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:

Вектор точного решения:

2. Плохо обусловленная матрица:

Вектор точного решения:

**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 из i-ого уравнения:

В результате получим систему:

Где

Обозначим: и запишем систему в матричной форме x = Bx + C. Выберем начальное приближение и подставим его в правую часть системы x = Bx + C. Вычисляя полученное выражение, находим 1-е приближение:

Расчетная фомула Якоби.

*Теорема о достаточном условии сходимости*

Пусть , тогда

1. Решение системы, x = Bx + C существует и единственно;
2. При любом начальном приближение метод Якоби сходится и справедлива оценка погрешности:

Из этого следует, что метод Якоби сходится, если .

*Апостериорная оценка погрешности:*

Если , то

*Критерий окончания интерационого процесса:*

Пусть требуется найти решение с точность , то есть:

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

**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:

Расчетная формула метода Зейделя.

*Апостериорная оценка погрешности:*

Если , то

*Критерий окончания интерационого процесса:*

Пусть требуется найти решение с точность , то есть:

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

**Результаты расчетов:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метод | Приближенное решение | Относительнавя погрешность | Число итераций | Время работы, с |
| Якоби |  |  | 136 |  |
| Зейделя |  |  | 86 |  |
| Гаусса |  |  |  |  |

(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 приведет к преждевременному окончанию итерационного процесса, так как эта величина будет малой вследствие медленной сходимости метода, а не хорошего приближения и 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:

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;

}