БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №2

Метод квадратного корня решения систем линейных алгебраических уравнений

Вариант 1

Выполнил:

Черепеннников Роман

2 курс 8 группа

Преподаватель:

Радкевич Елена Владимировна

Содержание

Постановка задачи

1. Методом квадратного корня найти решение системы линейных алгебраических уравнений.
2. Вычислить определитель матрицы.

Алгоритм решения

Описание метода нахождения решений системы линейных алгебраических уравнений методом квадратного корня

Метод квадратного корня:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_2 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

Метод квадратного корня можно применить в случае специального вида матрицы системы, а именно $A = A^*$, если $A \in R_{n\times n}$, то $A = A^T$. Если A является эрмитовой (симметрической), то существуют такие верхняя треугольная матрица S с вещественными положительными элементами на главной диагонали и диагональная матрица D с элементами ± 1 на диагонали, что справедливо разложение $A = S^* DS$,

$$S = \begin{pmatrix} s_{11} & s_{12} & \ldots & s_{1n} \\ 0 & s_{22} & \ldots & s_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & s_{nn} \end{pmatrix}, \quad S^* = \begin{pmatrix} s_{11} & 0 & \ldots & 0 \\ \bar{s}_{12} & s_{22} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ \bar{s}_{1n} & \bar{s}_{2n} & \ldots & s_{nn} \end{pmatrix}, \quad D = \begin{pmatrix} d_{11} & 0 & \ldots & 0 \\ 0 & d_{22} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & d_{nn} \end{pmatrix}, \quad \begin{matrix} d_{ii} = \pm 1 \\ i = \overline{1, n} \end{matrix}$$

Если матрица A положительно определена, то $D = E,\ A = S^* S$.
В дальнейшем рассматриваем алгоритм для положительно определённой матрицы.
Находим все ненулевые элементы матрицы S:

$$s_{11} = \sqrt{a_{11}};\quad s_{1j} = \frac{a_{1j}}{s_{11}},\quad j = \overline{2, n};$$

$$s_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} s_{ki}^2};\quad 1 \le i \le n;$$

$$s_{ij} = \begin{cases} \dfrac{a_{ij} - \sum\limits_{k=1}^{i-1} s_{kj} s_{ki}}{s_{ii}}, i < j \\ 0, i > j \end{cases}; \qquad (1)$$

Если разложение вида $A = S^* DS$ получено, то решение исходной системы сводится к решению двух систем с треугольными матрицами

$$\begin{cases} S^T y = b \\ Sx = y \end{cases}.$$

Алгоритм решения последней системы: прямой ход состоит в последовательном нахождении s_{ij} по соотношениям (1) и y_i по следующим рекуррентным формулам

$$y_1 = \frac{b_1}{s_{11}}, \quad y_i = \frac{b_i - \sum_{k=1}^{i-1} s_{ki} y_k}{s_{ii}}, \quad i = \overline{2, n}.$$

Обратный ход состоит в вычислении x_i по следующим формулам

$$x_n = \frac{y_n}{s_{nn}}, \quad x_i = \frac{y_i - \sum_{k=i+1}^{n} s_{ik} x_k}{s_{ii}}, \quad i = \overline{n-1,\ 1}.$$

Определитель матрицы:

$\det A = \prod_{i=1}^{n} d_{ii} {s_{ii}}^2$

Примечание:

Так как заданная матрица A не является симметрической, домножим исходную систему $Ax = b$ слева на A^T и обозначим $\widetilde{A} = A^T A$, $\widetilde{b} = A^T b$. Тогда получим систему вида $\widetilde{A}x = \widetilde{b}$, где $\widetilde{A} \in R_{n \times n}$, $\widetilde{A} = \widetilde{A}^T$. Решая эту систему, получим решение x, которое также будет решением исходной системы.

Листинг программы

```
import numpy as np
from math import fabs, sqrt


def sign(a):
    if a > 0:
        return 1
    elif a < 0:
        return -1
    else:
        return 0


def sqrt_system_solve(A, b):
    n = A.shape[0]
    s = np.zeros(A.shape)
    d = np.zeros(A.shape)
    a = np.array(np.hstack((A, b)))
    print("Исходная матрица системы:")
    print(a)
    a = np.dot(np.transpose(A), a)
    print("Матрица системы после умножения на A^T: ")
    print(a)
    s[0][0] = sqrt(a[0][0])
    det = s[0][0]
    d[0][0] = sign(a[0][0])
    for j in range(1, n):
        s[0][j] = a[0][j] / s[0][0]
    for i in range(1, n):
        s[i][i] = a[i][i]
        d[i][i] = a[i][i]
        for k in range(i):
            s[i][i] -= (s[k][i] ** 2 * d[k][k])
            d[i][i] -= (s[k][i] ** 2 * d[k][k])
        s[i][i] = sqrt(s[i][i])
        det *= s[i][i]
        d[i][i] = sign(d[i][i])
        for j in range(i + 1, n):
            s[i][j] = a[i][j]
            for k in range(i):
                s[i][j] -= s[k][i] * s[k][j] * d[k][k]
            s[i][j] /= s[i][i]
    print("Определитель матрицы A:")
    print(det)
    print("Матрица S:")
    print(s)
    print("Матрица D:")
    print(d)
    print("S^T * D * S")
    print(np.dot(np.transpose(s), np.dot(d, s)))
    y = np.zeros(b.shape)
    g = np.dot(np.transpose(s), d)
    y[0][0] = a[0][n] / s[0][0]
    for i in range(1, n):
```

```
        y[i][0] = a[i][n]
        for k in range(i):
            y[i][0] -= g[i][k] * y[k][0]
        y[i][0] /= s[i][i]
    print("Вектор y:")
    print(y.tolist())
    x = np.zeros(b.shape)
    x[n - 1][0] = y[n - 1][0] / s[n - 1][n - 1]
    for i in reversed(range(n - 1)):
        x[i][0] = y[i][0]
        for k in range(i + 1, n):
            x[i][0] -= s[i][k] * x[k][0]
        x[i][0] /= s[i][i]
    print("Вектор x:")
    print(x.tolist())
    print("Невязка:")
    print((np.dot(A, x) - b).tolist())


A = np.array([
    [0.6444, 0.0000, -0.1683, 0.1184, 0.1973],
    [-0.0395, 0.4208, 0.0000, -0.0802, 0.0263],
    [0.0132, -0.1184, 0.7627, 0.0145, 0.0460],
    [0.0395, 0.0000, -0.0960, 0.7627, 0.0000],
    [0.0263, -0.0395, 0.1907, -0.0158, 0.5523]
])
b = np.array([
    [1.2677],
    [1.6819],
    [-2.3657],
    [-6.5369],
    [2.8351]
])

sqrt_system_solve(A, b)
```

Входные данные

A					b
0,6444	0,0000	-0,1683	0,1184	0,1973	1.2677
-0,0395	0,4208	0,0000	-0,0802	0,0263	1.6819
0,0132	-0,1184	0,7627	0,0145	0,0460	-2.3657
0,0395	0,0000	-0,0960	0,7627	0,0000	-6.5369
0,0263	-0,0395	0,1907	-0,0158	0,5523	2.8351

Вывод программы

Исходная матрица системы:

```
 0.6444  0.       -0.1683  0.1184  0.1973  1.2677
-0.0395  0.4208   0.       -0.0802  0.0263  1.6819
 0.0132 -0.1184   0.7627   0.0145  0.046  -2.3657
 0.0395  0.       -0.096   0.7627  0.      -6.5369
 0.0263 -0.0395   0.1907  -0.0158  0.5523  2.8351
```

Матрица системы после умножения на A^T:

```
 0.41923779 -0.01922333 -0.09716147  0.10936737  0.14123396  0.53559917
-0.01922333  0.19265145 -0.09783633 -0.03484086 -0.01619521  0.87585595
-0.09716147 -0.09783633  0.65561867 -0.08509983  0.10720222 -0.84947733
 0.10936737 -0.03484086 -0.08509983  0.60262178  0.01319172 -5.04958356
 0.14123396 -0.01619521  0.10720222  0.01319172  0.34677027  1.75135471
```

Матрица S:

```
0.64748575 -0.02968919 -0.15005963  0.16891085  0.21812675
0.          0.43791552 -0.23358724 -0.0681091  -0.02219424
0.          0.          0.76061671 -0.09947526  0.1771588
0.          0.          0.          0.74803524 -0.01008103
0.          0.          0.          0.          0.51692508
```

Матрица D:

```
1. 0. 0. 0. 0.
0. 1. 0. 0. 0.
0. 0. 1. 0. 0.
0. 0. 0. 1. 0.
0. 0. 0. 0. 1.
```

Вектор y:

[0.8271983964466707, 2.056138147216592, -0.3221867195232914, -6.7928807857936215, 3.105196626662377]

Вектор x:

[0.9982150476244757, 1.9998652821237983, -2.999759708348939, -9.000008425832783, 6.007053532763625]

Определитель матрицы A:

0.08339424331325143

Вывод

Метод квадратного корня является одним из прямых методов решения систем линейных алгебраических уравнений. Данный метод сокращает вычисления примерно в 2 раза за счет симметрии, является более точным, чем метод Гаусса.

Метод квадратного корня дает выигрыш во времени в силу того, что:

1) существенно уменьшает число операций умножения и деления. По данному методу требуется произвести операций извлечения квадратного корня и $\approx \frac{n^3}{6}$ операций умножения и деления (для больших n это почти в два раза меньше, чем в методе исключения Гаусса);

2) при расчетах на ЭВМ позволяет экономить память машины, т.к. матрица A – симметрическая. В вычислительной схеме можно записывать $\frac{n}{2}$(n+1) верхних коэффициентов n.