БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Лабораторная работа №2

Решение систем линейных уравнений, нахождение определителя и обратной матрицы методом квадратного корня

Выполнил:
Студент 2 курса 5 группы ФПМИ
Карасик Семён

Руководитель:
Радкевич Елена Владимировна

Минск, 2017 г.

Оглавление

Постановка задачи

Решить систему линейных уравнений методом квадратного корня. Вычислить определитель системы. Найти ей обратную матрицу. Проверить обратную матрицу. Вывести невязку.

Описание метода нахождения решений системы алгебраических уравнений методом квадратного корня

Метод квадратного корня служит для решения СЛАУ с симметричной матрицей. То есть $Ax=f$, где $A=A^T$, при этом A - положительно определена.

Любую симметричную матрицу можно представить в виде $A=S^T S$, где S - верхняя треугольная матрица со строго положительными элементами на главной диагонали. Тогда наше уравнение принимает вид: $S^T Sx=f$, тогда можем разбить данное уравнение на два:

1. $Ly=f$
2. $Sx=y$

Так как матрица S^T является нижней треугольной, легко получить формулу для вычисления y:

$$y_i=\frac{f_i-\sum_{k=1}^{i-1} s_{ki} y_k}{s_{ii}} \quad , \quad i=\overline{1,n}$$

И x:

$$x_j=\frac{y_j-\sum_{k=j+1}^{n} S_{jk} x_k}{s_{jj}} \quad , \quad j=\overline{n,1}$$

В случае унитарной матрицы A($A=A^*=(\overline{A})^T$) разложение принимает вид $A=S^T DS$

Приведем формулы для нахождения матриц S и D:

$$d_{ii}=sign\left(a_{ii}-\sum_{k=1}^{i-1}|s_{ki}|^2 d_{kk}\right) \quad i=\overline{1,n}$$

$$s_{ii}=\sqrt{\left|a_{ii}-\sum_{k=1}^{i-1}|s_{ki}|^2 d_{kk}\right|} \quad i=\overline{1,n}$$

$$s_{ij}=\frac{a_{ij}-\sum_{k=1}^{i-1}\overline{s_{ki}} d_{kk} s_{kj}}{s_{ii} d_{ii}} \quad , \quad j=\overline{i+1,n}$$

$$s_{ij}=0,\ j<i$$

Нахождение определителя матрицы

$$detA = det\left(S^* DS\right) = det\, S^* \, detD \, detS = det^2 S \, detD = \prod_{i=1}^{n} s_{ii}{}^2 d_{ii}$$

Нахождение обратной матрицы

AX = E => X = A^{-1} , поэтому для нахождения обратной матрицы достаточно решить n уравнений, правая часть которых последовательно равна $(1,0,\ldots,0)^T, (0,1,0,\ldots,0)^T, \ldots, (0,\ldots,0,1)^T$

Замечание для несимметричной исходной матрицы

Метод квадратного корня применим только для симметричных матриц, если исходная матрица G не является симметричной, то можем удовлетворить условиям для применения метода квадратного корня следующим образом:
$G^T G = G^T f$, тогда $A = G^T G$ будет симметричной матрицей, при этом корни уравнения не изменятся. Существуют следующие связи между A и G:
$|detG| = \sqrt{detA}$
$G^{-1} = A^{-1} G^T$

Листинг программы

```
//lab2, v20. Simon Karasik, course 2, group 5.
#include <fstream>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include <vector>
#include <cmath>
#include <stdexcept>

using namespace std;

typedef vector<double> Vector;
typedef vector<Vector> Matrix;
const double EPS = 10E-4;

bool eq(double x, double y) {
   return abs(x - y) < EPS;
}

Matrix loadIdentity(int n) {
   Matrix mat(n);
   for (int i = 0; i < n; i++) {
      mat[i] = Vector(n, 0);
      mat[i][i] = 1;
   }
   return mat;
}
Matrix loadMatrix(int n, int m) {
   Matrix mat(n);
   for (int i = 0; i < n; i++)
      mat[i] = Vector(m, 0);
   return mat;
}
Matrix operator*(const Matrix & a, const Matrix & b)
{
   if (a[0].size() != b.size())
      throw invalid_argument("Bad size of
matricies.");
   int n = a.size(), m = b[0].size(), l = a[0].size();
   Matrix p = loadMatrix(n, m);
   for (int i = 0; i < n; i++)
      for (int j = 0; j < m; j++) {
         double sum = 0;
         for (int k = 0; k < l; k++)
            sum += a[i][k] * b[k]
                  [j];
         p[i][j] = sum;
      }
   return p;
}

Vector operator*(const Matrix & a, const Vector & v)
{
   if (a[0].size() != v.size())
      throw invalid_argument("Bad size.");
   int n = a.size(), m = v.size();
   Vector res = Vector(v.size(), 0);
   for (int i = 0; i < n; i++)
      for (int j = 0; j < m; j++)
         res[i] += a[i][j] * v[j];
   return res;
}

void printVector(const Vector & v, ostream & os, bool
            useScientific = false) {
   if (useScientific) {
      os << scientific;
      for (int i = 0; i < v.size(); i++)
         os << v[i] << '\t';
   } else {
      os << setprecision(4) << fixed;
      for (int j = 0; j < v.size(); j++)
         os << setw(8) << v[j];
   }
   os << endl;
}
void printMatrix(const Matrix & mat, ostream & os,
            bool useScientific = false) {
   for (int i = 0; i < mat.size(); i++)
      printVector(mat[i], os, useScientific);
}

double sqr(double x) {
   return x * x;
}

Matrix transpose(const Matrix & mat) {
   int n = mat.size(), m = mat[0].size();
   Matrix res = loadMatrix(m, n);
   for (int i = 0; i < n; i++)
      for (int j = 0; j< m; j++)
         res[j][i] = mat[i][j];
   return res;
}

int sgn(double x) {
         if (x > 0)
                  return 1;
         else if (x < 0)
                  return -1;
         else
                  return 0;
}

Vector squareRootMethod(
         const Matrix & a,
         const Vector & f,
         double & det,
         Matrix & s,
         Vector & d,
```

```
        Vector & y)
{
        int n = a.size();
        s = loadIdentity(n);
   d = Vector(n);

        for (int i = 0; i < n; i++) {
                double t = 0;
                for (int k = 0; k < i; k++)
                        t += sqr(s[k][i]) * d[k];
                d[i] = sgn(a[i][i] - t);
                s[i][i] = sqrt(abs(a[i][i] - t));

                for (int j = i + 1; j < n; j++) {
                        double t = 0;
                        for (int k = 0; k < i; k++)
                                t += s[k][i] * d[k] *
s[k][j];
                        s[i][j] = (a[i][j] - t) / (s[i][i] *
d[i]);
                }
        }

        y = Vector(n, 0);
        for (int i = 0; i < n; i++) {
                double t = 0;
                for (int k = 0; k < i; k++)
                        t += s[k][i] * y[k];
                y[i] = (f[i] - t) / s[i][i];
        }

        Vector x(n, 0);
        for (int j = n - 1; j >= 0; j--) {
                double t = 0;
                for (int k = j + 1; k < n; k++)
                        t += d[k] * s[j][k] * x[k];
                x[j] = (y[j] - t) / (d[j] * s[j][j]);
        }

        det = 1;
        for (int i = 0; i < n; i++)
                det *= d[i] * sqr(s[i][i]);

        return x;
}
```

```
int main() {
   ifstream fin("input.txt");
   ofstream fout("output.txt");
   int n;
   fin >> n;
   Matrix a = loadMatrix(n, n);
   Vector f = Vector(n);
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
         fin >> a[i][j];
   for (int i = 0; i < n; i++)
      fin >> f[i];
         f = transpose(a) * f;
   a = transpose(a) * a;
         double det;
         Matrix s;
         Vector d, y;
         const Vector x = squareRootMethod(a, f, det,
s, d, y);
   det = sqrt(det);
   Vector error = a * x;
   for (int i = 0 ; i< x.size(); i++)
      error[i] -= f[i];

         fout << "A^T * A:" << endl;
         printMatrix(a, fout);
   fout << "A^T * f:" << endl;
   printVector(f, fout);
         fout << "S:" << endl;
         printMatrix(s, fout);
         fout << "diag(D):" << endl;
         printVector(d, fout);
         fout << "y:" << endl;
         printVector(y, fout);
   fout << "x:" << endl;
   printVector(x, fout);
   fout << "|detA| = " << det << endl;
   fout << "error of x:" << endl;
   printVector(error, fout, true);

         return 0;
}
```

Входные данные

input.txt:

```
5
0.4974 0.0000 -0.1299 0.0914 0.1523
-0.0305 0.3284 0.00000 -0.0619 0.0203
0.0102 -0.0914 0.5887 0.0112 0.0355
0.0305 0.0000 -0.0741 0.5887 0.0000
0.0203 -0.0305 0.1472 -0.0122 0.4263
1.5875 -1.7590 1.4139 1.7702 -2.0767S
```

Выходные данные

```
output.txt
A^T * A:
  0.2498 -0.0116 -0.0579  0.0652  0.0842
 -0.0116  0.1171 -0.0583 -0.0210 -0.0096
 -0.0579 -0.0583  0.3906 -0.0507  0.0639
  0.0652 -0.0210 -0.0507  0.3590  0.0079
  0.0842 -0.0096  0.0639  0.0079  0.2066
A^T * f:
  0.8695 -0.6435  0.1893  1.3373 -0.6290
S:
  0.4998 -0.0231 -0.1158  0.1304  0.1684
  0.0000  0.3415 -0.1786 -0.0526 -0.0166
  0.0000  0.0000  0.5876 -0.0766  0.1368
  0.0000  0.0000  0.0000  0.5774 -0.0078
  0.0000  0.0000  0.0000  0.0000  0.3990
diag(D):
  1.0000  1.0000  1.0000  1.0000  1.0000
y:
  1.7398 -1.7668  0.1281  1.7791 -2.3937
x:
  5.0010 -3.9554  2.0057  3.0003 -5.9993
error of x:
2.2204e-16  1.1102e-16  0.0000e+00  0.0000e+00  -1.1102e-16
|detA| = 2.3103e-02
```