Лист ответа на контрольно-измерительный материал

Фамилия Имя Отчество обучающегося: Колесникова Елизавета Анатольевна

Направление подготовки: 02.03.03 – математическое обеспечение и администрирование информационных систем

Дисциплина: Численные методы

Курс: 3

Форма обучения: очная

Вид аттестации: текущая

Вид контроля: отчёт по лабораторной работе

ЛР – 1. Решение системы линейных алгебраических уравнений

с трёхдиагональной матрицей

Проверила

Махинова Ольга Алексеевна

1. Реализовать классы для работы с векторами и трехдиагональными матрицами.

Для вектора: сложение, вычитание, скалярное умножение, вычисление нормы как максимальной по модулю компоненты, считывание из файла / экрана, вывод в файл / на экран, заполнение случайными числами из диапазона

Для матрицы: сложение, вычитание, умножение на вектор, считывание с файла / экрана, вывод в файл / на экран

2. Реализовать метод прогонки.

3. Провести вычислительный эксперимент — использовать зависимость размера погрешности от системы
Результаты погрешностей в виде таблицы
1 столбец: размер системы
2 столбец: погрешность метода прогонки

4. Анализ задания.
Рассмотрим СЛАУ вида:

$$\begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ 0 & a_3 & b_3 & c_3 & & \\ 0 & 0 & \dots & & & \\ & & \dots & & & \\ & & a_{n-2} & b_{n-2} & c_{n-2} & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & a_n & b_n \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ \\ \\ x_{n-2} \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ \dots \\ \dots \\ d_{n-2} \\ d_{n-1} \\ d_n \end{pmatrix}$$

$$b_1 x_1 + c_1 x_2 = d_1$$
$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i, \quad i = \overline{2, n-1}$$
$$a_n x_{n-1} + b_n x_n = d_n$$

Метод прогонки

$$\begin{pmatrix} 1 & L_2 & & & \\ & 1 & L_3 & & \\ & & \dots & & \\ & & & 1 & L_{n-1} \\ & & & & 1 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} M_2 \\ M_3 \\ \dots \\ M_n \\ M_{n+1} \end{pmatrix}$$

Метод прогонки имеет 2 хода: прямой и обратный.
Во время прямого хода мы считаем коэффициенты L_i, M_i, на обратном — вектор-решение x_i.

Прямой ход:

$$L_{i+1} = \frac{c_i}{b_i - a_i L_i}, \quad M_{i+1} = \frac{d_i - a_i M_i}{b_i - a_i L_i}, \quad \text{где } i = \overline{1, n}, \; b_i - a_i c_i \neq 0$$

Обратный ход:

$$x_n = M_{n+1}, \quad x_i = M_{i+1} - L_{i+1} x_{i+1}, \quad i = n-1, \dots, 1$$

4. Тестовый пример.

1. Система уравнений:

$$\begin{pmatrix} 4 & 2 & 0 & 0 & 0 \\ 2 & 6 & 1 & 0 & 0 \\ 0 & 5 & 5 & 2 & 0 \\ 0 & 0 & 2 & 11 & 2 \\ 0 & 0 & 0 & 5 & 8 \end{pmatrix} \cdot x = \begin{pmatrix} 2 \\ -2 \\ 3 \\ -5 \\ 3 \end{pmatrix}.$$

2. Система уравнений, приведённая к верхнетреугольному виду:

$$\begin{pmatrix} 1 & 0.5 & 0 & 0 & 0 \\ 0 & 1 & 0.2 & 0 & 0 \\ 0 & 0 & 1 & 0.5 & 0 \\ 0 & 0 & 0 & 1 & 0.2 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot x = \begin{pmatrix} 0.5 \\ -0.6 \\ 1.5 \\ -0.8 \\ 1 \end{pmatrix}.$$

3. Точное решение:

$$x^* = \begin{pmatrix} 1 \\ -1 \\ 2 \\ -1 \\ 1 \end{pmatrix}.$$

Далее будет приведены результаты тестового примера на методе прогонки:

```
Данная матрица:
 4  2  0  0  0
 2  6  1  0  0
 0  5  5  2  0
 0  0  2  11  2
 0  0  0  5  8

Вспомогательные векторы:
Вектор L   :0 0.5 0.2 0.5 0.2 0
Вектор M   :0 0.5 -0.6 1.5 -0.8 1

Результаты вычислительного эксперимента
Точное решение: 1 -1 2 -1 1

Результат метода прогонки: 1 -1 2 -1 1
Погрешность: 0
```

Вычислительный эксперимент

Проведем несколько вычислительных экспериментов и составим таблицы погрешностей следующего вида:

- 1 столбец — размер системы;
- 2 столбец — погрешность метода прогонки;
- 3 столбец — погрешность неустойчивого метода.

В экспериментах будут меняться диапазоны значений для каждой из диагоналей.

Эксперимент 1

Матрица с перегруженной главной диагональю.

Случайно сгенерируем элементы диагоналей следующим образом:

- диапазон верхней диагонали c: [20; 30];
- диапазон главной диагонали b: [200; 300]; • диапазон нижней диагонали a: [20; 30].

```
Матрица с перегруженной главной диагональю.

Размер системы      | Погрешность метода прогонки
-------------------------------------------------
4                   |        0
-------------------------------------------------
16                  |        1.42109e-14
-------------------------------------------------
64                  |        2.84217e-14
-------------------------------------------------
256                 |        2.84217e-14
-------------------------------------------------
1024                |        2.84217e-14
-------------------------------------------------
4096                |        2.84217e-14
```

Эксперимент 2

Матрица с перегруженной нижней диагональю.

Случайно сгенерируем элементы диагоналей следующим образом:

- диапазон верхней диагонали c: [20; 30];
- диапазон главной диагонали b: [20; 30];
- диапазон нижней диагонали a: [200; 300].

Матрица с перегруженной нижней диагональю.

Размер системы	Погрешность метода прогонки
4	1.56319e-13
16	2.32321e-07
64	5.71934e+16
256	3.06651e+113
1024	inf
4096	inf

Эксперимент 3

Матрица с перегруженной верхней диагональю.

Случайно сгенерируем элементы диагоналей следующим образом:

- диапазон верхней диагонали c: [200; 300]; • диапазон главной диагонали b: [20; 30]; • диапазон нижней диагонали a: [20; 30].

Матрица с перегруженной верхней диагональю.

Размер системы	Погрешность метода прогонки
4	1.84741e-13
16	1.45731e-07
64	2.108e+17
256	1.31782e+113
1024	nan
4096	nan

Эксперимент 4

Матрица с перегруженной верхней и нижней диагоналями.

Случайно сгенерируем элементы диагоналей следующим образом:

- диапазон верхней диагонали c: [200; 300];
- диапазон главной диагонали b: [20; 30];
- диапазон нижней диагонали a: [200; 300].

Матрица с перегруженной верхней и нижней диагоналями.

Размер системы	Погрешность метода прогонки
4	1.56319e-13
16	1.56319e-13
64	3.23297e-13
256	1.14539e-11
1024	1.11875e-11
4096	1.32516e-11

Эксперимент 5

Матрица с близкими значений диагоналей.

Случайно сгенерируем элементы диагоналей следующим образом:

- диапазон верхней диагонали c: [20; 30];
- диапазон главной диагонали b: [22; 33]; • диапазон нижней диагонали a: [19; 29].

Матрица с близкими значений диагоналей

Размер системы	Погрешность метода прогонки
4	2.66454e-14
16	1.13687e-13
64	3.90799e-13
256	1.24558e-11
1024	0.00024607
4096	1.53244e+24

Вывод

Метод прогонки – модификация метода Гаусса для трехдиагональной матрицы. Сложность метода Гаусса равна $O(n^3)$, в то время как сложность метода прогонки равна O(n). Метод называется экономичным, если вычисление кадой неизвестной не зависит от размерности системы. Метод прогонки является устойчивым.

Также, при решении данной задачи немаловажна обусловленность системы, которая влияет на погрешность, если условие ($|b_i| < |a_i| + |c_i|$) не выполняется, то есть система является плохо обусловленной. В процессе вычисления возникают ошибки, так как погрешность растет, соответственно ошибки накапливаются. Соответственно метод прогонки не подходит для работы с плохо обусловленной матрицей, в остальных случаях существенных проблем не возникает

После проведение вычислительных экспериментов с использованием написанной программы, можно сделать вывод, что размерность трехдиагональной матрицы влияет на точность метода, исходя из результатов тестирования при больших размерностях систем.

Листинг программы

//Метод прогонки

```
vector_class SweepMethod(Matr_class &matrix) { // метод прогонки
    vector_class solution = vector_class(matrix.size);
    const int size = matrix.size+1;
    // прямой ход -- считаем векторы M и L
    vector_class L = vector_class(size);
    vector_class M = vector_class(size);
// теперь по реккурентным формулам рассчитаем остальные коэффициенты
    for (int i = 1; i < size; ++i) {

        if ((matrix.b.vec[i] - matrix.a.vec[i] * L.vec[i]) != 0)
        {
            L.vec[i+1] = matrix.c.vec[i] / (matrix.b.vec[i] -
matrix.a.vec[i] * L.vec[i]);
            M.vec[i+1] = (matrix.d.vec[i] - matrix.a.vec[i] *
M.vec[i]) /
                         (matrix.b.vec[i] - matrix.a.vec[i] *
L.vec[i]);

        }
    }

    L.vec[size] = 0;
    M.vec[size+1] = (matrix.d.vec[1] - matrix.a.vec[size-1] *
M.vec[size-1]) /
                    (matrix.b.vec[size-1] - matrix.a.vec[size-1] *
L.vec[size-1]);

    //обратный ход -- считаем вектор-решение
    solution.vec[size - 1] = M.vec[size];
    for (int i = size - 1; i >= 1; --i) {
        solution.vec[i] = M.vec[i+1] - L.vec[i+1] *
solution.vec[i+1];
    }
    return solution;
}
```

// vector_class.h

```
class vector_class {
public:

    vector<double> vec;
    int size;

    vector_class();
    vector_class(int n);

    void FillVectorConsole();
    void FillVectorFile(ifstream& file); //заполнение вектора из
файла

    vector_class Multiply(const vector_class& sub);
    vector_class Minus(const vector_class& sub);
    double VecNorm();
    void GenerateVec(int m, int n);

    void OutInConsole();


};
vector_class::vector_class() {
    size = 0;

}
vector_class::vector_class(int n) {
    size = n;
    for (int i = 0; i <= size; ++i) {
        vec.push_back(0);
    }
}

void vector_class::FillVectorFile(ifstream &file) {
    for (int i = 1; i <= size || !file.eof(); i++)
    {
        if (!file.eof())
            file >> vec[i];
    }
}

void vector_class::FillVectorConsole() {
    cout << "Введите вектор, из n элементов: \n";
    for (int i = 1; i <= size; i++)
    {
        cin >> vec[i];
    }
}

//vector_class vector_class::operator+(const vector_class &sub) {
//    for (int i = 1; i <= size; ++i)
//        this->vec[i] += sub.vec[i];
//}
//
```

```
//vector_class vector_class::operator-(const vector_class &sub) {
//    for (int i = 1; i <= size; ++i)
//        this->vec[i] -= sub.vec[i];
//}

vector_class vector_class::Multiply(const vector_class& sub) {
    vector_class res = vector_class(size);
    for (int i = 1; i <= size; ++i)
        this->vec[i] *= sub.vec[i];
    return res;
}
vector_class vector_class::Minus(const vector_class& sub) {
    vector_class res = vector_class(size);
    if (vec.size() == sub.vec.size()) {
        for (int i = 1; i <= size; ++i)
            res.vec[i] = vec[i] - sub.vec[i];
    }
    return res;
}

double vector_class::VecNorm() {
    double max = abs(vec[1]);
    for (int i = 1; i <= size; ++i) {
        if  (abs(vec[i]) > max)
            max = abs(vec[i]);
    }
    return max;
}

void vector_class::GenerateVec(int left, int right) {
    srand(time(NULL));
    for (int i = 1; i <= size; i++)
        this->vec[i] = ((double)rand() / (double)RAND_MAX * (right -
left) + left);
}
void vector_class::OutInConsole() {
    for (int i = 1; i <= size; ++i)
        cout << vec[i]  << ' ';
}
```

// Matr_class.h

```
class Matr_class {
    friend class vector_class;
    friend vector_class SweepMethod(Matr_class& matrix); // метод
прогонки
    vector_class a;
    vector_class b;
    vector_class c;
public:
    int size;

    vector_class d;

    Matr_class();
    explicit Matr_class(int n);

    void CreateMatrixFromConsole();
    void CreateMatrixFromFile(ifstream &file);
    void GenerateMatrix();
    void GenerateMatrixExtra(int a, int b, int c, int d, int x, int
y);

    void OutputToConsole();
    void OutputToFile();

    vector_class MultiplyMatrVec(const vector_class& v);

};

Matr_class::Matr_class() {
    size = 0;
    a = vector_class(size);
    b = vector_class(size);
    c = vector_class(size);
    d = vector_class(size);
}

Matr_class::Matr_class(int n) {
    size = n;
    a = vector_class(size);
    b = vector_class(size);
    c = vector_class(size);
    d = vector_class(size);
}

void Matr_class::CreateMatrixFromConsole() {
    cout << "Введите матрицу размерности " << size << " с
консоли\n";
    a.FillVectorConsole();
    b.FillVectorConsole();
    c.FillVectorConsole();
    d.FillVectorConsole();

    // a1 = cn = 0
    a.vec[1] = c.vec[size] = 0;
```

```
}

void Matr_class::CreateMatrixFromFile(ifstream &file) {
    if (file.is_open())
    {
        for (int i = 1; i <= size && !file.eof(); i++)
        {
            if (!file.eof())
                file >> a.vec[i];
        }
        for (int i = 1; i <= size && !file.eof(); i++)
        {
            if (!file.eof())
                file >> b.vec[i];
        }
        for (int i = 1; i <= size && !file.eof(); i++)
        {
            if (!file.eof())
                file >> c.vec[i];
        }
        for (int i = 1; i <= size && !file.eof(); i++)
        {
            if (!file.eof())
                file >> d.vec[i];
        }

        // a1 = cn = 0
        a.vec[1] = c.vec[size] = 0;
    }
    else
        cout << "проблемы с открытием файла";
    file.close();
}

void Matr_class::GenerateMatrix() {
    cout << "Введите границы диапазона для генерирования вектора a:
";
    int n, m; cin  >> n >> m;
    a.GenerateVec(n,m);
    cout << "Введите границы диапазона для генерирования вектора b:
";
    cin  >> n >> m;
    b.GenerateVec(n,m);
    cout << "Введите границы диапазона для генерирования вектора c:
";
    cin  >> n >> m;
    c.GenerateVec(n,m);


    // a1 = cn = 0
    a.vec[1] = c.vec[size] = 0;
}
void Matr_class::GenerateMatrixExtra(int q, int w, int e, int r, int
x, int y) {

    a.GenerateVec(q,w); // нижняя диагональ
```

```
    b.GenerateVec(e, r);
    c.GenerateVec(x, y);


    // a1 = cn = 0
    a.vec[1] = c.vec[size] = 0;
}

void Matr_class::OutputToConsole() {
    const int HalfSize = size / 2;
    for (int i = 1; i <= size; ++i) {
        //for (int j = 1; j <= size ; ++j) {
            // заполнение нулями когда они стоят в начале строк (с
3)
            if (i > HalfSize){
                for (int k = 0; k < i - HalfSize; ++k) {
                    cout << ' ' << '0' << ' ';
                }
            }
            // вывод диагональных векторов
            if (i != 1) // чтобы не учитывать a1
            {
                cout << ' ' << setprecision(3) << a.vec[i] << ' ';
            }
            cout << ' ' << setprecision(3) << b.vec[i] << ' ';
            if (i != size) {
                cout  << ' ' << setprecision(3) << c.vec[i] << ' ';
            }
            // заполнение нулями
            if (i <= HalfSize+1){
                for (int k = i + HalfSize; k <= size; ++k) {
                    cout << ' ' << '0' << ' ';
                }
            }
            // cout << "  " << d.vec[i];

        //}
        cout << endl;
    }

}
void Matr_class::OutputToFile() { // исправить открытие файла
    const int ElemCount = 3;
    for (int i = 1; i <= size; ++i) {
        for (int j = 1; j <= size ; ++j) {
            // заполнение нулями
            if (i >= ElemCount){
                for (int k = 1; k < i - k; ++k) {
                    cout << '0' << ' ';
                }
            }
            // вывод диагональных векторов
            if (i != 1) // чтобы не учитывать a1
            {
                cout << a.vec[i];
            }
```

```
            cout << b.vec[i];
            if (i != size) {
                cout << ' ' << c.vec[i];
            }
            // заполнение нулями
            if (i < 3){
                for (int k = ElemCount; k < k - i; ++k) {
                    cout << '0' << ' ';
                }
            }
            cout << "  " << d.vec[i];
        }
    }

}

vector_class Matr_class::MultiplyMatrVec(const vector_class &v) {
    if (v.size != size)
        cout << "ошибка. размерность вектора не совпадает с
размерностью матрицы";
    else {
        vector_class res = vector_class(size);
        for (int i = 1; i <= size; ++i) {
            if (i != 1)
                res.vec[i] += a.vec[i] * v.vec[i-1];
            res.vec[i] += b.vec[i] * v.vec[i];
            if (i != size)
                res.vec[i] += c.vec[i] * v.vec[i+1];
        }
        if (res.size != size)
            cout << "ошибка при умножении вектора на матрицу";
        else
            return res;
    }
}
```