МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
“Национальный исследовательский университет ИТМО”

ФАКУЛЬТЕТ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ
Кафедра проектирования и безопасности компьютерных систем

ЛАБОРАТОРНАЯ РАБОТА №6

по дисциплине
"ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА"

по теме:
«Векторные произведения»

Выполнил: Нгуен Ле Минь
Группа: N3251
Преподаватель: Гришенцев А.Ю.

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург
2021

Задание 6.1: Скалярное и векторное произведения векторов

1: Задание

Наименование задачи: скалярное и векторное произведения векторов.

Вид решения: программа и отчёт.

Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: скалярное и векторное произведение векторов (строк на строки и столбцов на столбцы) из одной матрицы.

Матрицу и её размер вводить в программу из консоли или из файла. Предусмотреть возможность выбора размера матрицы без перекомпиляции программы, размер матрицы NN, где $N = 1, 2, 3, \ldots$. Оценить вычислительную сложность скалярного и векторного произведения векторов.

2: Теория

Пусть в n-мерном пространстве E_n имеем векторы :

$x = (x_1, x_2, x_3, ..., x_n)$ и $y = (y_1, y_2, y_3, ..., y_n)$

Скалярным произведением векторов n-мерного пространства называется сумма произведения одноименных координат.

$$[\vec{a}, \vec{b}] = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix},$$

где $\mathbf{i} = (1, 0, 0)$, $\mathbf{j} = (0, 1, 0)$, $\mathbf{k} = (0, 0, 1)$

Векторным произведением двух вектором называется такой вектор, равный по величине значению мнемонического определителя матрицы, составленной из этих же двух исходных векторов.

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Laboratory : 6
 */

#include <iostream>
using namespace std;

void solver(){
    int N = 0;
    cout << "Enter the size of the matrix : ";
    cin >> N;
    if (N != 3){
        cout << "Invalid size !!!" << endl;
        exit(1);
    }
    auto** arr = new double* [N];
    auto** tmparr = new double* [N];
    auto** minor = new double* [N - 1];
    for (int i = 0; i < N - 1; i++)
        minor[i] = new double[N - 1];
    auto* cords = new double[N]();
    for (int i = 0; i < N; i++) {
        arr[i] = new double[N]();
        tmparr[i] = new double[N]();
        for (int j = 0; j < N; j++) {
            cout << "Enter [" << i << "][" << j << "]: ";
            cin >> arr[i][j];
            tmparr[i][j] = arr[i][j];
```

```
        }
    }
    int ki1, ki2, kj1, kj2;
    cout << "columns (1) or rows (0)? " << endl;
    int ans;
    cin >> ans;
    if (ans == 1) {
        cout << "Enter first and second vectors separated by space " << endl;
        cin >> kj1 >> kj2;
        if ((kj1 > 2) || (kj1 < 0) || (kj2 > 2) || (kj2 < 0)){
            cout << "You are trying to input a non-existent vector !" << endl;
            exit(1);
        }
        cords[0] = arr[1][kj1] * arr[2][kj2] - arr[1][kj2] * arr[2][kj1];
        cords[1] = -1 * (arr[0][kj1] * arr[2][kj2] - arr[0][kj2] * arr[2][kj1]);
        cords[2] = arr[0][kj1] * arr[1][kj2] - arr[0][kj2] * arr[1][kj1];
        double scal = 0;
        for (int i = 0; i < 3; i++)
            scal += arr[i][kj1] * arr[i][kj2];
        cout << "Vector product: " << std::endl;
        cout << cords[0] << "*i + " << cords[1] << "*j + " << cords[2] << "*k\n";
        cout << "Scalar product:  " << scal;
    }
    else if (ans == 0) {
        cout << "Enter first and second  separated by space" << endl;
        cin >> ki1 >> ki2;
        if ((ki1 > 2) || (ki1 < 0) || (ki2 > 2) || (ki2 < 0)){
            cout << "You are trying to throw invalid inputs !!!" << endl;
            exit(1);
        }
        cords[0] = arr[ki1][1] * arr[ki2][2] - arr[ki2][1] * arr[ki1][2];
        cords[1] = -1 * (arr[ki1][0] * arr[ki2][2] - arr[ki2][0] * arr[ki1][2]);
        cords[2] = arr[ki1][0] * arr[ki2][1] - arr[ki2][0] * arr[ki1][1];
        double scal = 0;
        for (int i = 0; i < 3; i++)
            scal += arr[ki1][i] * arr[ki2][i];
        cout << "Vector product :" << endl;
        cout << cords[0] << "*i + " << cords[1] << "*j + " << cords[2] << "*k" << endl;
        cout << "Scalar product " << scal;
    }
    else {
        cout << "No output can be imagined !" << endl;
        exit(1);
    }
    for(int i = 0; i < N; i++)
        delete [] arr[i], tmparr[i], minor[i];
    delete [] arr, tmparr, minor, cords;
}

int main() {
    solver();
    return 0;
}
```

4: Выводы программы

Enter the size of the matrix : 3
Enter [0][0]: 1
Enter [0][1]: 4
Enter [0][2]: 5
Enter [1][0]: 7
Enter [1][1]: 10
Enter [1][2]: 15
Enter [2][0]: 36
Enter [2][1]: -9
Enter [2][2]: 4
columns (1) or rows (0)?
1 Enter first and second separated by space
1 2
Vector product:
175*i + -61*j + 10*k
Scalar product: 134

При выполнении этого задания мы научились средствами языка C++ реализовывать скалярное и векторное произведение векторов.

Задание 6.2 : Ортогонализация методом Грамма-Шмидта

1: Задание

Разработать алгоритм и написать программу, реализующую ортогонализацию методом Грамма-Шмидта строк из одной матрицы. Матрицу и её размер вводить в программу из консоли или из файла. Предусмотреть возможность выбора размера матрицы без перекомпиляции программы, размер матрицы $N * N$, где $N = 1, 2, 3, \ldots$. Выполнить проверку ортогональности и нормировки с помощью вычисления скалярного произведения и расчёта длин векторов. Оценить вычислительную сложность скалярного и векторного произведения векторов.

2: Теория

Ортогонализация методом Грама-Шмидта — алгоритм, строящий на основе множества из n линейно-независимых векторов множество из n ортогональных векторов так, что каждый из полученных векторов может быть выражен линейной комбинацией исходных векторов.

Векторы a и b называются ортогональными, если их скалярное произведение равно нулю : $(a, b) = 0$

Системы $S_1 = (a_1, a_2, ..., a_n)$ и $S_2 = (b_1, b_2, ..., b_n)$ называются эквивалентными, когда векторы каждой из систем, линейно выражаются через векторы, другой системы.

Допустим, у нас есть линейно независимая система $S = (a_1, a_2, ..., a_n)$.Тогда всегда найдется такая система $S_* = (b_1, b_2, ..., b_n)$, которая будет эквивалентной и ортогональной к S которая получается следующим методом:

1) $b_1 = a_1$

2) $b_j = a_j + \sum_{i=1}^{j-1} \lambda_{ij} b_i, 2 \leq j \leq k$, при $\lambda_{ji} = -\frac{(a_j, b_i)}{(b_i, b_i)}$

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Lab : 6
 */

#include <iostream>
using namespace std;

void show(float **a,int n);
void fill(float **a,int n);
void fill_t(float **a,int n);
float scalyar(float **a,float **b,int col1,int col2,int n);
void ort(float **a,float **r,float **t,int n);
void multiply(float **r,float **t,float **ans,int n);
void get_r_mat(float **a,float **r,float **t,int col,int n);

void solver(){
    float **a,**r,**t,**ans;
    int n;
    cout << "Введите размерность матрицы: ";
    cin >> n;
    a = new float*[n];
    r = new float*[n];
    t = new float*[n];
    ans = new float*[n];
    for(int i=0; i<n; i++){
        a[i] = new float[n];
        r[i] = new float[n];
        t[i] = new float[n];
        ans[i] = new float[n];
    }
    fill(a,n);
    cout << "MATRIX A" << endl;
```

```cpp
    show(a,n);
    for(int i=0; i<n; i++){
        for(int j=0;j<n;j++)
            r[i][j] = 0;
        r[i][0] = a[i][0];
    }
    cout << "MATRIX R" << endl;
    show(r,n);
    fill_t(t,n);
    cout << "MATRIX T" << endl;
    show(t,n);
    ort(a,r,t,n);
    cout << "ОРТОГОНАЛЬНЫЕ" << endl;
    cout << "MATRIX A" << endl;
    show(a,n);
    cout << "MATRIX R" << endl;
    show(r,n);
    cout << "MATRIX T" << endl;
    show(t,n);
    multiply(r,t,ans,n);
    cout << "ПРОВЕРКА" << endl;
    cout << "MATRIX R*T" << endl;
    show(ans,n);
}

int main(){
    solver();
    return 0;
}

void show(float **a,int n){
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++)
            cout << a[i][j] << "\t";
        cout << endl;
    }
    cout << endl;
}

void fill(float **a,int n){
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++){
            cout << "Введите a[" << i << "][" << j <<"]: ";
            cin >> a[i][j];
        }
}

void fill_t(float **a,int n){
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            a[i][j] = 0;
    for(int i=0; i<n; i++)
        a[i][i] = 1;
}

float scalyar(float **a,float **b,int col1,int col2,int n){
    float q = 0;
    for(int i=0; i<n; i++)
        q = q+(a[i][col1]*b[i][col2]);
    return q;
}
```

```
void ort(float **a,float **r,float **t,int n){
    float s1 = 0;
    float s2 = 0;
    int c = 1;
    for(int i=0; i<n; i++){
        if(i>0)
            get_r_mat(a,r,t,i,n);
        for(int j=c; j<n; j++){
            s1 = scalyar(a,r,j,i,n);
            s2 = scalyar(r,r,i,i,n);
            t[i][j] = s1/s2;
        }
        c = c+1;
    }
}

void multiply(float **r,float **t,float **ans,int n){
    float *change;
    float q = 0;
    change = new float[n];
    for(int i=0; i<n; i++){
        for(int l=0; l<n; l++){
            for(int j=0; j<n; j++){
                change[j] = r[i][j]*t[j][l];
                q = q+change[j];
            }
            ans[i][l] = q;
            q = 0;
        }
    }
}

void get_r_mat(float **a,float **r,float **t,int col,int n){
    for (int i=0;i<n;i++){
        for(int  j=0;j<col;j++)
            r[i][col]+=t[j][col]*r[i][j];
        r[i][col]=a[i][col]-r[i][col];
    }
}
```

4: Выводы программы

Введите размерность матрицы: 3
Введите a[0][0]: 6
Введите a[0][1]: 1
Введите a[0][2]: -9
Введите a[1][0]: 10
Введите a[1][1]: 15
Введите a[1][2]: -7
Введите a[2][0]: 5
Введите a[2][1]: 4
Введите a[2][2]: 12
MATRIX A
6 1 -9
10 15 -7
5 4 12
MATRIX R
6 0 0
10 0 0
5 0 0
MATRIX T
1 0 0
0 1 0
0 0 1
ОРТОГОНАЛЬНЫЕ
MATRIX A
6 1 -9
10 15 -7
5 4 12
MATRIX R
6 -5.55901 -6.1708
10 4.06832 -3.34986
5 -1.46584 14.1047
MATRIX T
1 1.09317 -0.397516
0 1 0.0798897
0 0 1
ПРОВЕРКА
MATRIX R*T
6 1 -9
10 15 -7
5 4 12

В ходе работы мы научился как написать программу реализующую: ортогонализацию методом Грамма-Шмидта строк из одной матрицы.

Задание 6.3: Преобразование Фурье

1: Задание

Наименование задачи: преобразование Фурье.
Вид решения: программа и отчёт.
Реализация решения: язык C или C++.
Разработать алгоритм и написать программу реализующую: преобразование Фурье для разложения последовательности комплекснозначных чисел в ряд Фурье. Длина последовательности произвольная. Оценить вычислительную сложность преобразования Фурье.

2: Теория

Переобразование Фурье представляет собой операцию, сопоставляющую одной функции вещественной переменной другую функцию, вообще говоря, комплексной переменной. Эта новая функция описывает коэффициенты («амплитуды») при разложении исходной функции на элементарные составляющие — гармонические колебания с разными частотами (подобно тому, как музыкальный аккорд может быть выражен в виде суммы музыкальных звуков, которые его составляют).

Преобразование Фурье функции f вещественной переменной является интегральным и задаётся следующей формулой:

$\hat{f}(\omega) = \frac{1}{\sqrt{2\pi}} \int\limits_{-\infty}^{\infty} f(x) e^{-ix\omega}\, dx.$

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Lab : 6
 */

#include <complex>
#include <vector>
#include <iostream>
#include <cmath>
using namespace std;

#define pi 3.14159265358979323846
typedef complex<double> base;

void fft (std::vector<base> & a, bool invert) {
    int n = (int) a.size();
    if (n == 1)  return;
    std::vector<base> a0 (n/2),  a1 (n/2);
    for (int i=0, j=0; i<n; i+=2, ++j) {
        a0[j] = a[i];
        a1[j] = a[i+1];
    }
    fft (a0, invert);
    fft (a1, invert);
    double ang = 2*pi/n * (invert ? -1 : 1);
    base w (1),  wn (cos(ang), sin(ang));
    for (int i=0; i<n/2; ++i) {
        a[i] = a0[i] + w * a1[i];
        a[i+n/2] = a0[i] - w * a1[i];
        if (invert)
            a[i] /= 2,  a[i+n/2] /= 2;
        w *= wn;
    }
}
```

```cpp
void solver(){
    int N;
    bool check;
    cout << "Введите 0, если хотите прямое преобразование Фурье или 1 если обратное: ";
    cin >> check;
    complex<double> buff;
    cout << "Введите длину последовательности (степень двойки): ";
    cin >> N;
    vector<base> a(N);
    cout << "Введите комплексные числа вида (a,b)" << std::endl;
    for(int i = 0; i < N; i ++){
        cout << "Введите комплексное число [" << i << "]: ";
        cin >> buff;
        a[i] = buff;
    }
    fft(a, check);
    cout << "Ответ: " << std::endl;
    for(int i = 0; i < N; i++) {
        cout << a[i];
    }
}

int main(){
    solver();
    return 0;
}
```

4: Выводы программы

Введите 0, если хотите прямое преобразование Фурье или 1 если обратное:
1
Введите длину последовательности (степень двойки): 8
Введите комплексные числа вида (a,b)
Введите комплексное число [0]: (1,2)
Введите комплексное число [1]: (2,3)
Введите комплексное число [2]: (3,4)
Введите комплексное число [3]: (4,5)
Введите комплексное число [4]: (5,6)
Введите комплексное число [5]: (6,7)
Введите комплексное число [6]: (7,8)
Введите комплексное число [7]: (8,9)
Ответ:
(4.5,5.5)(-1.70711,0.707107)(-1,-1.11022e-16)(-0.707107,-0.292893)(-0.5,-0.5)(-0.292893,-0.707107)(1.11022e-16,-1)(0.707107,-1.70711)

Мы научились средствами C++ реализовывать преобразования Фурье для разложения последовательности комплекснозначных чисел.

Задание 6.4: Матрица унитарного преобразования Фурье

1: Задание

Наименование задачи: матрица преобразования Фурье.

Вид решения: программа и отчёт.

Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: вычисление, вывод в консоль матрицы преобразования Фурье и проверки полученной матрицы на унитарность.

Предусмотреть возможность выбора размера матрицы без перекомпиляции программы, размер матрицы $N * N$, где $N = 2n, n = 1, 2, 3, \ldots.$

2: Теория

Дискретное преобразование Фурье является линейным преобразованием, которое переводит вектор временных отсчётов x в вектор спектральных отсчётов той же длины. Таким образом преобразование может быть реализовано как умножение симметричной квадратной матрицы на вектор:

$\vec{X} = A\vec{x}$

Матрица A имеет вид :

$$\hat{A} = \begin{pmatrix} 1 & 1 & 1 & 1 & \ldots & 1 \\ 1 & e^{-\frac{2\pi i}{N}} & e^{-\frac{4\pi i}{N}} & e^{-\frac{6\pi i}{N}} & \ldots & e^{-\frac{2\pi i}{N}(N-1)} \\ 1 & e^{-\frac{4\pi i}{N}} & e^{-\frac{8\pi i}{N}} & e^{-\frac{12\pi i}{N}} & \ldots & e^{-\frac{2\pi i}{N}2(N-1)} \\ 1 & e^{-\frac{6\pi i}{N}} & e^{-\frac{12\pi i}{N}} & e^{-\frac{18\pi i}{N}} & \ldots & e^{-\frac{2\pi i}{N}3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e^{-\frac{2\pi i}{N}(N-1)} & e^{-\frac{2\pi i}{N}2(N-1)} & e^{-\frac{2\pi i}{N}3(N-1)} & \ldots & e^{-\frac{2\pi i}{N}(N-1)^2} \end{pmatrix}.$$

Элементы матрицы задаются следующей формулой :

$A(m, n) = e^{-2\pi i \frac{(m-1)(n-1)}{N}}$

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Lab : 6
 */
#include <iostream>
#include <complex>
#include <cmath>
#include <vector>
using namespace std;
typedef complex<double> base;
#define pi 3.14159265358979323846

void multiply(std::vector<base> &r,std::vector<base> &t,std::vector<base> &ans,int n){
    base *change;
    base q = 0;
    change = new base[n];
    for(int i=0; i<n; i++){
        for(int l=0; l<n; l++){
            for(int j=0; j<n; j++){
                change[j] = r[i* n + j]*t[j*n + l];
                q = q+change[j];
            }
            ans[i*n + l] = q/(base)n;
            q = 0;
        }
    }
```

```
}

void solver(){
    int N;
    cout << "Введите размерность матрицы: ";
    cin >> N;
    vector<base> a(N*N);
    vector<base> b(N*N);
    vector<base> c(N*N);
    for (int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            base tmp1(cos(-2*(i)*(j)*pi/N),sin(-2*(i)*(j)*pi/N));
            a[i*N + j] = tmp1;
            base tmp2(cos(-2*(i)*(j)*pi/N),-sin(-2*(i)*(j)*pi/N));
            b[i*N + j] = tmp2;
            cout << a[i*N + j] << "\t";
        }
        cout << endl;
    }
    multiply(a,b,c,N);
    cout << "ПРОВЕРКА" << endl;
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++) {
            cout << (complex<int>) c[i * N + j] << "\t";
        }
        cout << endl;
    }
}

int main(){
    solver();
    return 0;
}
```

4: Выводы программы

Введите размерность матрицы: 4
(1,0) (1,0) (1,0) (1,0)
(1,0) (6.12323e-17,-1) (-1,-1.22465e-16) (-1.83697e-16,1)
(1,0) (-1,-1.22465e-16) (1,2.44929e-16) (-1,-3.67394e-16)
(1,0) (-1.83697e-16,1) (-1,-3.67394e-16) (5.51091e-16,-1)
ПРОВЕРКА
(1,0) (0,0) (0,0) (0,0)
(0,0) (1,0) (0,0) (0,0)
(0,0) (0,0) (1,0) (0,0)
(0,0) (0,0) (0,0) (1,0)
Мы научились средствами C++ реализовывать вычисления матрицы преобразования Фурье.

Задание 6.5: Прямое и обратное преобразование Фурье на основе умножения на матрицу преобразования Фурье

1: Задание

Наименование задачи: прямое и обратное преобразование Фурье на основе умножения на матрицу преобразования Фурье.

Вид решения: программа и отчет.

Реализация решения: язык C или C++.

Разработать алгоритм и написать программу реализующую: прямое и обратное преобразование Фурье на основе умножения на матрицу преобразования Фурье.

Предусмотреть возможность выбора размера последовательности преобразования без перекомпиляции программы, размер последовательности , где . Оценить вычислительную сложность прямого и обратного преобразования Фурье на основе умножения на унитарную матрицу преобразования Фурье.

2: Теория

Дискретное преобразование Фурье (ДПФ) может быть выполнено посредством умножения на так называемую матрицу ДПФ.

$X = Zx$, (1)

Где Z – матрица ДПФ, x – входящий сигнал, – дискретное Фурье преобразование сигнала.

Обратное преобразование можно выполнив умножив X на комплексно сопряженную матрицу Z и нормировав все элементы делением на N – длину последовательности.

3: Код

```
/*
 * Author : Nguyen Le Minh
 * Group : N3251
 * Lab : 6
 */
#include <iostream>
#include <complex>
#include <cmath>
#include <vector>
using namespace std;

typedef complex<double> base;

#define pi 3.14159265358979323846

void multiply(vector<base> &r,vector<base> &t,vector<base> &ans,int n){
    base change;
    base q = 0;
    for(int l=0; l<n; l++){
        for(int j=0; j<n; j++){
            change = r[l*n + j]*t[j];
            q = q+change;
        }
        ans[l] = q/(base)n;
        q = 0;
    }
}

void solver(){
    int N, koef;
    label:
    cout << "Введите 1 для прямого преобразования или -1 для обратного: ";
    cin >> koef;
    if(koef != 1 && koef != -1)
        goto label;
```

```
    cout << "Введите количество элементов (1,2,4,8...): ";
    cin >> N;
    vector<base> a(N*N);
    vector<base> arr(N);
    vector<base> ans(N);
    cout << "Введите вектор комплексный типа (a,b): " << endl;
    for (int i = 0; i < N; i++){
        cout << "Введите элемент arr[" << i << "]: ";
        cin >> arr[i];
        for(int j = 0; j < N; j++){
            base tmp(cos(-2*(i)*(j)*pi/N*koef),sin(-2*(i)*(j)*pi/N*koef));
            a[i*N + j] = tmp;
        }
    }
    multiply(a, arr, ans, N);
    for(int i = 0; i < N; i++) {
        cout << ans[i] << endl;
    }
}

int main(){
    solver();
    return 0;
}
```

4: Выводы программы

Введите 1 для прямого преобразования или -1 для обратного: 1
Введите количество элементов (1,2,4,8...): 8
Введите вектор комплексный типа (a,b):
Введите элемент arr[0]: (7,8)
Введите элемент arr[1]: (8,11)
Введите элемент arr[2]: (-2,3)
Введите элемент arr[3]: (-6,4)
Введите элемент arr[4]: (9,12)
Введите элемент arr[5]: (-9,8)
Введите элемент arr[6]: (7,12)
Введите элемент arr[7]: (2,5)
(2,7.875)
(1.01149,0.183058)
(2.625,0.25)
(-1.15793,-2.77405)
(3.25,0.875)
(-3.76149,1.06694)
(0.125,1)
(2.90793,-0.475951)

Мы научились средствами C++ реализовывать прямое и обратное преобразования Фурье на основе умножения на матрицу преобразования Фурье.