Московский государственный технический университет им. Н.Э. Баумана
Факультет «Радиоэлектроника и лазерная техника (РЛ)»
Кафедра «Технология приборостроения (РЛ6)»

Лабораторная работа №3
"Итерационные методы решения СЛАУ."
по дисциплине "Численные методы"
Вариант №1

Выполнили студенты группы РЛ6-71
Филимонов С.В.

Преподаватель Чигирева О.Ю.

Москва, 2023

Исходные данные:

1. Хорошо обусловленная матрица:

$$(A,b)=\left(\begin{array}{cccc|c} 31.2000 & -1.3200 & -7.6800 & 4.0900 & -83.3200 \\ 7.2300 & -126.0000 & 7.1400 & 3.0400 & 38.9000 \\ 9.4900 & 6.4000 & 6.0000 & 8.4500 & -56.7000 \\ 2.6800 & -3.2900 & 0.2800 & 13.4000 & -504.0900 \end{array}\right)$$

Вектор точного решения: $x=\begin{pmatrix} 10 \\ 1 \\ 30 \\ -40 \end{pmatrix}$

2. Плохо обусловленная матрица:

$$(A,b)=\left(\begin{array}{cccc|c} 16.3820 & -2.0490 & -41.8290 & 16.3920 & 33.6130 \\ 307.6480 & -38.4660 & -840.3660 & 312.5280 & 710.3420 \\ 0.4560 & -0.0570 & -1.1770 & 0.4560 & 0.9490 \\ 23.2720 & -2.9090 & -66.3090 & 23.8720 & 57.6730 \end{array}\right)$$

Вектор точного решения: $x=\begin{pmatrix} 2 \\ 60 \\ -1 \\ 5 \end{pmatrix}$

Цель работы: изучение методов Якоби и Зейделя решения СЛАУ с невырожденной матрицей; сравнение точности и скорости их работы.

Содержание работы:

1. Реализовать методы Якоби и Зейделя (в программе предусмотреть проверку достаточного условия сходимости).

2. Провести решение заданных систем линейных алгебраических уравнений методами Гаусса, Якоби и Зейделя. Результаты представить в виде таблицы.

Алгоритм метода Якоби:

Рассмотрим СЛАУ Ax = b, detA не равно 0.

Приведем систему к виду, удобному для итераций. Для этого выразим неизвестное x_i из i-ого уравнения:

$$x_i = \frac{1}{a_{ii}} \cdot (\beta_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ii} \cdot x_j), a_{ii} \neq 0$$

В результате получим систему:

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} 0 & \beta_{12} & \cdots & \beta_{1n} \\ \beta_{21} & 0 & \cdots & \beta_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{n1} & \beta_{n2} & \cdots & 0 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix},$$

Где

$$\beta_{ij} = \begin{cases} -\dfrac{a_{ij}}{a_{ii}}, j \neq i \\ 0, j = i \end{cases}, i, j = \overline{1, n}$$

$$c_i = \frac{\beta_i}{a_{ii}}, i = \overline{1, n}.$$

Обозначим: $B = (\beta_{ij}) \in M_{nxn}(R)$, $c \in (c_i) \in M_{nx1}(R)$ и запишем систему в матричной форме $x = Bx + C$. Выберем начальное приближение $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})^T$ и подставим его в правую часть системы $x = Bx + C$. Вычисляя полученное выражение, находим 1-е приближение:

$$x^{(1)} = Bx^{(0)} + C \text{ и тд.}$$

$$x^{(k+1)} = Bx^{(k)} + C, k = 0,1,\ldots$$

Расчетная фомула Якоби.

Теорема о достаточном условии сходимости

Пусть $\|B\| < 1$, тогда

1) Решение $\bar{x}$ системы, $x = Bx + C$ существует и единственно;

2) При любом начальном приближение $x^{(0)}$ метод Якоби сходится и справедлива оценка погрешности:

$$\left\|x^{(n)} - \bar{x}\right\| \leq \|B\|^n \cdot \left\|x^{(0)} - \bar{x}\right\|.$$

Из этого следует, что метод Якоби сходится, если $\|B\| < 1$.

Апостериорная оценка погрешности:

Если $\|B\| < 1$, то

$$\left\|x^{(n)} - \bar{x}\right\| \leq \frac{\|B\|}{1 - \|B\|} \cdot \left\|x^{(n)} - x^{(n-1)}\right\|.$$

Критерий окончания интерационого процесса:

Пусть требуется найти решение с точность ε, то есть:

$$\left\|x^{(n)} - \bar{x}\right\| \leq \varepsilon$$

Тогда из апрстериорной оценки погрещности получается, что вычисления следует вести до выполнения неравенства:

$$\left\|x^{(n)} - x^{(n-1)}\right\| \leq \varepsilon_1, \text{ где } \varepsilon_1 = \frac{1 - \|B\|}{\|B\|}\varepsilon.$$

Алгоритм метода Зейделя:

Метод представляет собой модификацию метода Якоби. Представим матрицу B в виде суммы нижней (B1) и верхней (B2) треугольных матриц: B = B1 = B2. Тогда:

$$x^{(k+1)} = B_1 x^{(k+1)} + B_2 x^{(k)} + c, k = 0, 1, \dots$$

Расчетная формула метода Зейделя.

Апостериорная оценка погрешности:

Если $\|B\| < 1$, то

$$\left\|x^{(n)} - \bar{x}\right\| \leq \frac{\|B_2\|}{1 - \|B\|} \cdot \left\|x^{(n)} - x^{(n-1)}\right\|.$$

Критерий окончания интерационого процесса:

Пусть требуется найти решение с точность ε, то есть:

$$\left\|x^{(n)} - \bar{x}\right\| \leq \varepsilon$$

Тогда из апрстериорной оценки погрещности получается, что вычисления следует вести до выполнения неравенства:

$$\left\|x^{(n)} - x^{(n-1)}\right\| \leq \varepsilon_2, \text{где } \varepsilon_2 = \frac{1 - \|B\|}{\|B_2\|}\varepsilon.$$

Результаты расчетов:

Метод	Приближенное решение	Относительнавя погрешность	Число итераций	Время работы, с
Якоби	$\begin{pmatrix} 9.999999995774 \\ 0.999999880695 \\ 30.000000006139 \\ -39.999999999119 \end{pmatrix}$	$1.780797731498 \cdot 10^{-13}$	136	$2.9 \cdot 10^{-5}$
Зейделя	$\begin{pmatrix} 10.000000000003 \\ 1.00000000001 \\ 29.999999999836 \\ -39.999999999119 \end{pmatrix}$	$1.538769112130 \cdot 10^{-13}$	86	$6.3 \cdot 10^{-5}$
Гаусса	$\begin{pmatrix} 10.00000000002 \\ 1.00000000003 \\ 30.00000000034 \\ -40.00000000696 \end{pmatrix}$	$1.776356833940 \cdot 10^{-15}$	–	$5 \cdot 10^{-6}$

(Решение получно для хорошо обусловленной матрицы)

Вывод:

1. В конкретной программной реализации метод Гаусса обладает наибольшим быстродействием, а метод Зейделя - наименьшим. Однако делать вывод о быстродействии на основании решения единственной СЛАУ нельзя. Это же

подтверждается многократным запуском программы, приводящим к иному распределению "позиций" по быстродействию.

2. Самый точный - метод Гаусса.

3. Больше всего итераций из двух итерационных методов - метод Якоби.

Пояснения:

Для правильной работы метода Якоби необходимо проверять достаточное условие сходимости ||B|| < 1. Однако, если ||B|| ≈ 1, то применение критерия $\|x^n - x^{n-1}\| < \varepsilon_1$ приведет к преждевременному окончанию итерационного процесса, так как эта величина будет малой вследствие медленной сходимости метода, а не хорошего приближения x^n и x^{n-1} к решению. Поэтому, исходя из этого, можно сделать вывод, что метод Якоби можно применять, если условие сходимости выполняется. Аналогично для метода Зейделя.

В нашем случае для метода Якоби достаточное условие сходимости не выполнено, ||B|| > 1. Для того, чтобы узнать сходится ли метод, проверялось необходимое и достаточное условие сходимости. В результате был сделан вывод, что метод сходится.

Однако это не позволяет сделать оценку скорости сходимости и выбрать критерий окончания итерационного процесса. Поэтому был выбран следующий критерий окончания:

$$\left\|x^{(n)} - x^{(n-1)}\right\| \leq \varepsilon$$

Аналогично для метода Зейделя.

Текст программы:

```
static Matrix getSleMethodJacobiBetaM(const Matrix& AB) {
    Matrix beta(AB.getRows(), (AB.getCols() - 1));
    for (size_t i = 0; i < beta.getRows(); ++i)
        for (size_t j = 0; j < beta.getCols(); ++j)
            if (j != i)
                beta(i, j) = ((-AB(i, j)) / AB(i, i));
            else
                beta(i, j) = 0;
    return beta;
}

static Matrix getSleMethodJacobiCeM(const Matrix& AB) {
    Matrix ce(AB.getRows(), 1);
    const size_t last_elemen_j = (AB.getCols() - 1);
    for (size_t i = 0; i < ce.getRows(); ++i)
        ce(i, 0) = (AB(i, last_elemen_j) / AB(i, i));
    return ce;
}

static double calcMethodJacobiAbsX(const Matrix& x1, const Matrix& x2) {
    const Matrix x = (x1 - x2);
    double x_det = 1;
    for (size_t i = 0; i < x.getRows(); ++i)
        x_det *= x(i, 0);
    return std::abs(x_det);
}

static Matrix enumerationMethodJacobi(const Matrix& beta, const Matrix& ce, const double& eps1) {
    Matrix x1(beta.getRows(), 1), x2(beta.getRows(), 1);
    size_t i = 0;
    double x = (eps1 + 1.0);
    for (i = 0; x > eps1; ++i) {
        if ((i % 2) == 0) {
            x2 = ((beta * x1) + ce);
            x = calcMethodJacobiAbsX(x2, x1);
        } else {
            x1 = ((beta * x2) + ce);
            x = calcMethodJacobiAbsX(x1, x2);
        }
    }
     std::cout << i << std::endl;
    return ((i % 2) == 0) ? x2 : x1;
}
```

```
Matrix SLEmethodJacobi(const Matrix& A, const Matrix& B, const double& eps) {
    if (A.det() == 0)
        throw SingularMatrix();
        if ((A.rows != B.rows) || (B.cols > 1) || (A.isNull()) || (B.isNull()))
        throw DimensionMismatch(A, B);
    const Matrix AB = A.getExtendedMatrixOfTheSystem(B);
    const Matrix beta = getSleMethodJacobiBetaM(AB);
    const double Bdet = std::abs(beta.det());
    if (Bdet >= 1.0)
        throw MethodJacobiBdetMore1();
    const Matrix ce = getSleMethodJacobiCeM(AB);
    const double eps1 = (((1.0 - Bdet) / Bdet) * eps);
    return enumerationMethodJacobi(beta, ce, eps1);
}

static Matrix getMatrixB1(const Matrix& beta) {
    Matrix B1(beta.getRows(), beta.getCols());
    for (size_t i = 0; i < beta.getRows(); ++i) {
        for (size_t j = 0; j < i; ++j) {
            B1(i, j) = beta(i, j);
        }
    }
    return B1;
}

static Matrix enumerationMethodSeidel(const Matrix& B1, const Matrix& B2, const Matrix& ce, const double& eps2) {
    Matrix x1(B1.getRows(), 1), x2(B1.getRows(), 1);
    x2 = ((B2 * x1) + ce);
    size_t i = 0;
    double x = (eps2 + 1.0);
    for (i = 0; x > eps2; ++i) {
        if ((i % 2) == 0) {
            x2 = ((B1 * x2) + (B2 * x1) + ce);
            x = calcMethodJacobiAbsX(x2, x1);
        } else {
            x1 = ((B1 * x1) + (B2 * x2) + ce);
            x = calcMethodJacobiAbsX(x1, x2);
        }
    }
     std::cout << i << std::endl;
    return ((i % 2) == 0) ? x2 : x1;
}
Matrix SLEmethodSeidel(const Matrix& A, const Matrix& B, const double& eps) {
    if (A.det() == 0)
        throw SingularMatrix();
        if ((A.rows != B.rows) || (B.cols > 1) || (A.isNull()) || (B.isNull()))
```

```
        throw DimensionMismatch(A, B);

    const Matrix AB = A.getExtendedMatrixOfTheSystem(B);
    const Matrix beta = getSleMethodJacobiBetaM(AB);
    const double Bdet = std::abs(beta.det());
    if (Bdet >= 1.0)
        throw MethodJacobiBdetMore1();
    const Matrix B1 = getMatrixB1(beta);
    const Matrix B2 = (beta - B1);
    const Matrix ce = getSleMethodJacobiCeM(AB);
    const double eps2 = (((1.0 - Bdet) / Bdet) * eps);
    return enumerationMethodSeidel(B1, B2, ce, eps2);
}

double norm1(const Matrix& A) {
    double sum = 0.0;
    for (size_t i = 0; i < A.getRows(); ++i)
        sum += std::abs(A(i, 0));
    return sum;
}

double calcR(const Matrix& X, const Matrix& X_) {
    const Matrix deltaX = (X - X_);
    return norm1(deltaX);
}

int main() {
    const Matrix A = utils::readFromFile("./project/matrix/data/math_high_operation/Gausse/case_0/A.txt");
    const Matrix B = utils::readFromFile("./project/matrix/data/math_high_operation/Gausse/case_0/B.txt");
    const Matrix X = utils::readFromFile("./project/matrix/data/math_high_operation/Gausse/case_0/X.txt");

    double start = STSL::getCPUTime();
    Matrix x_gausse = SLEmethodGauss(A, B);
    double end = STSL::getCPUTime();
    std::cout << "Gausse time: " << (end - start) << std::endl;

    start = STSL::getCPUTime();
    Matrix x_jacobi = SLEmethodJacobi(A, B);
    end = STSL::getCPUTime();
    std::cout << "Jacobi time: " << (end - start) << std::endl;

    start = STSL::getCPUTime();
    Matrix x_zeidel = SLEmethodSeidel(A, B);
    end = STSL::getCPUTime();
    std::cout << "Zeidel time: " << (end - start) << std::endl;
```

```
    // Вывод матриц
    std::cout << x_gausse << x_jacobi << x_zeidel;

    // Расчет погрешностей
    std::cout << "Якоби: ||r0||1 = " << calcR(X, x_jacobi) << std::endl;
    std::cout << "Зейдель: ||r0||1 = " << calcR(X, x_zeidel) << std::endl;
    std::cout << "Гаусс: ||r0||1 = " << calcR(X, x_gausse) << std::endl;
    return 0;
}
```