**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики и информатики**

Лабораторная работа № 2

Выполнил: Ульяницкий Владимир Александрович

2 курс, 3 группа

Преподаватель: Полещук Максим Александрович

Минск, 2018

**Задание 1.** Разработать программу численного решения задачи ортогонализации векторов (вычислить *QR*-разложение) методом Хаусхолдера. Пусть дана матрица *A*

.

Для вычислений выбрать параметры:

1. *m* — номер в списке студенческой группы;
2. *n* — одно из чисел в пределах от 1 до 6.

Ненулевые компоненты векторов  (кроме первой такой ненулевой компоненты) в алгоритме Хаусхолдера хранить на месте матрицы *A*, первую компоненту хранить в отдельном векторе *d*. На каждом шаге алгоритма умножение слева матрицы Хаусхолдера на матрицу  вычислять эффективно — воспользоваться тем свойством, что произведение вектора  на матрицу  является вектором и произведение вектор-столбца  на вектор-строку  является произведением двух скаляров для каждого элемента получаемой матрицы результата. Для предотвращения получения машинного нуля реализовать вычисление евклидовой нормы многокомпонентного вектора аналогично функции *hypot*.

Вычислить и представить в отчёте следующие величины:

1. Преобразованная матрица  после первого шага алгоритма и первая компонента вектора *d*.
2. Выходная (преобразованная) матрица , *R* и вектор *d*.
3. Евклидовы нормы вектор-столбцов матрицы *Q* (преобразованная матрица ).
4. Скалярные произведения последнего вектор-столбца c остальными столбцами матрицы .

**2. Входные данные**

m = 23, n = 3

**3. Ход работы**

1.Создание матрицы, соответствующей условию поставленной задачи.

2. Вычисление *w* и *H=H(w)=E−2wwT.*

3. Преобразование = *– 2wTwA,* получение первой компоненты вектора *d*, равной *A*[0][0].

4. Выполнение 3 шагов алгоритма:

Шаг *k, 1≤k≤n–1*. Строим матрицу отражения *Hk* порядка *n–k+1*, переводящую вектор размерности n–k+1



в вектор, коллинеарный вектору (размерности *n–k+1*):

Hk=En–k+1−2,

wk=, (5)

где ; если = 0, то *wk*=.

Умножим матрицу *A(k–1) (A(0) = A,* если *k = 1)* слева на ортогональную блочную матрицу

Qk= (6)

В результате получим матрицу

*A(k)=QkA(k–1)* (7)

у которой все поддиагональные элементы столбцов с первого по k-й равны нулю.

Имеем формулы преобразования элементов исходной матрицы:

*if k≥2: , 1≤i≤k–1, 1≤j≤n,*

*if k≥2:=0, k≤i≤n, 1≤j≤k–1,*

*=−2(,wk)wk, k≤j≤n.*  (8)

Матрица *A*(*n–*1) является верхней треугольной матрицей — верхний треугольник которой есть матрица *R*.

Матрица *Q = Qn–1…Q2Q1.*

**4. Вывод программы**

Matrix A(0):

236 1 0 0

1 235 1 0

2 0 236 1

3 0 2 238

6 1 0 0

9 2 1 0

12 1 0 1

15 0 2 1

Matrix H(0):

-0.996 -0.00422 -0.00844 -0.0127 -0.0253 -0.038 -0.0506 -0.0633

-0.00422 1 -1.78e-05 -2.68e-05 -5.35e-05 -8.03e-05 -0.000107 -0.000134

-0.00844 -1.78e-05 1 -5.35e-05 -0.000107 -0.000161 -0.000214 -0.000268

-0.0127 -2.68e-05 -5.35e-05 1 -0.000161 -0.000241 -0.000321 -0.000401

-0.0253 -5.35e-05 -0.000107 -0.000161 1 -0.000482 -0.000642 -0.000803

-0.038 -8.03e-05 -0.000161 -0.000241 -0.000482 0.999 -0.000963 -0.0012

-0.0506 -0.000107 -0.000214 -0.000321 -0.000642 -0.000963 0.999 -0.00161

-0.0633 -0.000134 -0.000268 -0.000401 -0.000803 -0.0012 -0.00161 0.998

Matrix A(1):

-237 -2.14 -2.19 -3.13

-5.96e-07 235 0.995 -0.00663

-1.19e-06 -0.0133 236 0.987

-1.43e-06 -0.0199 1.99 238

-2.86e-06 0.96 -0.0277 -0.0398

-3.81e-06 1.94 0.958 -0.0596

-5.72e-06 0.92 -0.0554 0.92

-7.63e-06 -0.0995 1.93 0.901

d[0] = 0.999

Matrix H(2):

1 0 0 0 0 0 0 0

0 -1 5.65e-05 8.47e-05 -0.00409 -0.00826 -0.00392 0.000424

0 5.65e-05 1 -2.39e-09 1.15e-07 2.33e-07 1.11e-07 -1.2e-08

0 8.47e-05 -2.39e-09 1 1.73e-07 3.5e-07 1.66e-07 -1.79e-08

0 -0.00409 1.15e-07 1.73e-07 1 -1.69e-05 -8e-06 8.65e-07

0 -0.00826 2.33e-07 3.5e-07 -1.69e-05 1 -1.62e-05 1.75e-06

0 -0.00392 1.11e-07 1.66e-07 -8e-06 -1.62e-05 1 8.29e-07

0 0.000424 -1.2e-08 -1.79e-08 8.65e-07 1.75e-06 8.29e-07 1

Matrix A(2):

-237 -2.14 -2.19 -3.13

6.58e-07 -235 -0.989 0.0243

-1.19e-06 4.66e-09 236 0.987

-1.43e-06 9.31e-09 1.99 238

-2.86e-06 -3.58e-07 -0.0318 -0.0397

-3.81e-06 -7.15e-07 0.95 -0.0595

-5.72e-06 -3.58e-07 -0.0593 0.921

-7.63e-06 3.73e-08 1.93 0.901

Matrix H(3):

1 0 0 0 0 0 0 0

0 1 0 0 0 0 0 0

0 0 -1 -0.00842 0.000135 -0.00403 0.000251 -0.00818

0 0 -0.00842 1 5.66e-07 -1.69e-05 1.06e-06 -3.44e-05

0 0 0.000135 5.66e-07 1 2.71e-07 -1.69e-08 5.51e-07

0 0 -0.00403 -1.69e-05 2.71e-07 1 5.06e-07 -1.65e-05

0 0 0.000251 1.06e-06 -1.69e-08 5.06e-07 1 1.03e-06

0 0 -0.00818 -3.44e-05 5.51e-07 -1.65e-05 1.03e-06 1

Matrix A(3):

-237 -2.14 -2.19 -3.13

6.58e-07 -235 -0.989 0.0243

1.28e-06 -2.3e-09 -236 -3

-1.42e-06 9.28e-09 -5.96e-07 238

-2.86e-06 -3.58e-07 1.12e-08 -0.0394

-3.8e-06 -7.15e-07 -1.79e-07 -0.0675

-5.72e-06 -3.58e-07 1.12e-08 0.921

-7.62e-06 3.72e-08 -4.77e-07 0.884

Matrix A(n-1) matrix:

-237 -2.14 -2.19 -3.13

6.58e-07 -235 -0.989 0.0243

1.28e-06 -2.3e-09 -236 -3

-1.42e-06 9.28e-09 -5.96e-07 238

-2.86e-06 -3.58e-07 1.12e-08 -0.0394

-3.8e-06 -7.15e-07 -1.79e-07 -0.0675

-5.72e-06 -3.58e-07 1.12e-08 0.921

-7.62e-06 3.72e-08 -4.77e-07 0.884

vector d : 0.999 0.00211 0.00422 0.00633 0.0127 0.019 0.0253 0.0317

Matrix R:

-237 -2.14 -2.19 -3.13

0 -235 -0.989 0.0243

0 0 -236 -3

0 0 0 238

0 0 0 0

0 0 0 0

0 0 0 0

0 0 0 0

Norms vector-columns of Q: 1 1 1 1 1 1 1 1

Norms vector-columns of A(n-1): 237 235 236 238

Q:

-0.996 -0.00422 -0.00844 -0.0127 -0.0253 -0.038 -0.0506 -0.0633

0.0048 -1 7.68e-05 0.000115 -0.00402 -0.00816 -0.00379 0.000576

0.0092 -9.49e-06 -1 -0.00836 0.000251 -0.00385 0.000485 -0.00789

-0.0126 5.77e-05 -0.00847 1 -0.000159 -0.000256 -0.000318 -0.000433

-0.0253 -0.00414 2.78e-05 -0.00016 1 -0.000498 -0.00065 -0.000801

-0.0379 -0.00834 -0.00419 -0.000257 -0.000497 0.999 -0.000977 -0.00122

-0.0506 -0.00402 3.75e-05 -0.00032 -0.00065 -0.000978 0.999 -0.0016

-0.0632 0.00029 -0.00845 -0.000435 -0.0008 -0.00122 -0.0016 0.998

Scalar products of the last column of Q and others: 5.96e-08 -5.24e-10 3.73e-09 4.07e-10 5.24e-10 1.4e-09 1.98e-09

Скалярные произведения последнего столбца матрицы Q с остальными близки к *0*. Т. к. последний столбец испытывает наибольшее влияние ошибок округления, из этого следует, что *Q* практически ортогональна, нормы векторов матрицы *Q* равны *1*, что говорит о достаточно высокой точности вычислений.

**5. Листинг**

// Hausholder.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include <iostream>

#include <vector>

#include <cmath>

#include <iomanip>

using namespace std;

const int N = 3, M = 23;

vector<vector<float>> matrix(8);

vector<vector<float>> Q;

vector<float> d;

int sign(float x);

void generate\_matrix() {

matrix[0] = { 6 + 10 \* M, 1, 0, 0 };

matrix[1] = { 1, 2 + 10 \* M + N, 1, 0 };

matrix[2] = { 2, 0, 3 + 10 \* M + N, 1 };

matrix[3] = { 3, 0, 2, 5 + 10 \* M + N };

matrix[4] = { 2 \* N, 1, 0, 0 };

matrix[5] = { 3 \* N, 2, 1, 0 };

matrix[6] = { 4 \* N, 1, 0, 1 };

matrix[7] = { 5 \* N, 0, 2, 1 };

}

vector<vector<float>> generate\_I\_matrix(int size) {

vector<vector<float>> result(size);

for (int i = 0; i < size; ++i) {

for (int j = 0; j < size; ++j) {

if (i == j) {

result[i].push\_back(1);

}

else {

result[i].push\_back(0);

}

}

}

return result;

}

float calculate\_eukl\_norm(vector<float> const& v) {

float r = sign(v[0]) \* v[0];

for (int i = 1; i != v.size(); ++i) {

r \*= sqrt(1 + pow(v[i] / r, 2));

}

return r;

}

int sign(float x) {

return (x >= 0) ? 1 : -1;

}

vector<float> getColumn(int row, int column) {

vector<float> col;

for (int i = row; i < matrix.size(); ++i) {

col.push\_back(matrix[i][column]);

}

return col;

}

vector<float> getColumnFromM(vector<vector<float>> matrix, int row, int column) {

vector<float> col;

for (int i = row; i < matrix.size(); ++i) {

col.push\_back(matrix[i][column]);

}

return col;

}

void writeColumn(int row, int column, vector<float>& info) {

for (int i = row; i < matrix.size(); ++i) {

matrix[i][column] = info[i - row];

}

}

void writeW(int index, vector<float> w) {

d.push\_back(w[0]);

for (int i = index + 1; i < matrix.size(); ++i) {

matrix[i][index] = w[i];

}

}

vector<float> calculate\_w(vector<float> s) {

float normS = calculate\_eukl\_norm(s);

s[0] += sign(s[0]) \* normS;

normS = calculate\_eukl\_norm(s); //modifiedNorm

for (auto& x : s) {

x /= normS;

}

return s;

}

vector<float> calculate\_and\_write\_w(int index) {

vector<float> s = getColumn(index, index);

s = calculate\_w(s);

writeColumn(index, index - 1, s);

return s;

}

decltype(auto) multiplyWWt(vector<float>& w) {

vector<vector<float>> matrix(w.size());

for (int i = 0; i < w.size(); ++i) {

for (int j = 0; j < w.size(); ++j) {

matrix[i].push\_back(w[i] \* w[j]);

}

}

return matrix;

}

decltype(auto) calculate\_H(vector<float>& w) {

vector<vector<float>> wWt = multiplyWWt(w);

for (int i = 0; i < w.size(); ++i) {

for (int j = 0; j < w.size(); ++j) {

if (i == j) {

wWt[i][j] = 1 - 2 \* wWt[i][j];

}

else {

wWt[i][j] \*= -2;

}

}

}

return wWt;

}

vector<float> operator- (vector<float> const& v1, vector<float> const& v2) {

vector<float> result;

if (v1.size() == v2.size()) {

for (int i = 0; i < v2.size(); ++i) {

result.push\_back(v1[i] - v2[i]);

}

}

return result;

}

float operator\* (vector<float> const& v1, vector<float> const& v2) {

try {

if (v1.size() != v2.size())

{ throw "Error. Impossible to multiply"; }

}

catch (char\* s) {

cerr << s;

}

float result = 0;

for (int i = 0; i != v1.size(); ++i) {

result += v1[i] \* v2[i];

}

return result;

}

vector<float> operator\* (float n1, vector<float> v1) {

for (float& temp : v1) {

temp \*= n1;

}

return v1;

}

vector<vector<float>> operator\* (float n1, vector<vector<float>> m) {

for (int i = 0; i < m.size(); ++i) {

for (int j = 0; j < m[0].size(); ++j) {

m[i][j] \*= n1;

}

}

return m;

}

ostream& operator<< (ostream& os, vector<vector<float>> const& vec) {

for (auto x : vec) {

for (auto temp : x) {

os << setprecision(3) << setw(10) << temp << " ";

}

os << endl;

}

return os;

}

ostream& operator<< (ostream& os, vector<float> const& vec) {

for (auto x : vec) {

os << x << " ";

}

cout << endl;

return os;

}

//multiply only matrix of the same size. Different sizes are not supposed.

decltype(auto) operator\*(vector<vector<float>> const& v1, vector<vector<float>> const& v2) {

vector<vector<float>> result(v1.size());

if (v1.size() == v2.size()) {

for (int i = 0; i < v1.size(); ++i) {

for (int j = 0; j < v2[0].size(); ++j) {

//get column

vector<float> columnV2;

for (auto x : v2) {

columnV2.push\_back(x[j]);

}

result[i].push\_back(v1[i] \* columnV2);

}

}

}

return result;

}

vector<float> operator\* (vector<float> const& v1, vector<vector<float>> const& v2) {

vector<float> result, col;

for (int i = 0; i < v2[0].size(); ++i) {

for (int j = 0; j < v2.size(); ++j) {

col.push\_back(v2[j][i]);

}

result.push\_back(v1 \* col);

col.clear();

}

return result;

}

//multiplying col \* row

vector<vector<float>> operator\* (vector<vector<float>> m, vector<float> v) {

vector<vector<float>> result(m.size());

for (int i = 0; i < m.size(); ++i) {

for (int j = 0; j < v.size(); ++j) {

result[i].push\_back(m[i][0] \* v[j]);

}

}

return result;

}

vector<vector<float>> operator- (vector<vector<float>> m1, vector<vector<float>> m2) {

for (int i = 0; i < m1.size(); ++i) {

for (int j = 0; j < m1[0].size(); ++j) {

m1[i][j] -= m2[i][j];

}

}

return m1;

}

vector<vector<float>> transposeVec(vector<float> vec) {

vector<vector<float>> result(vec.size());

for (int i = 0; i < vec.size(); ++i) {

result[i].push\_back(vec[i]);

}

return result;

}

int main() {

generate\_matrix();

cout << "Matrix A(0):\n" << matrix << endl;

auto w = calculate\_w(getColumn(0, 0));

vector<float> x1 = w \* matrix;

vector<vector<float>> y1 = transposeVec(w) \* x1;

y1 = 2 \* y1;

matrix = matrix - y1;

d = w;

Q = calculate\_H(w);

cout << "Matrix H(0):\n" << Q << endl;

//point 1

cout << "Matrix A(1):\n" << matrix << endl;

cout << "d[0] = " << d[0] << endl << endl;

for (int k = 1; k < matrix[0].size() - 1; ++k) {

vector<float> w = calculate\_w(getColumn(k, k));

vector<float> wCopy(8 - w.size());

for (float temp : w) {

wCopy.push\_back(temp);

}

vector<float> x = wCopy \* matrix;

vector<vector<float>> y = transposeVec(wCopy) \* x;

y = 2 \* y;

matrix = matrix - y;

auto Hk = calculate\_H(w);

auto Hn = generate\_I\_matrix(8);

int firstI = matrix.size() - Hk.size();

for (int i = firstI; i < matrix.size(); ++i) {

for (int j = firstI; j < matrix.size(); ++j) {

Hn[i][j] = Hk[i - firstI][j - firstI];

}

}

Q = Hn \* Q;

cout << "Matrix H(" << k + 1 << "):\n" << Hn << endl;

cout << "Matrix A(" << k + 1 << "):\n" << matrix << endl;

}

//point 2

cout << "Matrix A(n-1) matrix:\n" << matrix;

cout << "vector d : " << d;

cout << "Matrix R:\n";

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

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

if (i > j) { cout << setprecision(3) << setw(10) << 0 << " "; }

else { cout << setprecision(3) << setw(10) << matrix[i][j] << " "; }

}

cout << endl;

}

cout << endl;

//point 3

vector<float> temp;

cout << "Norms vector-columns of Q: ";

for (int i = 0; i < 8; ++i) {

for (auto& q : Q) {

temp.push\_back(q[i]);

}

cout << calculate\_eukl\_norm(temp) << " ";

temp.clear();

}

cout << endl << endl;

cout << "Norms vector-columns of A(n-1): ";

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

cout << calculate\_eukl\_norm(getColumn(0, i)) << " ";

}

cout << endl << endl;

cout << "Q: \n" << Q << endl;

//point 4

cout << "Scalar products of the last column of Q and others: ";

temp = getColumnFromM(Q, 0, Q.size() - 1);

for (int i = 0; i < Q[0].size() - 1; ++i) {

cout << setprecision(3) << setw(12) << temp \* getColumnFromM(Q, 0, i);

}

cout << endl;

system("pause");

return 0;

}