIdx,
                                 vector<vector<double>> matrix)
{
  int i, j;
  vector<vector<double>> minor;

  for (i = 1; i < matrix.size(); ++i)
  {
    vector<double> row;

    for (j = 0; j < matrix.size(); ++j)
      if (j != columnIdx)
        row.push_back (matrix[i][j]);
```

```

    minor.push_back (row);
  }

  return (minor);
}

int computeDet (vector<vector<double>> matrix)
{
  if (matrix.size() == 1)
    return (matrix[0][0]);

  int det = 0, multiplier = 1, i;

  for (i = 0; i < matrix.size(); ++i)
  {
    int elt = matrix[0][i];

    if (elt != 0)
      det += multiplier * elt * computeDet (getMinor (i, matrix));

    multiplier *= -1;
  }

  return (det);
}

void printMatrix (vector<vector<double>> matrix)
{
  int i, j;

  for (i = 0; i < matrix.size(); ++i)
  {
    for (j = 0; j < matrix.size(); ++j)
      cout << matrix[i][j] << " ";

    cout << "\n";
  }
}

```

```
void convertArrayToVector (double ** U,
                           vector<vector<double>>& U_vector,
                           int dimension)
{
  int i, j;

  for (i = 1; i <= dimension; ++i)
  {
    vector<double> row;

    for (j = 1; j <= dimension; ++j)
    row.push_back (U[i][j]);

    U_vector.push_back (row);
  }
}

vector<vector<double>> formIdentity (int * P, int dimension)
{
  int i, j;
  vector<vector<double>> permutatedIdentity;

  for (i = 0; i < dimension; ++i)
  {
    vector<double> row (dimension, 0);
    permutatedIdentity.push_back (row);
  }

  for (i = 1; i <= dimension; ++i)
    permutatedIdentity[i - 1][P[i]] = 1;

  return (permutatedIdentity);
}

int main()
{
  int n, i = 0;
  double ** L = NULL;
  double ** U = NULL;
  int * P = NULL;
```

```
  double ** A = NULL;

  cout << "Введите размерность матрицы:\n";
  cin >> n;

  L = new double * [n + 1];
  for (i = 1; i <= n; ++i)
    L[i] = new double[n + 1];

  U = new double * [n + 1];
  for (i = 1; i <= n; ++i)
    U[i] = new double[n + 1];

  A = new double * [n + 1];
  for (i = 1; i <= n; ++i)
    A[i] = new double[n + 1];

  P = new int[n];

  inputMatrix (A, n);
  LUP_Decomposition (A, L, U, P, n);
  cout << "L:\n";
  printL (L, n);
  cout << "U:\n";
  printL (U, n);
  cout << "P:\n";
  print (P, n);

  vector<vector<double>> U_vector;

  cout << "\nОпределитель заданной вами матрицы равен: ";

  convertArrayToVector (U, U_vector, n);
  cout << computeDet (U_vector)
       * computeDet (formIdentity (P, n)) << "\n";

  return (EXIT_SUCCESS);
}
```

Протестируем программу на следующей вырожденной матрице.

```
Введите размерность матрицы:
3
Введите вашу матрицу:
1 2 3
4 5 6
7 8 9


L:
1       0       0
0.14    1       0
0.57    0.5     1
U:
7       8       9
0       0.88    1.74
0       0       1.11022e-16
P:
2       1       0
Определитель заданной вами матрицы равен: 0
```

Рисунок 1 – Результат работы программы на вырожденной матрице.

В результате работы программы получаем НВП-разложение матрицы. Поскольку $\det(L) = 1$, а $\det(P) = \pm 1$ - перестановочная единичная матрица, то необходимо вычислить только $\det(U)$ и определить знак $\det(P)$ (также вычислив определитель этой матрицы). Так как U - верхняя квадратичная матрица, то её определитель может быть вычислен за $O(n)$ шагов. Проверим правильность работы программы:

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = 1 \cdot \begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} + 2 \cdot \begin{vmatrix} 4 & 6 \\ 7 & 9 \end{vmatrix} + 3 \cdot \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} =$$

$$(45 - 48) + 2 \cdot (36 - 42) + 3 \cdot (32 - 35) = -3 + 12 - 9 = 0.$$

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. Н. Вирт. Алгоритмы и структуры данных. М.: Мир, 1989, 360 стр.
2. А. Ахо, Дж.Хопкрофт, Дж.Ульман. Построение и анализ вычислительных алгоритмов. М.: Мир, 1979, 536 стр.