МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Практическое задание № 1 по учебному курсу
«Численные методы линейной алгебры»
Метод Холецкого
Матрица №2

ОТЧЁТ
о выполненном задании

студента **301** учебной группы факультета ВМК МГУ

Матвеевой Софьи Вячеславовны

гор. Москва

2023г.

Постановка задачи

Требуется решить систему линейных алгебраических уравнений с помощью метода Холецкого

$$Ax = F$$

С квадратной невырожденной матрицей $A \in R^{n \times n}$. Элементы матрицы a_{ij} являются вещественными числами, расположенными на отрезке $[-1;\ 1]$(за исклбчением диогональных элементов). Матрица предоставляется в виде файла в формате csv.

Для проверки метода требуется случайно сгенерировать вектор-столбец $\boldsymbol{x}$ с равномерно распределенными на отрезке [-1; 1] компонентами, вычислить правую часть по формуле $F = Ax$.

Описание метода решения задачи :

Метод Холецкого это метод для решения системы линейных алгебраических уравнений $Ax = F$, где матрица A является симметричной и положительно определенной, т.е.

$$A = A^T \quad \text{и} \quad \Delta_k A > 0, \ k = 1, \ldots n$$

Последнее означает, что все главные миноры матрицы A являются положительными.
Разложением Холецкого матрицы $A \in R^{n \times n}$ называется $A = LL^T$, где L – нижняя треугольная матрица с положительными диагональными элементами.
Основой метода Холецкого является является теорема о единственности разложения Холецкого.

Вычисление матрицы L:
Для вычисления матрицы L использовались следующие формулы:

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}, \qquad k = 1, \ldots n$$

$$l_{ij} = \frac{1}{l_{jj}} (a_{ji} - \sum_{k=1}^{j-1} l_{ik} l_{jk}), \quad i = j + 1, \ldots, n$$

Обратный ход:
Если разложение получено, то решение системы водится к последовательному решению двух линейных систем равнений с треугольными матрицами:

$$Ax = F \ <=> \ LL^T = F$$

$$Ly = F, \quad L^T x = y$$

Решение этих систем находится по формулам обратного метода Гаусса:

$$y_1 = \frac{f_1}{l_{11}}, \quad y_i = \frac{1}{l_{ii}}(f_i - \sum_{j=1}^{i-1} l_{ij} y_j), \quad i = 2, \dots, n$$

$$x_n = \frac{n}{l_{nn}}, \quad x_i = \frac{1}{l_{ii}}(y_i - \sum_{j=j+1}^{n} l_{ji} x_j), \quad i = n-1, \dots, 1$$

Листинг программы

Программа написана на языке «C++14».

В программе были использованы следующие библиотеки и using:

```
#include <iostream>
#include <vector>
#include <string>
#include <fstream>
#include <random>
#include <chrono>

using namespace std;
using namespace chrono;
```

В программе используются следующие псевдонимы для типов данных:

```
typedef vector<vector<double>> matrix;
typedef vector<double> vec;
```

matrix – тип матриц вещественных чисел
vec – тип векторов вещественных чисел

Функция, которая производит умножение матрицы на вектор-столбец:

```
vec MulMV(const matrix &m, const vec &x)
{
   vec res = vec(m.size(), 0.);
   for (size_t i = 0; i < m.size(); i++)
   {
      for (size_t j = 0; j < m.size(); j++)
      {
         res[i] += m[i][j] * x[j];
      }
   }
   return res;
}
```

Класс системы линейных алгебраических уравнений:

```
struct Equasions
{
  long N = 0; //Размерность матрицы
  matrix Matrix; //Матрица
  vec x; //Правильный ответ
  vec res; //Полученный методом Холецкого ответ
  vec f; //Соответствующая x правая часть
```

Конструктор класса: считывает из заданного файла матрицу, считает размерность матрицы:

```
Equasions(string filename)
  {
    string line;
    ifstream in(filename);
    if (in.is_open())
    {
      while (getline(in, line))
      {
        if (line.empty())
          break;
        Matrix.emplace_back();
        for (size_t i = 0; i < line.length(); i++)
        {
          string x = "";
          while (line[i] != ',' && i != line.length())
          {
            x += line[i];
            ++i;
          }
          Matrix[N].push_back(stod(x, nullptr));
        }
        ++N;
      }
    }
    in.close();
  }
```

Функция, которая случайным образом генерирует вектор-столбец решений x с равномерно распределёнными на отрезке [-1; 1] компонентами:

```
void Equasions::CalculateRandX(const unsigned int seed)
  {
    x = vec(N, 0);

    default_random_engine rand(seed);
    uniform_real_distribution<double> dist(0.0, 1.0);

    for (size_t i = 0; i < N; i++)
    {
      x[i] = dist(rand);
```

```
    }
}
```

Функция, вычисляющая правую часть системы уравнений по вектору x:

```
void Equasions::CalculateF()
{
    f = MulMV(Matrix, x);
}
```

Функция, вычисляющая элемент l_{ii} матрицы L:

```
void Equasions::CalculateDiag(size_t i)
{
    double sum = 0;
    for (size_t j = 0; j < i && i > 0; j++) {
        sum += Matrix[j][i] * Matrix[j][i];
    }

    Matrix[i][i] =  sqrt(Matrix[i][i] - sum);
}
```

Функция, вычисляющая элемент l_{ki}, матрицы L^T:

```
void Equasions::CalculateNonDiag(size_t k, size_t i)
{
    double sum = 0;
    for (size_t j = 0; j < k && k > 0; j++) {
        sum += Matrix[j][k] * Matrix[j][i];
    }
    Matrix[k][i] = (Matrix[k][i] - sum) / Matrix[k][k];
}
```

Функция, которая вычисляет элементы k-ой строки матрицы L^T:

```
void Equasions::ProcessRow(size_t k)
{
    for (size_t i = 0; i < k; i++)
        Matrix[k][i] = 0.;

    CalculateDiag(k);

    for (size_t i = k + 1; i < N; i++)
        CalculateNonDiag(k, i);
}
```

Функция, вычисляет матрицу L^T, при этом матрица записывается вместо исходной матрицы A:

```
void Equasions::CalculateL()
```

```
{
    for (size_t i = 0; i < N; i++)
    {
        ProcessRow(i);
    }
}
```

Функция, осуществляющая обратный ход метода Гаусса для нижней треугольной матрицы для системы $Ly = F$:

```
void Equasions::FirstGaussBackward()
{
    res = vec(N, 0.);
    for (size_t i = 0; i < N; i++)
    {
        double d = 0;
        for (size_t j = 0; j < i; j++)
        {
            d += Matrix[j][i] * res[j];
        }
        res[i] = (f[i] - d) / Matrix[i][i];
    }
}
```

Функция, осуществляющая обратный ход метода Гаусса для верхней треугольной матрицы для системы $L^T x = y$:

```
void Equasions::SecondGaussBackward()
{
    for (size_t i = N; i > 0; i--)
    {
        double d = 0;
        for (size_t j = i + 1; j <= N; j++)
        {
            d += Matrix[i - 1][j - 1] * res[j - 1];
        }
        res[i - 1] = (res[i - 1] - d) / Matrix[i - 1][i - 1];
    }
}
```

Функция, вычисляющая погрешность и максимум-норму погрешности:

```
double Equasions::MaxNorm()
{
    double ans = 0;
    for (int i = 0; i < N; ++i)
    {
        ans = max(ans, fabs(x[i] - res[i]));
    }
    return ans;
}
```

Функция, которая проверяет, что матрица является симметричной:

```
bool CheckMatrix(const Equasions m)
{
    matrix mt = m.Matrix;
    bool res = true;
    for (size_t i = 0; i < m.N; i++)
    {
        for (size_t j = 0; j < m.N; j++)
        {
            if (mt[j][i] != mt[i][j])
            {
                res = false;
                break;
            };
        }
    }
    return res;
}
```

Main:

```
int main()
{
    Equasions m = Equasions("SLAU_var_2.csv"); //Считывает матрицу из файла
    if (!CheckMatrix(m)) //Проверка на симметричность
    {
        cout << "Матрица не является симметричной" << endl;
        return 0;
    }
    Equasions copy_m = m;// Сохраняет копию матрицы

    microseconds timev(0ll); //Счетчик для среднего времени выполнения
    double norm = 0; //Переменная для подсчета средней максимум-нормы
    for (size_t i = 0; i < 100; i++) //Производим 100 итераций вычисления решения системы
                                                        уравнений
    {
        m.CalculateRandX(i);
        m.CalculateF();
        microseconds start_time = duration_cast<microseconds>(system_clock::now().time_since_epoch()); //Начинаем
отсчет времени
                                                        работы метода
        m.CalculateL();

        m.FirstGaussBackward();
        m.SecondGaussBackward();
        microseconds end_time = duration_cast<microseconds>(system_clock::now().time_since_epoch()); //Заканчиваем
подсчет
                                                        времени работы
        timev += (end_time - start_time); //Считаем время работы метода
        norm += m.MaxNorm(); //Считаем максимум-норму погрешности
```

```
        m = copy_m; //возвращаем матрице ее первоначальные значения
    }

    cout << "Средняя максимум-норма погрешности = " << norm / 100 << endl;
    cout << "Среднее время, затраченное на вычисление решения = " << (timev / 100).count() << " mcs" << endl;
    return 0;
}
```

Полученные результаты

Средняя максимум-норма погрешности на 100 итерациях = 8.60423e-16

Среднее время, затраченное на вычисление решения на 100 итерациях = 1122 mcs