Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчет по лабораторной работе №2**

**Дисциплина: «**Практикум по вычислительной математике».

Выполнил

студент гр. 5130901/20003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Вагнер

(подпись)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Н. Цыган

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Санкт-Петербург   
2024

**ЗАДАНИЕ**

**Вариант 24:**

Написать процедуру вычисления матрицы Q и вектора z по заданным числам N , x , y , α , gk где

Решить систему уравнений Qw = z с помощью **DECOMP** и **SOLVE**, если N=6, α = 4, x = 4 + γ, y = 4 - γ, gk = 2k-4 при следующих значениях параметра γ: 1.0; 0.5, 0.25, 0.125. Так как gk - компоненты вектора **точного** решения (убедиться в этом!), использовать g для оценки погрешности по формуле:

||w − g|| / ||g||, где . Объяснить результаты.

Предусмотреть вычисление и вывод числа обусловленности *cond* матрицы Q.

**ХОД РАБОТЫ**

Для выполнения лабораторной работы был использован язык программирования C++, а также библиотека GSL, позволяющая реализовывать и проводить операции с матрицами и векторами, в том числе использовать DECOMP и SOLVE.

Во время работы были поставлены следующие цели:

1. Составить матрицу Q.
2. Вычислить вектор zi.
3. Найти корень уравнения Qw = z при помощи методов DECOMP и SOLVE.
4. Определить число обусловленности *cond* матрицы Q.
5. Оценить погрешность

Для создания и хранения матриц и векторов используется библиотечный класс gls\_matrix и gls\_vector соответственно, выделение памяти происходит в блоке main. Составление матрицы Q и вектора zi происходит внутри функции define. Далее при помощи методов gsl\_linalg\_LU\_decomp, gsl\_linalg\_LU\_solve найдём w – решение уравнения. Алгоритм поиска числа обусловленности нетрудно реализовать самостоятельно – напишем функцию поиска нормы Фробениуса и используем методы gsl\_linalg\_LU\_decomp и gsl\_linalg\_LU\_invert для поиска норм Фробениуса обратной и исходных матриц, результаты алгоритмов перемножим для определения cond(Q). Найдём погрешность программы, отметив что .

Изображение выглядит как текст, снимок экрана, дизайн

Автоматически созданное описаниеРезультат работы программы для всех параметров γ:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

**ВЫВОД**

В ходе работы была изучена математическая библиотека gsl, а именно её подраздел gsl/linalg. Были определены способы нахождения корней матричных уравнений и чисел обусловленности матриц.

Заметим, что элементы вектора w никак не зависят от выбора параметра γ, т.е. остаются неизменными на всех наборах входных данных, что соответствует утверждению, что gk - компоненты вектора точного решения уравнения.

Относительная погрешность находится в рамках погрешности вычислений для чисел типа double, вследствие чего решение можно считать верным.

**ПРИЛОЖЕНИЕ**

#include <iostream>

#include <vector>

#include <gsl/gsl\_linalg.h>

void define(double gamma, int N, gsl\_matrix\* Q, gsl\_vector\* z) {

std::string str = " ";

double gam = gamma;

int gamInt = 0;

while (true) {

gam \*= 10;

gamInt = gam;

if (gamInt % 10 == 0) {

break;

}

str += " ";

}

double element;

std::cout << "Q for gamma = " << gamma << "\n"; // составим матрицу Q

std::cout << "\n";

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

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

if (i == j)

element = 4; // альфа на главной диагонали

else if (j > i)

element = 4 + gamma; // x выше диагонали

else

element = 4 - gamma; // y ниже диагонали

gsl\_matrix\_set(Q, i, j, element);

std::cout << element << " "; // вывод элементов матрицы

std::cout << str;

}

std::cout << "\n";

}

std::cout << "\n";

std::cout << "\nz\_i = \n"; // найдём вектор z

for (int i = 1; i <= N; i++) {

double sum1 = 0.0, sum2 = 0.0;

for (int k = 1; k <= i - 1; k++) { // первая сумма

sum1 += pow(2, k - 4);

}

for (int k = i + 1; k <= N; k++) { // вторая сумма

sum2 += pow(2, k - 4);

}

double res = (4 - gamma) \* sum1 + 4 \* pow(2, i - 4) + (4 + gamma) \* sum2; // z\_i = y \* sum1 + a \* g\_i + x \* sum2

gsl\_vector\_set(z, i - 1, res);

std::cout << res << "\n"; // вывод элементов вектора

}

std::cout << "\n";

}

double frobeniusNorm(int N, gsl\_matrix\* Q) { // найдём норму Фробениуса

double norm = 0.0;

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

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

norm += pow((gsl\_matrix\_get(Q, i, j)), 2); // прибавим квадрат элемента к сумме

}

}

return sqrt(norm); // корень суммы квадратов элементов матрицы

}

double matrixCond(int N, gsl\_matrix\* Q) { // найдём cond матрицы при помощи нормы Фробениуса и методов из библиотеки gsl

double cond; // способ определения cond при помощи библиотеки gsl был найден в документации gsl

int signum;

gsl\_permutation\* perm = gsl\_permutation\_alloc(N);

gsl\_linalg\_LU\_decomp(Q, perm, &signum);

gsl\_matrix\* invertedQ = gsl\_matrix\_alloc(N, N);

gsl\_linalg\_LU\_invert(Q, perm, invertedQ);

cond = frobeniusNorm(N, Q) \* frobeniusNorm(N, invertedQ);

gsl\_permutation\_free(perm);

gsl\_matrix\_free(invertedQ);

return cond;

}

int main() {

int N = 6;

double gamma[] = { 1.0, 0.5 , 0.25 , 0.125 };

for (double y : gamma) {

gsl\_matrix\* Q = gsl\_matrix\_alloc(N, N); // создадим матрицу Q

gsl\_vector\* z = gsl\_vector\_alloc(N); // и вектор z

define(y, N, Q, z);

gsl\_permutation\* perm = gsl\_permutation\_alloc(N); // матрицу permutations

int signum; // число signium

gsl\_vector\* w = gsl\_vector\_alloc(N); // вектор w

gsl\_linalg\_LU\_decomp(Q, perm, &signum);

gsl\_linalg\_LU\_solve(Q, perm, z, w);

std::cout << "\nw = \n";

gsl\_vector\_fprintf(stdout, w, "%g"); // вывод решения уравнения

double cond = matrixCond(N, Q);

std::cout << "\nCondition number cond(Q) = " << cond << ".\n"; // вывод cond(Q)

gsl\_vector\* g = gsl\_vector\_alloc(N);

for (int k = 1; k <= N; k++) {

gsl\_vector\_set(g, k - 1, pow(2, k - 4));

}

double gc = 0.0; // ||g|| = sum g\_k from k=1 to N

for (int k = 1; k <= N; k++) {

gc += pow(2, k - 4);

}

gsl\_vector\_sub(w, g); // в целях экономии памяти определим w = w - g

double err = gsl\_blas\_dnrm2(w) / gc; // отн. погрешность равна ||w|| / ||g||

std::cout << "\nRelative error ||w - g|| / ||g||: " << err << "\n\n";

std::cout << "------------------------------------------------------\n";

gsl\_matrix\_free(Q);

gsl\_vector\_free(z);

gsl\_vector\_free(w);

gsl\_vector\_free(g);

gsl\_permutation\_free(perm);

}

return 0;

}