БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Мастыкина Елизавета Александровна

**Отчет**

**по Лабораторной работе 3**

студента 2 курса 6 группы

**Преподаватель**

*Горбачева Юлия Николаевна*

**Минск, 2021**

**Постановка задачи**

Для проведения вычислительного эксперимента сгенерировать симметрическую положительно определенную матрицу A с диагональным преобладанием (матрица генерируется один раз, для всех заданий она одна и та же). Для этого сначала заполняем нижнюю треугольную

часть матрицы A (т.е. элементы аij , где i > j ), используя случайные целые числа из диапазона от -100 до 100. Верхнюю треугольную часть, где i < j , заполнить симметрично нижней части. Затем заполнить диагональ.

Задать точное решение x = (1, 2, , n)T и f = Ax.

1. Решить СЛАУ с матрицей A порядка n = 10 и правой частью f методом градиентного спуска.
2. Для СЛАУ с матрицей A порядка n=10 и правой частью f исследовать сходимость метода релаксации в зависимости от параметра релаксации ω∈{0.2, 0.5, 0.8,1,1.3,1.5,1.8}. Результаты вычислительного эксперимента оформить в виде таблицы.

Краткие теоретические сведения

Метод градиентного спуска

Алгоритм метода градиентного спуска принимает вид:

= A *f,*

,

, k = 0,1…

Метод сходится при любых начальных условиях если .

Метод релаксации.

Алгоритм метода релаксации:

Достаточное условие

Если и метод релаксации будет сходится при любом начальном приближении.

Листинг программы:

#include<iostream>

#include<iomanip>

#include<math.h>

using namespace std;

double\* matrixMultiplication(double\*\* first, double\* second, int dim);

double\* generateF(double\*\* matrix, int dim);

void printMatrix(double\*\* matrix, int dim)

{

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

{

for (int j = 0; j < dim; ++j)

{

cout << setw(10) << matrix[i][j];

}

cout << endl;

}

cout << endl;

}

void inputMatrix(double\*\* matrix, int dim)

{

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

{

for (int j = 0; j < dim; ++j)

{

cin >> matrix[i][j];

}

}

}

void inputMatrix(double\* matrix, int dim)

{

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

{

cin >> matrix[i];

}

}

void printMatrix(double\* matrix, int dim);

double\*\* generateMatrix(int n)

{

srand(time(0));

double sum = 0; // required to generate doubleerval

double\*\* matrix = new double\* [n];

int a=-100, b =100;

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

{

matrix[i] = new double[n];

}

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

{

for (int j = 0; j < n; ++j)

{

matrix[i][j] = 0;

}

}

for (int j = 0; j < n; ++j)

{

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

{

matrix[j][i] = matrix[i][j] =( a + rand() % (b - a + 1));

}

}

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

{

sum = 0;

for (int j = 0; j < n; j++)

{

sum += abs(matrix[i][j]);

}

a = sum + 6;

b = sum + 60;

matrix[i][i] = rand() % (b - a + 1) + a;

}

return matrix;

}

double\* generateF(double\*\* matrix, int dim)

{

double\* vector = new double[dim];

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

{

vector[i] = i + 1;

}

cout << "Exact solution: " << endl;

printMatrix(vector, dim);

double\* result = matrixMultiplication(matrix, vector, dim);

return result;

}

double\* generateInitialAppr(double\*\* matrix,double\*b, int dim)

{

double\* result = new double[dim];

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

{

result[i] = b[i];

}

return result;

}

void matrixMultiplication(double\*\* first, double\*\* second, int dim)

{

double\*\* result = new double\* [dim];

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

{

result[i] = new double[dim];

}

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

{

for (int j = 0; j < dim; j++)

{

result[i][j] = 0;

for (int k = 0; k < dim; k++)

{

result[i][j] += (first[i][k] \* second[k][j]);

}

}

}

}

double\* matrixMultiplication(double\*\* first, double\* second, int dim)

{

double\* result = new double [dim];

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

{

result[i] = 0;

for (int j = 0; j < dim; j++)

{

result[i] += (first[i][j] \* second[j]);

}

}

return result;

}

double scolarMultiplication(double\* first, double\*second, int dim)

{

double result = 0;

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

{

result += first[i] \* second[i];

}

return result;

}

double\* calculateR(double\*\* matrix, double\* x, double\* f, int dim)

{

double\* r = new double[dim];

double\* temp = matrixMultiplication(matrix, x, dim);

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

{

r[i] = temp[i] - f[i];

}

return r;

}

double\* calculateX(double\*\* matrix, double\* x, double\* r, int dim)

{

double\* result = new double[dim];

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

{

result[i] = x[i] - (scolarMultiplication(r, r, dim) / scolarMultiplication(matrixMultiplication(matrix, r, dim), r, dim) ) \* r[i];

}

return result;

}

double\* calculateX(double\*\* matrix, double\* x,double\* f, double w, int dim)

{

double\* result = new double[dim];

double temp = 0;

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

{

temp = 0;

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

{

temp += matrix[i][j] \* result[j];

}

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

{

temp += matrix[i][j] \* x[j];

}

result[i] = (1 - w) \* x[i] + w \* (f[i] - temp) / matrix[i][i];

}

return result;

}

double findMaxNorm(double\*\* matrix, double\* f, double\*x, int dim)

{

double\* temp = new double[dim];

temp = matrixMultiplication(matrix, x, dim);

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

{

temp[i] -= f[i];

}

double res = 0;

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

{

if (abs(temp[i]) > res)

{

res = abs(temp[i]);

}

}

return res;

}

double findPogr(double\* a, double\* b, int dim)

{

double res = 0;

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

{

if(abs(a[i]-b[i]) > res)

{

res = abs(a[i] - b[i]);

}

}

return res;

}

void printMatrix(double\* matrix, int dim)

{

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

{

cout << matrix[i] << endl;

}

cout << endl;

}

void main()

{

setlocale(LC\_ALL, "Russian");

int dim = 10;

double\* exactSolution = new double[dim];

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

{

exactSolution[i] = i + 1;

}

double\*\* matrix = generateMatrix(dim);

cout << "matrix" << endl;

printMatrix(matrix, dim);

double\* f = generateF(matrix, dim);

cout << "f" << endl;

printMatrix(f, dim);

double\* x0 = generateInitialAppr(matrix, f, dim);

cout << "Initial approximation: " << endl;

printMatrix(x0, dim);

double\* r;

r = calculateR(matrix, x0, f, dim);

cout << "r" << endl;

printMatrix(r, dim);

double\* x;

x = calculateX(matrix, x0, r, dim);

double eps = 1e-7;

cout << "Accuracy: " << eps << endl;

int max = 1;

while(findMaxNorm(matrix, f, x, dim) > eps && max < 5000)

{

x = calculateX(matrix, x, r, dim);;

r = calculateR(matrix, x, f, dim);

max++;

}

cout << "Solution" << endl;

printMatrix(x, dim);

cout << "Num of iterations: " << max << endl;

cout << "Neviazka: " << findMaxNorm(matrix, f, x, dim) << endl;

cout << "Inaccuracy: " << findPogr(x, exactSolution, dim) << endl;

double w[] = {0.2,0.5, 0.8, 1, 1.3, 1.5, 1.8};

for (int i = 0; i < 7; i++)

{

cout << w[i] << endl;

x = calculateX(matrix, x0, f, w[i], dim);

int max = 1;

while (findMaxNorm(matrix, f, x, dim) > eps && max < 5000)

{

x = calculateX(matrix, x, f, w[i], dim);;

max++;

}

cout << "Solution" << endl;

printMatrix(x, dim);

cout << "Num of iterations: " << max << endl;

cout << "Neviazka: " << findMaxNorm(matrix, f, x, dim) << endl;

cout << "Inaccuracy: " << findPogr(x, exactSolution, dim) << endl;

}

}

Вывод программы:

matrix

416 -65 28 45 -59 5 -53 41 -56 16

-65 573 -83 -97 -9 -68 -32 -91 7 -83

28 -83 400 10 69 -27 27 48 38 10

45 -97 10 593 -70 -81 -54 55 -60 79

-59 -9 69 -70 513 -22 -78 30 -63 -73

5 -68 -27 -81 -22 391 -31 20 -85 5

-53 -32 27 -54 -78 -31 543 66 90 81

41 -91 48 55 30 20 66 491 -55 58

-56 7 38 -60 -63 -85 90 -55 487 23

16 -83 10 79 -73 5 81 58 23 487

Exact solution:

1

2

3

4

5

6

7

8

9

10

f

-102

-1728

2300

1729

680

928

5121

4968

3810

5969

Initial approximation:

-102

-1728

2300

1729

680

928

5121

4968

3810

5969

r

-8866

-2.49434e+06

1.67862e+06

1.32648e+06

-539181

-91591

3.87749e+06

3.30648e+06

2.03215e+06

3.94789e+06

Accuracy: 1e-07

Solution

1

2

3

4

5

6

7

8

9

10

Num of iterations: 50

Neviazka: 4.55479e-08

Inaccuracy: 1.66264e-10

0.2

Solution

1

2

3

4

5

6

7

8

9

10

Num of iterations: 267

Neviazka: 9.02444e-08

Inaccuracy: 3.346e-10

0.5

Solution

1

2

3

4

5

6

7

8

9

10

Num of iterations: 88

Neviazka: 7.78914e-08

Inaccuracy: 2.51958e-10

0.8

Solution

1

2

3

4

5

6

7

8

9

10

Num of iterations: 41

Neviazka: 5.20322e-08

Inaccuracy: 1.45185e-10

1

Solution

1

2

3

4

5

6

7

8

9

10

Num of iterations: 20

Neviazka: 4.24818e-08

Inaccuracy: 1.07192e-10

1.3

Solution

1

2

3

4

5

6

7

8

9

10

Num of iterations: 31

Neviazka: 2.74625e-08

Inaccuracy: 7.21709e-11

1.5

Solution

1

2

3

4

5

6

7

8

9

10

Num of iterations: 51

Neviazka: 5.30642e-08

Inaccuracy: 1.1845e-10

1.8

Solution

1

2

3

4

5

6

7

8

9

10

Num of iterations: 153

Neviazka: 9.06127e-08

Inaccuracy: 2.05351e-10

Результаты

|  |  |  |  |
| --- | --- | --- | --- |
| Параметр ω | Номер итерации q, при которой достигнута требуемая точность |  |  |
| 0.2 | 267 | 9.02444e-08 | 3.346e-10 |
| 0.5 | 88 | 7.78914e-08 | 2.51958e-10 |
| 0.8 | 41 | 5.20322e-08 | 1.45185e-10 |
| 1 | 20 | 4.24818e-08 | 1.07192e-10 |
| 1.3 | 31 | 2.74625e-08 | 7.21709e-11 |
| 1.5 | 51 | 5.30642e-08 | 1.1845e-10 |
| 1.8 | 153 | 9.06127e-08 | 2.05351e-10 |

Выводы: Метод градиентного спуска сходится, так как А – симметрическая и положительно определённая. Точность решения и скорость сходимости зависят от исходной матрицы, а количество итераций определяется выбранным значением эпсилон . Сравнивая с методом релаксации (при хорошем выборе параметра релаксации w = 1.3) метод градиентного спуска даёт чуть-чуть худшую точность (погрешность порядка 1.66264e-10 против 7.21709e-11) и меньшую скорость сходимости (50 итераций против 31).