Отчет по дисциплине: «Численные методы»

Лабораторная работа №3

«Решение СЛАУ методом прогонки»

Подготовил студент 3 курса 4 группы

Кондратович Артём

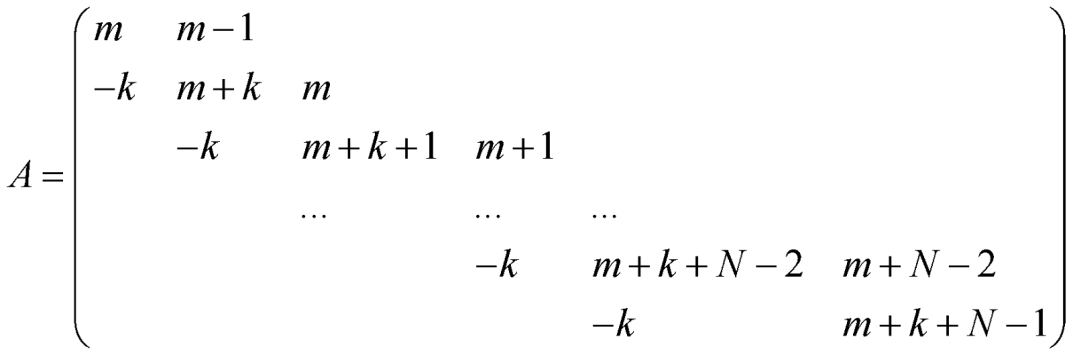
Цель работы:

Изучить основы метода Гаусса для случая трехдиагональной матрицы. Убедиться, что адаптированный на случай трехдиагональной матрицы алгоритм Гаусса выполняется многократно быстрее, чем в общем случае.

Задание:

Разработать программу численного решения методом прогонки СЛАУ *Ay=f* вида (1) в файле «Метод прогонки».

Матрицу системы задать следующим образом:

*.*

Правую часть *f* задать умножением матрицы *A* на вектор *y=*(1, 2, ... , *N*+1):  *f=Ay*.

Для вычислений выбрать параметры:

* *N*+1  – (порядок матрицы) одно из чисел в пределах от 1500 до 2000;
* *m* – номер в списке студенческой группы;
* *k* – номер студенческой группы.

Программно реализовать (в качестве языка программирования выбрать C или C++) вычисления для рассматриваемого примера. Для вычислений использовать тип float.

Листинг программы:

#include <iostream>

#include <chrono>

#include <iomanip>

#include <random>

#include <cmath>

const int N = 2000;

const int M = 4;

const int K = 4;

std::ostream& operator <<(std::ostream& output, const float\* vector)

{

for (size\_t i = 0; i < 5; i++)

{

output << vector[i] << " ";

}

output << "\n";

return output;

}

// генерация вектора А

float\* GetVectorA()

{

auto a = new float[N + 1];

for (size\_t i = 1; i < N + 1; i++)

a[i] = -K;

return a;

}

// генерация вектора В

float\* GetVectorB()

{

auto b = new float[N + 1];

for (int i = 0; i < N; i++)

{

b[i] = float(M - 1 + i);

}

return b;

}

// генерация вектора С

float\* GetVectorC()

{

auto c = new float[N + 1];

c[0] = M;

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

{

c[i] = float(M + K + i - 1);

}

return c;

}

// генерация вектора У

float\* GetVectorY()

{

auto y = new float[N + 1];

y[0] = 1;

for (size\_t i = 1; i < N + 1; i++)

{

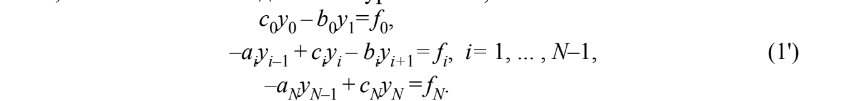
y[i] = y[i - 1] + 1;

}

return y;

}

Для подсчёта вектора f использовал формулу



float\* GetVectorF(const float\* a, const float\* b, const float\* c, const float\* y)

{

auto f = new float[N + 1];

f[0] = c[0] \* y[0] + b[0] \* y[1];

for (size\_t i = 1; i < N; i++)

{

f[i] = a[i] \* y[i - 1] + c[i] \* y[i] + b[i] \* y[i + 1];

}

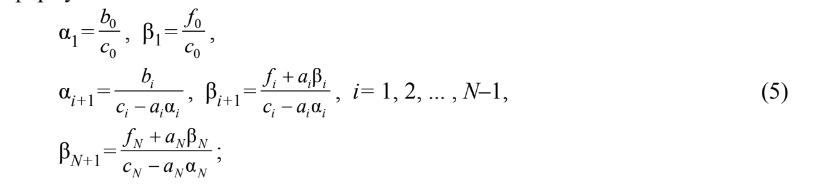
f[N] = a[N] \* y[N - 1] + c[N] \* y[N];

return f;

}

Прямая прогонка

Для подсчёта векторов alpha & beta использовал формулу



float\* GetVectorAlpha(const float\* a, const float\* b, const float\* c, const float\* f)

{

auto alpha = new float[N + 1];

alpha[1] = - b[0] / c[0];

for (size\_t i = 1; i <= N - 1; i++)

{

alpha[i + 1] = (-b[i]) / (c[i] + a[i] \* alpha[i]);

}

return alpha;

}

float\* GetVectorBeta(const float\* a, const float\* c, const float\* f, const float\* alpha)

{

auto beta = new float[N + 1];

beta[0] = f[0] / c[0];

for (size\_t i = 1; i < N; i++)

{

/\*std::cout << "(" << f[i] << " - " << "( " << a[i] << " \* " << beta[i - 1] << "))";

std::cout << " / (" << c[i] << " + " << a[i] << " \* " << alpha[i] << ")" << std::endl;\*/

beta[i] = (f[i] - a[i] \* beta[i - 1]) / (c[i] + a[i] \* alpha[i]);

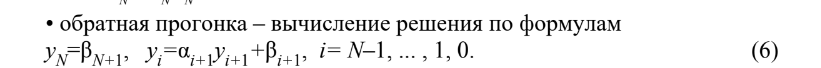
}

beta[N] = (f[N] - a[N] \* beta[N - 1]) / (c[N] + a[N] \* alpha[N]);

return beta;

}

Обратная прогонка



float\* GetVectorY1(const float\* alpha, const float\* beta)

{

auto y1 = new float[N + 1];

y1[N] = beta[N];

for (int i = N - 1; i >= 0; i--)

{

y1[i] = alpha[i + 1] \* y1[i + 1] + beta[i];

}

return y1;

}

float GetRelativeError(const float\* y, const float\* y1)

{

float normNumerator = 0.f;

float normDenominator = 0.f;

for (size\_t i = 0; i < N + 1; i++)

{

normDenominator = std::max(normDenominator, std::fabsf(y[i]));

}

for (size\_t i = 0; i < N + 1; i++)

{

normNumerator = std::max(normNumerator, std::fabsf(y[i] - y1[i]));

}

return (normNumerator / normDenominator);

}

void solution()

{

std::cout << std::setprecision(8);

auto a = GetVectorA();

auto b = GetVectorB();

auto c = GetVectorC();

auto y = GetVectorY();

auto f = GetVectorF(a, b, c, y);

auto t3 = std::chrono::steady\_clock::now();

auto alpha = GetVectorAlpha(a, b, c, f);

auto beta = GetVectorBeta(a, c, f, alpha);

auto y1 = GetVectorY1(alpha, beta);

auto t4 = std::chrono::steady\_clock::now();

std::cout << "1) Accurate solution y = " << y;

std::cout << "2) Approximate solution y\* = " << y1;

std::cout << "3) Relative error = " << GetRelativeError(y, y1) << std::endl;

std::cout << "4) Time = " << std::chrono::duration<double>(t4 - t3).count() << "s " << std::endl;

}

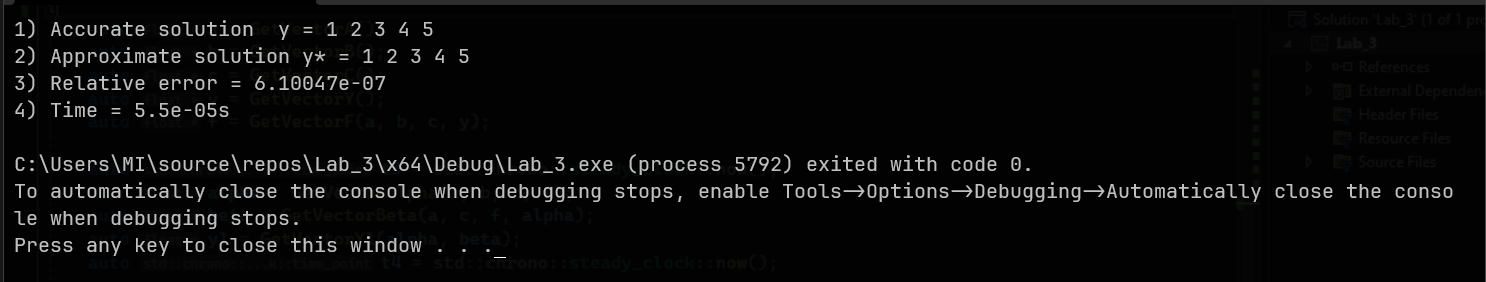
int main()

{

solution();

}

Результаты:



Выводы:

Метод правой прогонки является устойчивым решением СЛАУ с трех-диагональной матрицей, а скорость работы данного алгоритма   
O(n).