Лабораторная работа №3

Тужилкина Н.Г., БПМ-18-2, вариант 24.

# Прямые методы решения СЛАУ

Пусть дана система линейных алгебраических уравнений (СЛАУ), записываемая в виде

* ‒ квадратная матрица ,
* ‒ заданный вектор ,
* ‒ неизвестный вектор .

Требуется найти неизвестный вектор , который путем его подстановки в (1) превращает уравнение в верное равенство.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  |  |  |  |
| 2.8745 | 6.35792 | 5.66149 | -4.21491 | -0.575884 |  |  | = | 2.12867 |
| -3.81329 | 8.3874 | -4.00311 | 8.53389 | -9.16745 | -3.81024 |
| -2.59194 | 9.72777 | -8.67855 | 1.45604 | 4.5732 | 9.84924 |
| -5.41612 | -4.64095 | 8.08344 | -5.21043 | -0.0540179 | -1.08676 |
| -2.08411 | 5.56688 | 1.51708 | 8.31599 | -0.0991851 | -8.5815 |

Требование, накладываемое на матрицу : система (1) должна иметь единственное решение при любом векторе . Необходимым и достаточным условием этого является невырожденность матрицы , т. е. .

## Цель работы:

1. Изучить прямые методы решения СЛАУ:
   1. Метод -разложения
   2. Метод вращений
   3. Метод решения трехдиагональных систем
2. Написать и отладить программу расчёта корней СЛАУ.
3. Выполнить проверку, т. е. умножить матрицу на найденный вектор , сравнить результат с заданным вектором и показать отклонение.

## Метод LU-разложения

### Изложение метода

Метод опирается на возможность представления матрицы системы в форме произведения двух треугольных матриц:

‒ нижняя, а ‒ верхняя треугольные матрицы, имеющие вид:

С учетом (2) система представляется в форме

Решение системы (3) сводится к последовательному решению двух простых систем с треугольными матрицами.

Прямой ход. Пусть . В результате решения системы

находится вектор .

Обратный ход. В результате решения системы

находится решение задачи ‒ вектор .

Решения обеих систем находятся рекуррентно.

Формулы для определения элементов матриц :

### Описание алгоритма

1. Выполнить операцию факторизации исходной матрицы , применяя формулы (4.1) и (4.2), и получить матрицы и .
2. Решить систему
3. Решить систему

### Текст программы

// 1. Операция факторизации исходной матрицы

matrix matrix::make\_LU() const {

matrix LU{ \*this };

for (int i(1); i <= n; ++i)

for (int j(1); j <= n; ++j) {

if ((i == 1) && (j == 1)) continue;

// Формула 4.1

if (i >= j)

for (int s(1); s < j; ++s)

LU.at(i, j) -= LU.at(i, s) \* LU.at(s, j);

// Формула 4.2

else if (i < j) {

for (int s(1); s < i; ++s)

LU.at(i, j) -= LU.at(i, s) \* LU.at(s, j);

LU.at(i, j) /= LU.at(i, i);

}

}

return LU;

}

std::vector<double> matrix::LU\_method(const std::vector<double>& b) const {

matrix LU{ make\_LU() };

std::vector<double> x{ b };

// 2. Решение системы Ly=b

for (int i(1); i <= n; ++i) {

for (int j(1); j < i; ++j)

x[i - 1] -= LU.at(i, j) \* x[j - 1];

x[i - 1] /= LU.at(i, i);

}

// 3. Решение системы Ux=y

for (int i(n); i >= 1; --i) {

for (int j(n); j > i; --j) {

x[i - 1] -= LU.at(i, j) \* x[j - 1];

}

}

check(x, b);

return x;

}

std::vector<double> matrix::check(const std::vector<double>& x, const std::vector<double>& b) const {

std::vector<double> b0{ x };

for (auto& it : b0) it = 0;

// Умножение исходной матрицы на найденный вектор x

for (int i(1); i <= n; ++i) {

for (int j(1); j <= n; ++j) {

b0[i - 1] += at(i, j) \* x[j - 1];

}

}

// Сравнение результатов

for (int i(0); i < b0.size(); ++i) {

b0[i] -= b[i];

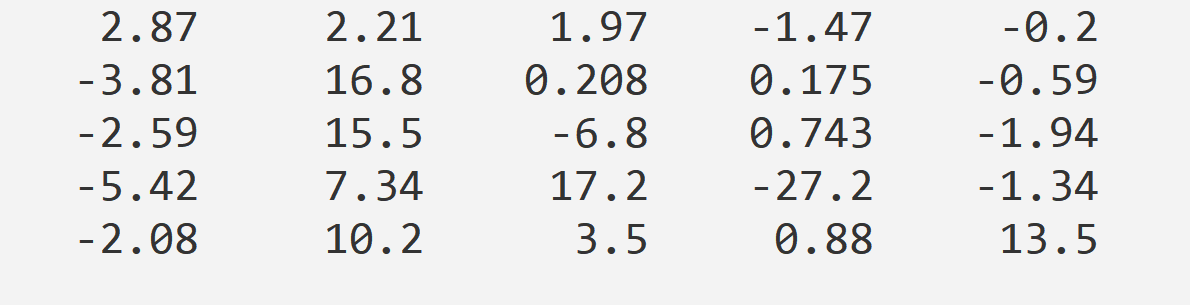
}

return b0;

}

### Представление результатов

Получившаяся матрица имеет вид:

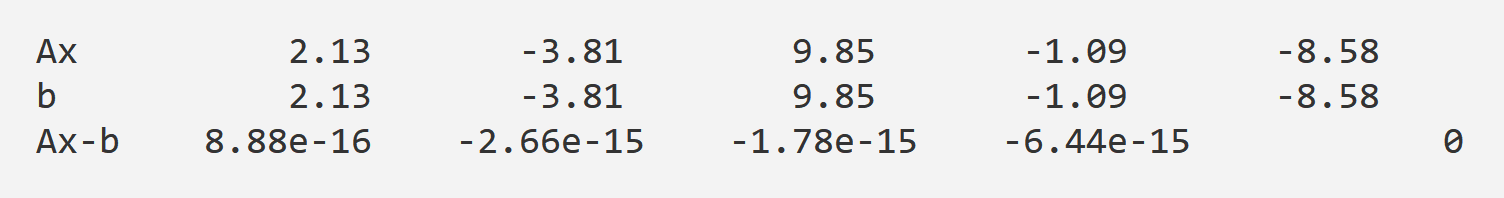


Решение системы :

Решение системы:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  |  |  |  |
| 2.8745 | 6.35792 | 5.66149 | -4.21491 | -0.575884 |  | -0.194 | = | 2.12867 |
| -3.81329 | 8.3874 | -4.00311 | 8.53389 | -9.16745 | 0.371 | -3.81024 |
| -2.59194 | 9.72777 | -8.67855 | 1.45604 | 4.5732 | -0.811 | 9.84924 |
| -5.41612 | -4.64095 | 8.08344 | -5.21043 | -0.0540179 | -1.18 | -1.08676 |
| -2.08411 | 5.56688 | 1.51708 | 8.31599 | -0.0991851 | 0.0915 | -8.5815 |

Подстановка найденного решения и нахождение отклонения:



Получившаяся разница объясняется особенностями представления вещественных чисел в ЭВМ.

## Метод вращений

### Изложение метода

Как и в методе Гаусса, целью прямого хода преобразований в методе вращений — приведение системы линейных уравнений к треугольному виду последовательным обнулением поддиагональных элементов сначала первого столбца, затем второго и т. д.

Пусть и — некоторые отличные от нуля числа. Умножим первое уравнение системы на , второе на и сложим их; полученным уравнением заменим первое уравнение системы. Затем первое уравнение исходной системы умножаем на – , второе – на и результатом их сложения заменяем второе уравнение. Таким образом, первые два уравнения системы заменяются уравнениями:

На введенные два параметра накладываются два условия:

— условие обнуление (т. е. исключение из второго уравнения) и

— условие нормировки.

и , удовлетворяющие этим условиям можно принять соответственно

После фиксирования и система принимает вид

где

Далее первое уравнение системы (6) заменяется новым, полученным сложение результатов умножения первого и третьего уравнений (6) на

а третье ‒ уравнением, полученным сложением результатов умножения тех же уравнений соответственно на . Получаем систему

где

Проделав такие преобразования раз, придем к системе

Длина любого вектора-столбца расширенной матрицы системы (7) остается такой же как у соответствующего столбца исходной системы (1).

Дальше точно так же за промежуточных шага преобразуем подсистему

системы (7), создавая нули под элементом , и т.д.

В результате таких этапов прямого хода исходная система будет приведена к треугольному виду

Такая система решается рекуррентно.

### Описание алгоритма

#### Промежуточный шаг:

Повторять для всех :

Вычислить и по формулам

Вычислить новые коэффициенты матрицы:

1. Преобразовать 1-е уравнение системы, выполняя промежуточный шаг.
2. Составить новую систему, удалив из исходной преобразованное 1-е уравнение.
3. Перейти к п. 1 для новой системы. Выполнять, пока не преобразуются все уравнения заданной системы.
4. Решить систему, составленную из преобразованных уравнений, рекуррентно.

### Текст программы

// Промежуточный шаг

void matrix::JR\_step(std::vector<double>& b) {

for (int k(1); k <= n\_rows - 1; ++k) {

double c = at(1, 1) / sqrt(at(1, 1) \* at(1, 1) + at(k + 1, 1) \* at(k + 1, 1));

double s = at(k + 1, 1) / sqrt(at(1, 1) \* at(1, 1) + at(k + 1, 1) \* at(k + 1, 1));

for (int j(1); j <= n\_cols; ++j) {

matrix temp{ \*this };

at(k + 1, j) = -s \* temp.at(1, j) + c \* temp.at(k + 1, j);

at(1, j) = c \* temp.at(1, j) + s \* temp.at(k + 1, j);

}

std::vector<double> temp{ b };

b[1 - 1] = c \* temp[1 - 1] + s \* temp[k + 1 - 1];

b[k + 1 - 1] = -s \* temp[1 - 1] + c \* temp[k + 1 - 1];

}

}

// Удаление 1-й строки и 1-го столбца

void matrix::cut\_matrix() {

std::vector<double> cut\_data;

for (int i(2); i <= n\_rows; ++i)

for (int j(2); j <= n\_cols; ++j)

cut\_data.push\_back(at(i,j));

n\_rows -= 1;

n\_cols -= 1;

data = cut\_data;

}

std::vector<double> matrix::JR\_method(const std::vector<double>& b) const {

matrix work{ \*this };

matrix result{ \*this };

for (auto& it : result.data)

it = 0;

std::vector<double> work\_b{ b };

std::vector<double> result\_b{ b };

for (auto& it : result\_b)

it = 0;

for (int i(1); i <= 5; ++i) {

// Преобразование i-го уравнения

work.JR\_step(work\_b);

// Сохранение получившегося уравнения в измененную матрицу

int k(1);

for (int j(i); j <= n\_cols; ++j) {

result.at(i, j) = work.at(1,k);

k += 1;

}

result\_b[i-1] = work\_b[0];

// Получение новой системы для обработки следующего уравнения

work.cut\_matrix();

for (int j(1); j < work\_b.size(); ++j)

work\_b[j - 1] = work\_b[j];

work\_b.pop\_back();

}

// Решение преобразованной системы

std::vector<double> x{ result\_b };

for (int i(n\_rows); i >= 1; --i) {

for (int j(n\_rows); j > i; --j) {

x[i - 1] -= result.at(i, j) \* x[j - 1];

}

x[i - 1] /= result.at(i, i);

}

check(x, b);

return x;

std::vector<double> matrix::check(const std::vector<double>& x, const std::vector<double>& b) const {

std::vector<double> b0{ x };

for (auto& it : b0) it = 0;

// Умножение исходной матрицы на найденный вектор x

for (int i(1); i <= n\_rows; ++i) {

for (int j(1); j <= n\_rows; ++j) {

b0[i - 1] += at(i, j) \* x[j - 1];

}

}

// Сравнение результатов

for (int i(0); i < b0.size(); ++i) {

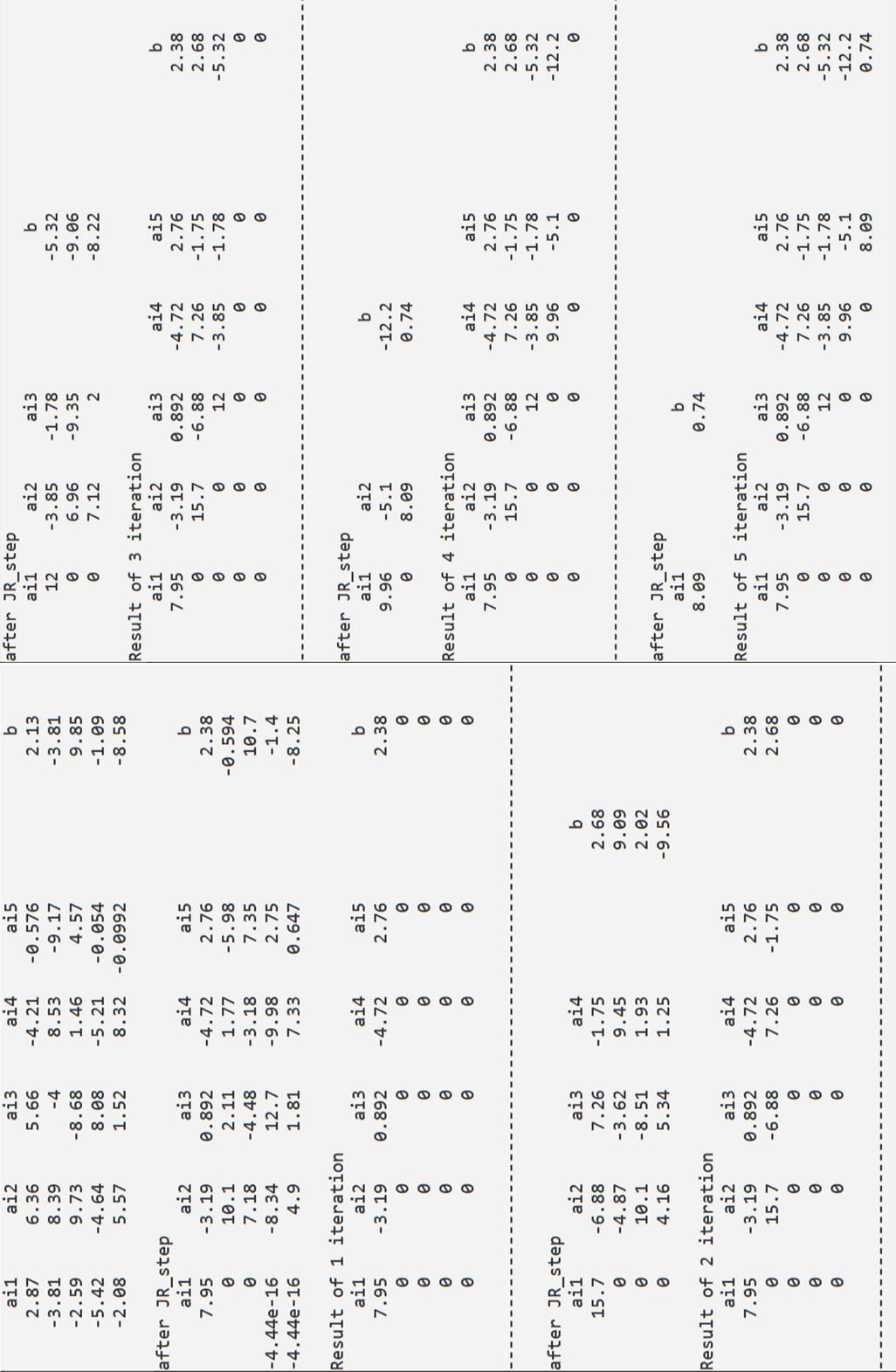
b0[i] -= b[i];

}

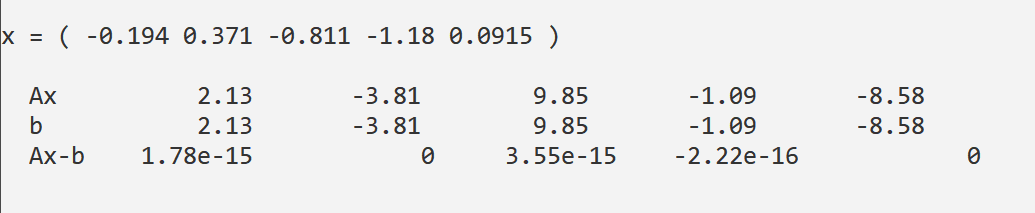
return b0;

}

### Представление результатов



Найденное решение и отклонение:

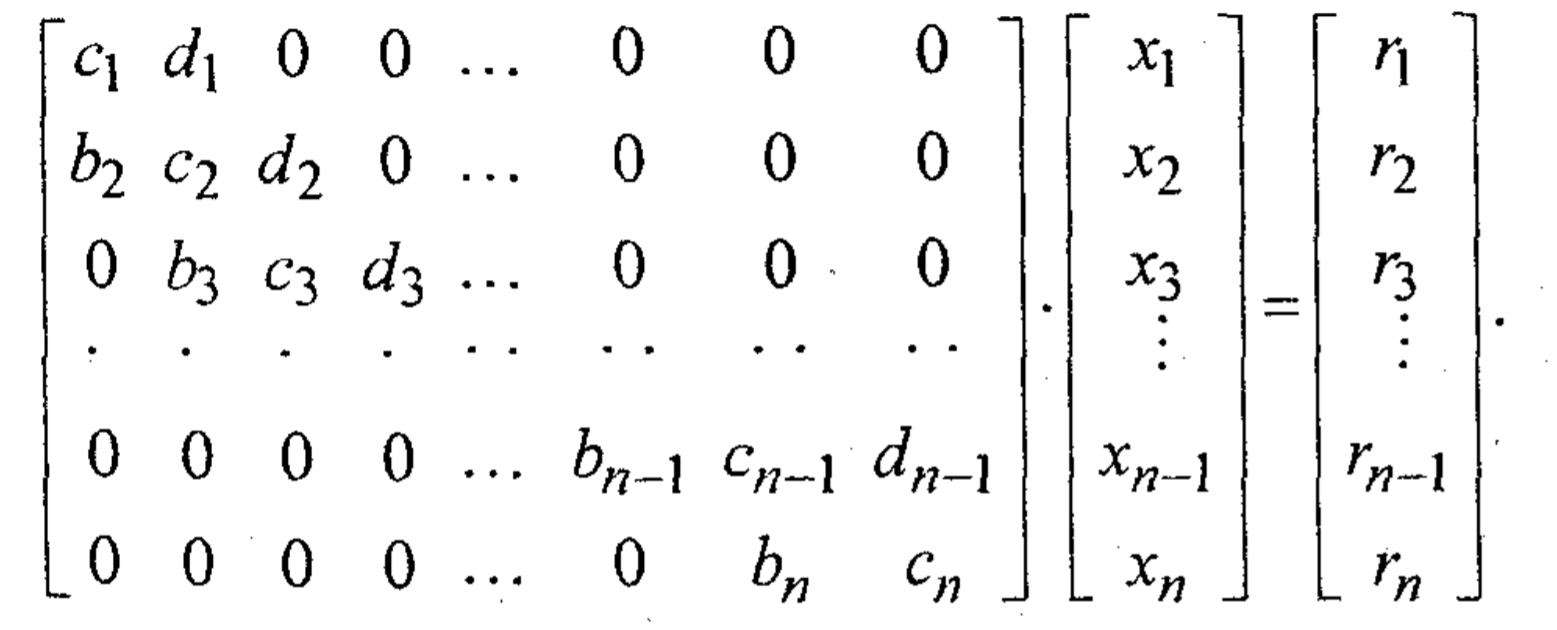


Величина отклонения в целом соответствует отклонению в -методе.

## Метод прогонки решения систем с трехдиагональными матрицами коэффициентов

### Изложение метода

Дана система линейных алгебраических уравнений с трехдиагональной матрицей :



С целью избавиться от ненулевых элементов в поддиагональной части матрицы системы, рассмотрим такие наборы чисел и , при которых

Уменьшим в (9) индекс на единицу, и подставим полученное выражение в (8), откуда получим

Представление (9) будет иметь место, если при всех выполняются рекуррентные соотношения

Процесс вычисления может быть начат со значений

и продолжен далее по формулам (10) последовательно при .

Полагая в (9) , будем иметь

где ‒ уже известные с предыдущего шага числа. Далее по формулам (9) последовательно находятся .

### Описание алгоритма

1. Вычислить
2. Прямая прогонка. Найти прогоночные коэффициенты по формулам (10) при .
3. Найти
4. Обратная прогонка. Получить неизвестные по формулам (9) при .

### Текст программы

std::vector<double> matrix::TD\_method(const std::vector<double>& b) const {

// δ\_1 = - d\_1 / c\_1

std::vector<double> delta{ -at(1, 2) / at(1, 1) };

// λ\_1 = r\_1 / c\_1

std::vector<double> lambda{ b[0] / at(1, 1) };

// Прямая прогонка

for (int i(2); i <= n\_rows - 1; ++i) {

// δ\_i = - d\_i / (c\_i + b\_i \* δ\_(i-1))

double d\_i(- at(i, i + 1) / (at(i, i) + at(i, i - 1) \* delta.back()));

// λ\_i = (r\_i - b\_i \* λ\_(i-1)) / (c\_i + b\_i \* δ\_(i-1))

double l\_i((b[i - 1] - at(i, i - 1) \* lambda.back()) / (at(i, i) + at(i, i - 1) \* delta.back()));

delta.push\_back(d\_i);

lambda.push\_back(l\_i);

}

std::vector<double> x{ b };

// x\_n = (r\_n - b\_n \* λ\_(n-1)) / (c\_n + b\_n \* δ\_(n-1))

x.back() = (b.back() - at(n\_rows, n\_cols - 1) \* lambda.back()) / (at(n\_rows, n\_cols) + at(n\_rows, n\_cols - 1) \* delta.back());

// Обратная прогонка

for (int i(n\_rows - 1); i >= 1; --i) {

x[i - 1] = delta[i - 1] \* x[i] + lambda[i - 1];

}

check(x, b);

return x;

}

std::vector<double> matrix::check(const std::vector<double>& x, const std::vector<double>& b) const {

std::vector<double> b0{ x };

for (auto& it : b0) it = 0;

// Умножение исходной матрицы на найденный вектор x

for (int i(1); i <= n\_rows; ++i) {

for (int j(1); j <= n\_rows; ++j) {

b0[i - 1] += at(i, j) \* x[j - 1];

}

}

// Сравнение результатов

for (int i(0); i < b0.size(); ++i) {

b0[i] -= b[i];

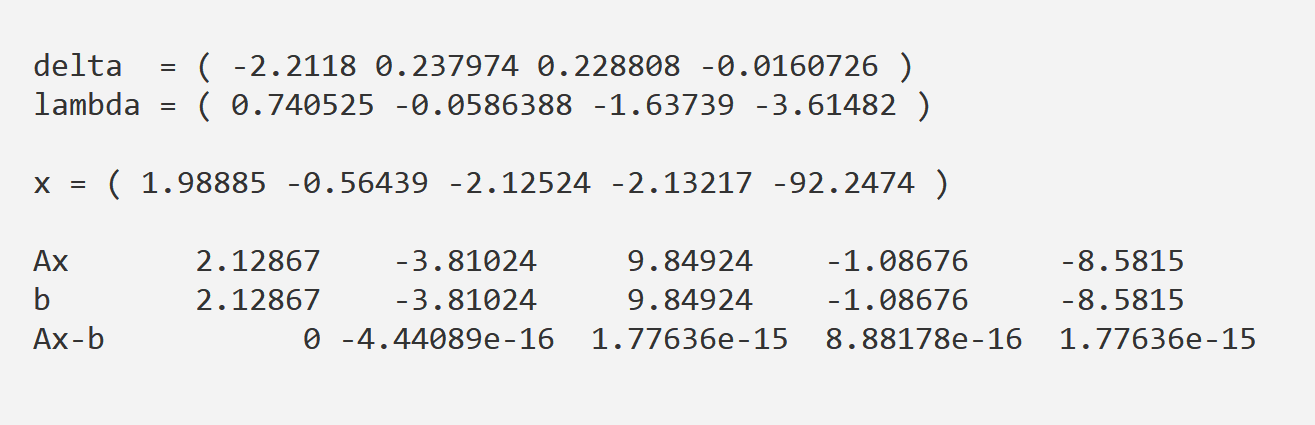
}

return b0;

}

### Представление результатов

На выводе программы представлены векторы прогоночных коэффициентов и , решение трехдиагональной системы и отклонение при его подстановке в систему.



## Вывод

Все представленные методы решения систем линейных алгебраических уравнений находят решение с точностью, определяемой вещественным типов данных. В случае double на языке C++ точность приблизительно равна .