|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Лабораторная работа № 4 | | |
| по дисциплине « Уравнения математической физики» | | |
| **решение несимметричных слау** | | |
|  | | |
|  | Бригада 1 | ЮЗЯК МАРИНА |
| Группа ПМ-92 | глушко владислав |
| Вариант 9 |  |
|  |  |
|  |  |
|  |  |
| Преподаватели | патрушев илья игоревич |
|  | задорожный александр геннадьевич |
| Новосибирск, 2022 | | |

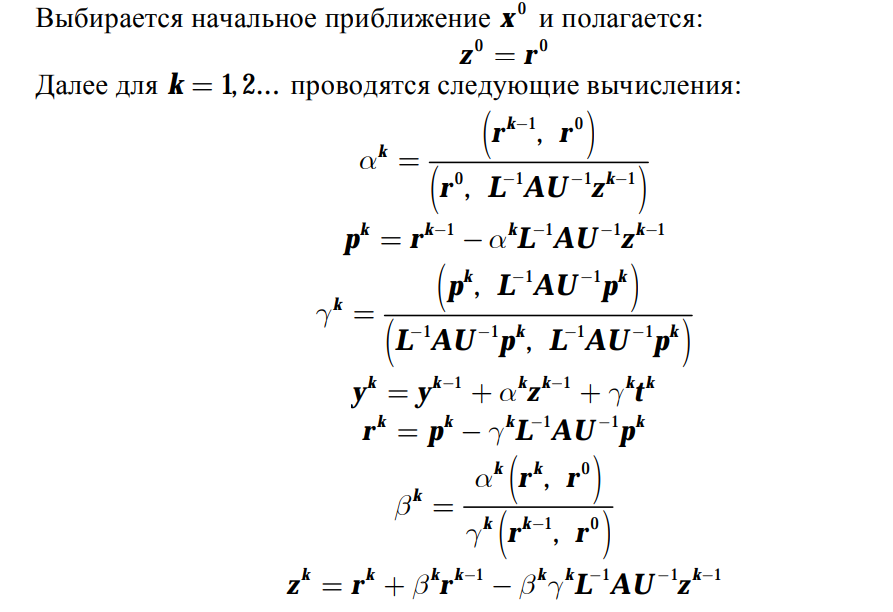
**Цель работы**

Изучить особенности реализации итерационных методов BCG, BCGStab, GMRES для СЛАУ с несимметричными разреженными матрицами. Исследовать влияние предобусловливания на сходимость этих методов.

**Задание**

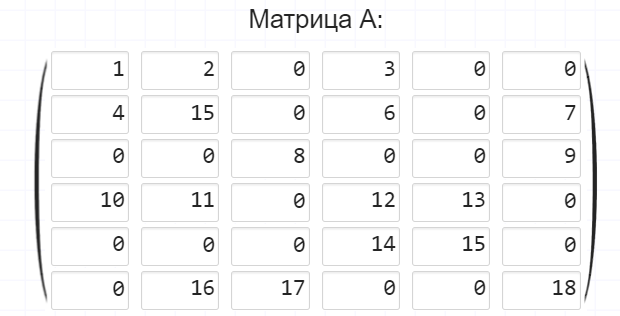
Реализовать решение СЛАУ методом BSGSTAB с LU-предобуславливанием.

**Теоретическая часть**

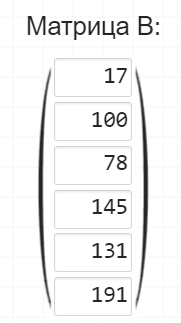
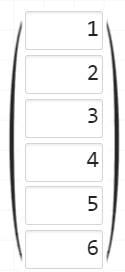


**Исследования**

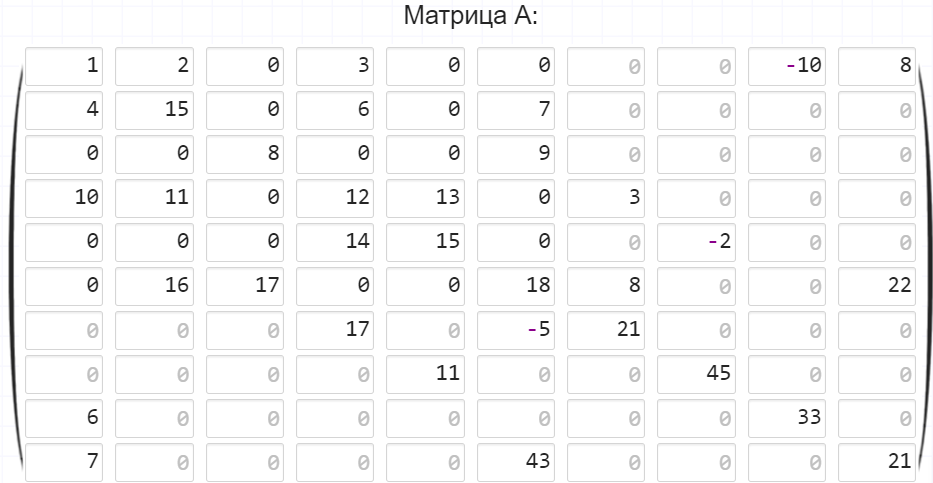
*На матрице небольшой размерности*



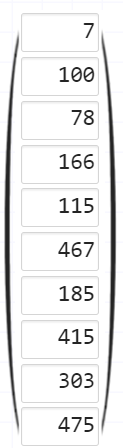
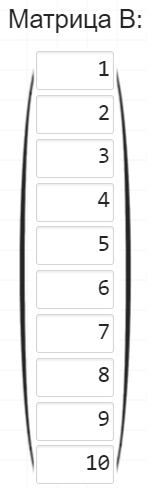
Вектор правой части: Точное решение:

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод решения** | **Время**  **(c)** | **Количество итераций** | **Решение** |
| LOS (LU) | 0 | 15 | 1.00000000000004  2.00000000000001  3.00000000000004  3.99999999999997  5.00000000000002  5.99999999999996 |
| BCG STAB (LU) | 0 | 4 | 0.999999999999991  2  3  4  5  6 |



Вектор правой части: Точное решение:

|  |  |  |  |
| --- | --- | --- | --- |
| **Метод решения** | **Время**  **(c)** | **Количество итераций** | **Решение** |
| LOS (LU) | 0.18 | 10000 (max) | -4.55869056806626  5.10361432124894  -0.23216578498979  -2.27342923917211  10.429954371949  8.63956472279989  12.4863783882631  6.42424016591432  4.04300465318339  6.05165287695612 |
| BCG STAB (LU) | 0 | 11 | 0.999999999999883  1.99999999999998  2.99999999999998  4.00000000000006  4.99999999999996  6.00000000000001  7  8.00000000000001  9.00000000000002  10 |

*Матрица 0945 из третьей лабораторной по численным методам*

|  |  |  |
| --- | --- | --- |
| **Метод решения** | **Время**  **(c)** | **Количество итераций** |
| LOS (LU) | 0.013 | 11 |
| BCG STAB (LU) | 0.014 | 7 |

*Матрица 4545 из третьей лабораторной по численным методам*

|  |  |  |
| --- | --- | --- |
| **Метод решения** | **Время**  **(c)** | **Количество итераций** |
| LOS (LU) | 0.063 | 11 |
| BCG STAB (LU) | 0.07 | 7 |

**Выводы**

В результате проделанной работы можно сделать следующие выводы:

* метод BCG сходится за меньшее количество итераций, чем LOS;
* время работы методов отличается незначительно (десятые, сотые секунды, не более);
* для данных тестовых матриц небольшой размерности (10 на 10) метод BCG сошелся, а вот LOS превысил максимальное количество итераций и не сошелся к истинному решению.

Рекомендуется использовать метод BCG STAB для решения СЛАУ больших размерностей.

**Код программы**

*Solver.hpp*

#ifndef \_SOLVER\_HPP

#define \_SOLVER\_HPP

#include <filesystem>

#include <ctime>

#include "lightweight.hpp"

#include "Vector.hpp"

using std::filesystem::path;

class Solver {

public:

struct Matrix {

Vector di;

Vector ggl;

Vector ggu;

std::vector<size\_t> ig;

std::vector<size\_t> jg;

Vector operator\*(const Vector& v) const;

};

Solver(const path& path) {

readParams(path);

clock\_t start = clock();

//size\_t iter = losLU();

size\_t iter = BСGSTABLU();

clock\_t end = clock();

double seconds = double(end - start) / CLOCKS\_PER\_SEC;

std::cout << "Number of iterations: " << iter << "\nTime: " << seconds << " c.\n" << std::endl;

printVector(x);

};

private:

void readParams(const path& path);

void readSparseMatrix(const path& path); // чтение матрицы, если она в разреженном формате.

void readDenseMatrix(std::ifstream& fin); // чтение матрицы в плотном формате.

void readGg(Vector& gg, std::ifstream& fin); // чтение массивов ggl и ggu.

void convertDenseMatrix(); // из плотного формата в разреженый строчно-стоблцовый.

void incompleteLUDecomposition(Vector& L, Vector& U, Vector& D);

void straightMove(const Vector& L, const Vector& D, const Vector& v1, Vector& v2);

void backMove(const Vector& U, const Vector& v1, Vector& v2);

size\_t BСGSTABLU();

size\_t losLU();

private:

size\_t n; // размерность матрицы системы.

size\_t max\_iter; // макисмальное количество итераций.

size\_t is\_sparse; // нужно или не нужно конвертировать плотный формат в разреженный.

double eps; // точность вычислений.

Matrix A\_sparse; // матрица системы в разреженном строчно-столбцовом формате.

Vector b;

Vector x;

std::vector<std::vector<double>> A; // матрица в плотном формате (только для матриц небольших размерностей!)

};

size\_t Solver::BСGSTABLU() {

size\_t n = A\_sparse.di.size();

Vector r(n), r\_0(n), p(n), s(n), LAUp(n), LAUs(n);

Vector L(A\_sparse.ig[n]), U(A\_sparse.ig[n]), D(n);

double alpha, omega, betta, dotPrr\_0r;

double normb = sqrt(b \* b);

double residual;

incompleteLUDecomposition(L, U, D);

straightMove(L, D, b, r\_0);

r = r\_0;

backMove(U, r\_0, p);

size\_t k = 1;

for (; k < max\_iter && (residual = sqrt(r \* r) / normb) > eps; k++) {

backMove(U, p, LAUp);

LAUp = A\_sparse \* LAUp;

straightMove(L, D, LAUp, LAUp);

dotPrr\_0r = r\_0 \* r;

alpha = dotPrr\_0r / (LAUp \* r\_0);

s = r - alpha \* LAUp;

backMove(U, s, LAUs);

LAUs = A\_sparse \* LAUs;

straightMove(L, D, LAUs, LAUs);

omega = (LAUs \* s) / (LAUs \* LAUs);

x += alpha \* p + omega \* s;

r = s - omega \* LAUs;

betta = alpha \* (r \* r\_0) / (omega \* dotPrr\_0r);

p = r + betta \* (p - omega \* LAUp);

}

backMove(U, x, x);

return k;

}

size\_t Solver::losLU() {

size\_t n = A\_sparse.di.size();

Vector r(n), z(n), p(n), LAUr(n), Ur(n);

Vector L(A\_sparse.ig[n]);

Vector U(A\_sparse.ig[n]);

Vector D(n);

double alpha, betta, dotPrpp;

double normb = sqrt(b \* b);

double residual;

incompleteLUDecomposition(L, U, D);

straightMove(L, D, b, r);

backMove(U, r, z);

straightMove(L, D, A\_sparse \* z, p);

size\_t k = 1;

for (; k < max\_iter && (residual = sqrt(r \* r) / normb) > eps; k++) {

dotPrpp = p \* p;

alpha = (p \* r) / dotPrpp;

x += alpha \* z;

r -= alpha \* p;

backMove(U, r, Ur);

LAUr = A\_sparse \* Ur;

straightMove(L, D, LAUr, LAUr);

betta = (-p \* LAUr) / dotPrpp;

z = Ur + betta \* z;

p = LAUr + betta \* p;

}

return k;

}

void Solver::incompleteLUDecomposition(Vector& L, Vector& U, Vector& D) {

size\_t n = A\_sparse.di.size();

D = A\_sparse.di;

L = A\_sparse.ggl;

U = A\_sparse.ggu;

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

double d = 0;

size\_t temp = A\_sparse.ig[i];

for (size\_t j = A\_sparse.ig[i]; j < A\_sparse.ig[i + 1]; j++) {

double ls = 0;

double us = 0;

for (size\_t h = A\_sparse.ig[A\_sparse.jg[j]], k = temp; h < A\_sparse.ig[A\_sparse.jg[j] + 1] && k < j;)

if (A\_sparse.jg[k] == A\_sparse.jg[h]) {

ls += L[k] \* U[h];

us += L[h++] \* U[k++];

}

else (A\_sparse.jg[k] < A\_sparse.jg[h]) ? k++ : h++;

L[j] -= ls;

U[j] = (U[j] - us) / D[A\_sparse.jg[j]];

d += L[j] \* U[j];

}

D[i] -= d;

}

}

void Solver::straightMove(const Vector& L, const Vector& D, const Vector& v1, Vector& v2) {

size\_t n = A\_sparse.di.size();

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

double sum = 0;

for (size\_t j = A\_sparse.ig[i]; j < A\_sparse.ig[i + 1]; j++)

sum += v2[A\_sparse.jg[j]] \* L[j];

v2[i] = (v1[i] - sum) / D[i];

}

}

void Solver::backMove(const Vector& U, const Vector& v1, Vector& v2) {

size\_t n = A\_sparse.di.size();

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

v2[i] = v1[i];

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

for (size\_t j = A\_sparse.ig[i]; j < A\_sparse.ig[i + 1]; j++)

v2[A\_sparse.jg[j]] -= v2[i] \* U[j];

}

void Solver::readParams(const path& path) {

std::ifstream fin(path / "isSparse.txt");

if (fin.is\_open()) { fin >> is\_sparse; fin.close(); }

else printErrorFile("isSparse.txt");

fin.open(path / "params.txt");

if (fin.is\_open()) { fin >> eps >> max\_iter; fin.close(); }

else printErrorFile("params.txt");

fin.open(path / "A\_size.txt");

if (fin.is\_open()) { fin >> n; fin.close(); }

else printErrorFile("A\_size.txt");

b.resize(n);

fin.open(path / "b.txt");

if (fin.is\_open()) { readVector(b, fin); fin.close(); }

else printErrorFile("b.txt");

x.resize(n);

fin.open(path / "init\_x.txt");

if (fin.is\_open()) { readVector(x, fin); fin.close(); }

else printErrorFile("init\_x.txt");

switch (is\_sparse) {

case 0:

A.resize(n);

fin.open(path / "A.txt");

if (fin.is\_open()) { readDenseMatrix(fin); fin.close(); }

else printErrorFile("A.txt");

convertDenseMatrix();

break;

case 1:

A\_sparse.di.resize(n);

A\_sparse.ig.resize(n + 1);

readSparseMatrix(path);

break;

default:

std::cerr << "There is no such matrix format. 0 - sparse format, 1 - dense format." << std::endl;

std::exit(1);

break;

}

}

void Solver::convertDenseMatrix() {

}

void Solver::readSparseMatrix(const path& path) {

std::ifstream fin(path / "di.txt");

if (fin.is\_open()) { readVector(A\_sparse.di, fin); fin.close(); }

else printErrorFile("di.txt");

fin.open(path / "ig.txt");

if (fin.is\_open()) { readVector(A\_sparse.ig, fin); fin.close(); }

else printErrorFile("ig.txt");

fin.open(path / "ggu.txt");

if (fin.is\_open()) { readGg(A\_sparse.ggu, fin); fin.close(); }

else printErrorFile("ggu.txt");

fin.open(path / "ggl.txt");

if (fin.is\_open()) { readGg(A\_sparse.ggl, fin); fin.close(); }

else printErrorFile("ggl.txt");

A\_sparse.jg.resize(A\_sparse.ggu.size());

fin.open(path / "jg.txt");

if (fin.is\_open()) { readVector(A\_sparse.jg, fin); fin.close(); }

else printErrorFile("jg.txt");

}

void Solver::readDenseMatrix(std::ifstream& fin) {

for (size\_t i = 0; i < A.size(); i++) {

for (size\_t j = 0; j < A.size(); j++)

fin >> A[i][j];

}

}

void Solver::readGg(Vector& gg, std::ifstream& fin) {

double value;

while (fin >> value)

gg.push\_back(value);

}

Vector Solver::Matrix::operator \* (const Vector& v) const {

Vector result(v.size());

for (size\_t i = 0; i < v.size(); i++) {

result[i] = di[i] \* v[i];

for (size\_t k = ig[i]; k < ig[i + 1]; k++) {

size\_t j = jg[k];

result[i] += ggl[k] \* v[j];

result[j] += ggu[k] \* v[i];

}

}

return result;

}

#endif // !\_SOLVER\_HPP

*Vector.hpp*

#ifndef \_VECTOR\_HPP

#define \_VECTOR\_HPP

#include <vector>

typedef std::vector<double> V\_n;

class Vector : public V\_n {

public:

Vector(int n = 0) : V\_n(n) {}

Vector(const V\_n& v) : V\_n(v) {}

Vector(int n, double value) : V\_n(n, value) {}

Vector operator+(const Vector& v) const;

Vector operator-(const Vector& v) const;

Vector operator\*(double value) const;

double operator\*(const Vector& v) const;

Vector operator-() const;

Vector& operator+=(const Vector& v);

Vector& operator-=(const Vector& v);

friend Vector operator\*(double value, const Vector& v) { return v \* value; }

};

Vector Vector::operator+(const Vector& v) const

{

Vector result(v.size());

for (int i = 0; i < v.size(); i++)

result[i] = (\*this)[i] + v[i];

return result;

}

Vector Vector::operator-(const Vector& v) const

{

Vector result(v.size());

for (int i = 0; i < v.size(); i++)

result[i] = (\*this)[i] - v[i];

return result;

}

Vector Vector::operator\*(double value) const

{

Vector result((\*this).size());

for (int i = 0; i < (\*this).size(); i++)

result[i] = value \* (\*this)[i];

return result;

}

double Vector::operator\*(const Vector& v) const

{

double result = 0;

for (int i = 0; i < (\*this).size(); i++)

result += v[i] \* (\*this)[i];

return result;

}

Vector Vector::operator-() const

{

Vector result((\*this).size());

for (int i = 0; i < (\*this).size(); i++)

result[i] = -(\*this)[i];

return result;

}

Vector& Vector::operator+=(const Vector& v)

{

for (int i = 0; i < (\*this).size(); i++)

(\*this)[i] += v[i];

return \*this;

}

Vector& Vector::operator-=(const Vector& v)

{

for (int i = 0; i < (\*this).size(); i++)

(\*this)[i] -= v[i];

return \*this;

}

#endif // !\_VECTOR\_HPP

*lightweight.cpp*

#ifndef \_LIGHTWEIGHT\_HPP

#define \_LIGHTWEIGHT\_HPP

#include <iostream>

#include <vector>

#include <fstream>

#include <iomanip>

void printErrorFile(std::string file\_name) {

std::cerr << "File " << file\_name << " is not open." << std::endl;

std::exit(1);

}

template <typename T>

void readVector(std::vector<T>& v, std::ifstream& fin) {

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

fin >> v[i];

}

void printVector(std::vector<double>& v) {

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

std::cout << std::setprecision(15) << v[i] << std::endl;

}

#endif // !\_LIGHTWEIGHT\_HPP

*main.cpp*

#include "Solver.hpp"

int main() {

using ::std::filesystem::path;

path dir\_to\_file = "files/test4";

Solver s(dir\_to\_file);

return 0;

}