Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

Студент	Никифорова Ирина Андреевна
Группа	РК6-71б
Тип задания	лабораторная работа
Тема лабораторной работы	Сочетания
Вариант	15 С

Студент _____ **Никифорова И. А.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Волосатова Т.М.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2019 г.

Оглавление

Задание на лабораторную работу

Определить все базисные решения СЛАУ:

$$\begin{cases} X_1 + X_2 + X_3 = 4 \\ -X_1 + X_2 + X_4 = 2 \\ 2X_1 + X_2 + X_5 = 6 \end{cases} \quad (1)$$

Полученные решения должны быть лексикографически упорядочены по индексам переменных

Идея решения

Для решения СЛАУ (1), составим соответствующее ей матричное уравнение:

1	1	1			X_1		4
-1	1		1		X_2	=	2
2	1			1	X_3		6
					X_4		
					X_5		

Приведенная система (1) является недоопределенной, так как количество переменных в ней превышает количество уравнений.[1] Также понятно, что система имеет решения, так как ранг матрицы равен рангу расширенной матрицы (по теореме Кронекера-Капелли).[2] Ниже представлена система в ступенчатом виде в доказательство наличия решений:

1	1	1			X_1		4
	2	1	1		X_2	=	6
		-3	1	2	X_3		2
					X_4		
					X_5		

Из нее видно, что максимальное число линейно независимых строк (ранг) как для матрицы коэффициентов, так и для расширенной матрицы $r = 3$. Отсюда получается, что в качестве базисных можно выбрать любые 3 переменные из 5 возможных (любые миноры размера 3 будут базисными), то есть количество вариантов выбора базисных переменных $C_5^3 = \frac{5!}{3!2!} = 10$.

Идея решения заключается в том, чтобы выбрать 3 базисные переменные из 5, далее занулить все свободные для данного случая переменные и найти базисное решение. Проделать это следует всеми возможными способами, причем выбранные переменные должны следовать в лексикографическом порядке.

В результате задача разбивается на две подзадачи:

1. выбор текущего сочетания в соответствии с лексикографическим порядком следования переменных
2. решение СЛАУ

Решение СЛАУ не относится непосредственно к теме курса, поэтому для решения этой задачи использовался ранее написанный для курса МиМАПР модуль. Выбор сочетания же осуществлялся согласно алгоритму, описанному далее.

Используемый комбинаторный алгоритм

Для дальнейшего перечисления сочетаний всем пяти переменным были сопоставлены натуральные числа от 1 до 5. Далее был использован следующий алгоритм перечисления сочетаний в лексикографическом порядке :

1. Составление минимального сочетания:

   $C_{min} = (C_1 = 1, C_2 = 2, C_3 = 3)$

2. Составление максимального сочетания для удобства дальнейших сравнений:

   $C_{min} = (C_1 = 3, C_2 = 4, C_3 = 5)$

3. Просмотр элементов текущего сочетания справа налево и поиск самого правого элемента, не достигшего своего максимального значения:

   $C_j = max\ \{i: C_i < (n - m + i)\}, j = m, \ldots, i$

4. Увеличение этого элемента на 1:

   $C_j = C_j + 1$

5. Присвоение каждому элементу справа от него наименьшего возможного допустимого значения, которое обязательно будет на 1 больше, чем у соседнего слева элемента:

   $C' = (C_1' = C_1, \ldots, C_{j-1}' = C_{j-1}, C_j' = C_j + 1, C_{j+1}' = C_j + 2, \ldots, C_m' = C_j + m - j + 1)$

6. Повторение пунктов 3 - 5, пока не получится максимально возможное сочетание. [4]

Листинг программы

1) Файл *main.cpp*:

```
#include "matrix.hpp"

#define M 3
#define N 5

using namespace std;

void print_combination(int C[M]) {
   for (int i = 0; i < M; i++) {
       cout << C[i];
   }
   cout << endl;
}

void array_to_Matrix(double* arr, int a, int b, Matrix<double>* m) {
    for (int i = 0; i < a; i++) {
        for (int j = 0; j < b; j++) {
        m->set(i, j, arr[i*b + j]);
        }
    }
}

void solve_slay(int basis[M]) {
    // задание начальных значений для СЛАУ
    Matrix<double> A(N - 2, N);
    double a[N][N] = {
            {1, 1, 1, 0, 0},
            {-1, 1, 0, 1, 0},
            {2, 1, 0, 0, 1},
    };
    array_to_Matrix(*a, N - 2, N, &A);

    Matrix<double> FR(N - 2, 1);
    double fr[N][1] = {
            {4},
            {2},
            {6},
    };
    array_to_Matrix(*fr, N - 2, 1, &FR);

    // поиск свободных переменных
    int non_basis[N-M];

    // переменные, которые присутствуют в базисе
    // будут отмечены по индексу i-1 значением true
    bool exists_in_basis[N];

    // false по всему массиву
    for (int i = 0; i < N; i++) {
        exists_in_basis[i] = false;
    }

    // отмечаем присутствие
```

```
    for (int i = 0; i < M; i++) {
        exists_in_basis[basis[i] - 1] = true;
    }

    // выделяем отсутствующие номера
    int j = 0;
    for (int i = 0; i < N; i++) {
        if (exists_in_basis[i] == false) {
            non_basis[j] = i;
            j++;
        }
    }

    // удаление столбцов свободных переменных,
    // чтобы упростить решение методом Гаусса
    // A.print(&FR, "Начальное уравнение:");

    // удаление всех столбцов, соответствующих
    // свободным переменным
    A.delete_cols(non_basis, N-M);

    // печать полученного уравнения
    // A.print(&FR, "После удаления столбцов свободных переменных:");

    // решение слау
    A.Gauss(&FR);

    // печать решения
    // A.print(&FR, "Ответ:");

    // печать базисных переменных (по ним упорядочено)
    for (int i = 0; i < M; i++) {
        cout << "X" << basis[i] << " = " << FR.get(i, 0) << ", ";
    }

    for (int i = 0; i < N - M; i++) {
        cout << "X" << non_basis[i] + 1 << ", ";
    }
    cout << "\b\b = 0\n";
}

int main() {
    // задание начальных значений сочетаний
    int Cmax[M], C[M];
    for (int i = 0; i < M; i++) {
       Cmax[i] = N + i - M + 1;
   }

   for (int i = 0; i < M; i++) {
       C[i] = i + 1;
   }

    // алгоритм перечисления + решение слау на каждой итерации
    // print_combination(C);
    solve_slay(C);

    while (C[0] != Cmax[0]) {
```

```
        // поиск справа-налево самого правого элемента,
         // не достигшего своего допустимого максимума
        int i_change = 0;
        for (int i = M - 1; (i >= 0) && (i_change == 0); i--) {
            if(C[i] < Cmax[i]) {
               i_change = i;
            }
        }

        // увеличение следующих за ним элементов
        C[i_change]++;
        for (int j = i_change + 1; j < M; j++) {
            C[j] = C[i_change] + j - i_change;
        }

         // печать нового сочетания и решение слау с ним
         // print_combination(C);
         solve_slay(C);
    }

    return 0;
}
```

2) Файл *matrix.cpp*:

```
#include "matrix.hpp"

// конструктор матрицы по строкам и столбцам
template <typename T>
Matrix<T>::Matrix(const int rows, const int cols) {
   _rows = rows;
   _cols = cols;
   _arr = new T*[rows];
   for (int i = 0; i < rows; i++) {
       _arr[i] = new T[cols];
   }
}

// возвращает количество строк в матрице
template <typename T>
const int Matrix<T>::get_rows() const {
   return _rows;
}

// возвращает количество столбцов в матрице
template <typename T>
const int Matrix<T>::get_cols() const {
   return _cols;
}

// возвращает элемент с индексами i и j
template <typename T>
const T Matrix<T>::get(int i, int j) const {
   if (!check_i_j(i, j)) {
       return 0;
   }

   return _arr[i][j];
```

```
}

// устанавливает элемент с индексами i и j в значение val
template <typename T>
void Matrix<T>::set(int i, int j, T val)  {
    if (!check_i_j(i, j)) {
        return;
    }

    _arr[i][j] = val;
}

// проверяет на корректность номера элементов
template <typename T>
const bool Matrix<T>::check_i_j(int i, int j) const {
    if (i >= _rows) {
        cout << "too high row:" << i << endl;
        return false;
    } else if (j >= _cols) {
        cout << "too high col:" << j << endl;
        return false;
    } else if (i < 0) {
        cout << "too low row:" << i << endl;
        return false;
    } else if (j < 0) {
        cout << "too low col:" << j << endl;
        return false;
    }

    return true;
}

// делит главную строку на диагональный элемент,
// чтобы там была единица
template <typename T>
void Matrix<T>::make_diag_one_Gauss(int i_from, Matrix<double>* fr) {
    T coeff = _arr[i_from][i_from];
    for (int j = 0; j < _cols; j++) {
        _arr[i_from][j] = _arr[i_from][j] / coeff;
    }
    fr->set(i_from, 0, (fr->get(i_from, 0) / coeff));

}

// вычитает из обрабатываемой строки главную,
// умноженную на рассчитанный коэффициент
template <typename T>
void  Matrix<T>::substract_rows_Gauss(int  i_from,  int  i_to,  T  coeff,
Matrix<double>* fr) {
    for (int j = 0; j < _cols; j++) {
        _arr[i_to][j] -= _arr[i_from][j] * coeff;
    }
    fr->set(i_to, 0, fr->get(i_to, 0) - (fr->get(i_from, 0) * coeff));
}

// выполняет прямой ход метода Гаусса
template <typename T>
```

```
void Matrix<T>::forward_Gauss(Matrix<double>* fr) {
    //cout << "Forward: " << endl;
    int i;
    for (i = 0; i < get_rows() - 1; i++) {
        for (int k = i + 1; k < get_rows(); k++) {
            make_diag_one_Gauss(i, fr);
            T coeff = _arr[k][i];
            substract_rows_Gauss(i, k, coeff, fr);
        //    print(fr);
        }
    }
    make_diag_one_Gauss(i, fr);
    //print(fr);
}

// выполняет обратный ход метода Гаусса
template <typename T>
void Matrix<T>::backward_Gauss(Matrix<double>* fr) {
    //cout << "Backward: " << endl;
    int i;
    for (i = get_rows() - 1; i > 0 ; i--) {
        for (int k = i - 1; k >= 0; k--) {
            T coeff = _arr[k][i];
            substract_rows_Gauss(i, k, coeff, fr);
            //print(fr);
        }
    }
    //print(fr);
}

// выполняет приведение матрицы к диагональному виду методом Гаусса
template <typename T>
void Matrix<T>::Gauss(Matrix<double>* fr) {
    forward_Gauss(fr);
    backward_Gauss(fr);
}

// печатает переданное сообщение, саму матрицу
// и матрицу свободных членов
template <typename T>
void Matrix<T>::print(Matrix<double>* fr, string message) const {
    cout.precision(3);
    cout << message << endl;
    for (int i = 0; i < _rows; i++) {
        cout << "| ";
        for (int j = 0; j < _cols; j++) {
            cout << setw(5) << _arr[i][j] << " ";
        }
        cout << "|";
        if (fr != NULL) {
            cout << "  |" << setw(5) << fr->get(i, 0) << "|" << endl;
        } else {
            cout << endl;
        }
    }
    cout << endl;
}
```

```
// меняет местами строки
template <typename T>
void Matrix<T>::swap_rows(int i1, int i2) {
    check_i_j(i1, 0);
    check_i_j(i2, 0);
    T sw;
    for (int j = 0; j < _cols; j++) {
        sw = _arr[i1][j];
        _arr[i1][j] = _arr[i2][j];
        _arr[i2][j] = sw;
    }
    return;
}

// удаляет матрицу и очищает память
template <typename T>
Matrix<T>::~Matrix() {
    for (int i = 0; i < _rows; i++) {
        delete[] _arr[i];
    }
    delete[] _arr;
}

// удаляет указанные в j_dels столбцы матрицы (всего n штук)
template <typename T>
void Matrix<T>::delete_cols(int j_dels[], const int n) {
    T** new_arr = new T*[_rows];
    for (int i = 0; i < _rows; i++) {
        new_arr[i] = new T[_cols - n];
    }

    bool del[_cols];

    for (int i = 0; i < _cols; i++) {
        del[i] = false;
    }

    for (int j = 0; j < n; j++) {
        del[j_dels[j]] = true;
    }
    cout << endl;


    for (int i = 0; i < _rows; i++) {
        int c = 0;
        for (int j = 0; j < _cols; j++) {
            if (!del[j]) {
                new_arr[i][c] = _arr[i][j];
                c++;
            }
        }
    }

    for (int i = 0; i < _rows; i++) {
        delete[] _arr[i];
    }
```

```
    delete[] _arr;

    _arr = new_arr;
    _cols = _cols - n;
}

template class Matrix<double>;
```

3) Файл *matrix.hpp*:

```
#ifndef MATRIX_H
#define MATRIX_H

#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

template <typename T>
class Matrix {
private:
    T** _arr;
    int _rows;
    int _cols;
public:
    // методы, относящиеся к методу Гаусса
    void make_diag_one_Gauss(int i_from, Matrix<double>* fr);
        void substract_rows_Gauss(int i_from, int i_to, T coeff,
Matrix<double>* fr);
    void forward_Gauss(Matrix<double>* fr);
    void backward_Gauss(Matrix<double>* fr);
    void Gauss(Matrix<double>* fr);

    // основные используемые методы для матриц
    Matrix(const int rows = 1, const int cols = 1);
    const int get_rows() const;
    const int get_cols() const;
    const T get(int i, int j) const;
    void set(int i, int j, T val);
    const bool check_i_j(int i, int j) const;
    void swap_rows(int i1, int i2);

    // дополнительные методы для комбинаторики
    void delete_cols(int j_dels[], const int n);

    void print(Matrix<double>* fr, string message = "") const;
    ~Matrix();
};

#endif // MATRIX_H
```

Результаты работы программы.

X1 = 1.33333, X2 = 3.33333, X3 = -0.666667, X4, X5 = 0

X1 = 2, X2 = 2, X4 = 2, X3, X5 = 0

X1 = 1, X2 = 3, X5 = 1, X3, X4 = 0

X1 = 3, X3 = 1, X4 = 5, X2, X5 = 0

X1 = -2, X3 = 6, X5 = 10, X2, X4 = 0

X1 = 4, X4 = 6, X5 = -2, X2, X3 = 0

X2 = 6, X3 = -2, X4 = -4, X1, X5 = 0

X2 = 2, X3 = 2, X5 = 4, X1, X4 = 0

X2 = 4, X4 = -2, X5 = 2, X1, X3 = 0

X3 = 4, X4 = 2, X5 = 6, X1, X2 = 0

Источники

[1] https://ru.wikipedia.org/wiki/Недоопределенная_система

[2] https://math1.ru/education/sys_lin_eq/kapelli.html

[3] https://math1.ru/education/sys_lin_eq/basis1.html

[4]http://bigor.bmstu.ru/?cnt/?doc=Comby/nat_numb_comb_enum.mod/?cou=Comby/base.cou