Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)

Институт фундаментального образования

Департамент «школа бакалавриата»



ОТЧЕТ

отчет по лабораторной работе дисциплины «Алгоритмы и анализ

сложности»

**«Алгебраический алгоритм вычисления обратной матрицы»**

Студент: Чернышев Иван Антонович РИ-210911

Преподаватель: Трофимова Ольга Геннадиевна, Доцент, к.т.н.

Екатеринбург

2021

**Оглавление**

[Теоретическая часть 4](#_Toc73663921)

[1. Вычисление определителя матрицы 4](#_Toc73663922)

[2. Вычисление обратной матрицы с помощью алгебраических дополнений 6](#_Toc73663923)

[3. Решение квадратной системы уравнений с помощью обратной матрицы 8](#_Toc73663924)

[Инструкция для пользователя 9](#_Toc73663925)

[Инструкция для программиста 10](#_Toc73663926)

[Тестирование 11](#_Toc73663927)

[Вывод 12](#_Toc73663928)

[Литература 13](#_Toc73663929)

[Приложение 14](#_Toc73663930)

**Задание**

1. Написать функцию вычисления определителя квадратной матрицы, раскрывая определитель по строке.
2. Написать функцию вычисления обратной матрицы, с использованием алгебраических дополнений.
3. Написать функцию умножения матрицы на вектор.
4. Решить квадратную систему линейных уравнений методом обратной матрицы.

Функция выдает ответы:

* решение в случае его единственности;
* нет решений или решений бесконечно много

**Теоретическая часть**

1. **Вычисление определителя матрицы**

Пусть задана квадратная матрица второго порядка

Определитель этой матрицы (определитель второго порядка) вычисляется по следующей формуле:

Далее, пусть задана квадратная матрица третьего порядка Определитель этой матрицы (определитель третьего порядка) вычисляется так:

Для определителя четвёртого и более высоких порядков обычно применяются иные методы вычисления, нежели использование готовых формул как для вычисления определителей второго и третьего порядков.

Один из методов вычисления определителей высших порядков – использование следствия из теоремы Лапласа. Это следствие позволяет разложить определитель по элементам некоторой строки или столбца. При этом вычисление определителя порядка сводится к вычислению определителей порядка. Именно поэтому такое преобразование именуют понижением порядка определителя.

Например, вычисление определителя четвёртого порядка сводится к нахождению четырёх определителей третьего порядка.

Допустим, нам задана квадратная матрица порядка, т.е. Вычислить определитель этой матрицы можно, разложив его по строке или по столбцу.

Зафиксируем некоторую строку, номер которой равен . Тогда определитель матрицы  можно разложить по выбранной строке, используя следующую формулу:

Аналог такой формулы существует и для столбцов. Формула для разложения определителя по столбцу выглядит следующим образом:

Правила, выраженные формулами данными, можно сформулировать так:

*определитель равен сумме произведений элементов некоей строки или столбца на алгебраические дополнения этих элементов.*

Для наглядности рассмотрим определитель четвёртого порядка, записанный в общем виде. Для примера разложим его по элементам четвёртого столбца (элементы этого столбца выделены жирным):

Аналогично, раскладывая, к примеру, по третьей строке, получим такую формулу для вычисления определителя:

Таким образом, вычислить определитель можно с помощью рекурсивного алгоритма, последовательно рассматривая каждую строку матрицы.

1. **Вычисление обратной матрицы с помощью алгебраических дополнений**

Нахождение обратной матрицы – задача, которая решается методом алгебраических дополнений. В ходе решения задачи необходимо находить определители, миноры и алгебраические дополнения и транспонировать матрицы.

Квадратная матрица называется невырожденной, если её определитель не равен нулю, и вырожденной, если её определитель равен нулю.

Обратная матрица может быть найдена только для квадратной матрицы. Естественно, обратная матрица также будет квадратной и того же порядка, что и данная матрица. Матрица, для которой может быть найдена обратная матрица, называется обратимой матрицей. Союзной с квадратной матрицей называется матрица того же порядка, элементами которой являются алгебраические дополнения соответствующих элементов определителя матрицы , транспонированной относительно матрицы .

Для неособенной квадратной матрицы обратной является матрица

где определитель матрицы , a матрица, союзная с матрицей

Пусть существует квадратная матрица :

Транспонированная относительно матрицы матрица получается, если из строк матрицы сделать столбцы, а из её столбцов - наоборот, строки, то есть заменить строки столбцами:

Далее необходимо вычислить минор и алгебраические дополнения к матрице. Рассмотрим этот процесс на примере.

Пусть есть квадратная матрица третьего порядка:

Её определитель:

Вычислим алгебраическое дополнение элемента , то есть элемента 2, стоящего на пересечении первой строки и второго столбца.

Для этого нужно сначала найти минор этого элемента. Он получается вычёркиванием из определителя строки и столбца, на пересечении которых стоит указанный элемент. В результате останется следующий определитель, который и является минором элемента :

Алгебраическое дополнение элемента  получим, если умножим , где  номер строки исходного элемента, а номер столбца исходного элемента, на полученный в предыдущем действии минор этого исходного элемента. Получаем алгебраическое дополнение элемента :

Таким образом, союзная матрица, составленная из алгебраических дополнений, умноженная на число, обратное определителю матрицы и является обратной для матрицы .

1. **Решение квадратной системы уравнений с помощью обратной матрицы**

Пусть для матрицы  порядка  существует обратная матрица . Умножим обе части матричного уравнения  слева на . Имеем . Так как для операции умножения матриц подходящих порядков характерно свойство ассоциативности, то последнее равенство можно переписать как, а по определению обратной матрицы  ( – единичная матрица порядка), поэтому

Таким образом, решение системы линейных алгебраических уравнений матричным методом определяется по формуле . Другими словами, решение СЛАУ находится с помощью обратной матрицы .

Мы знаем, что квадратная матрица  порядка  имеет обратную матрицу  только тогда, когда ее определитель не равен нулю. Следовательно, систему  линейных алгебраических уравнений снеизвестными можно решать матричным методом только тогда, когда определитель основной матрицы системы отличен от нуля.

**Инструкция для пользователя**

1. Исходный код размещен в репозитории на GitHub по ссылке: <https://github.com/sharphurt/MatrixMath_LR4_Lukoyanov>, его необходимо склонировать на свой компьютер.
2. Далее необходимо средствами вашей IDE подключить библиотеку к проекту и запустить сборку всего решения

В библиотеке существуют три основных класса: Vector, Matrix, InverseMatrixSolver. В классах Vector и Matrix представлены основные методы для работы с этими структурами данных, такие как: доступ по индексу к элементам структуры, поэлементные операции и основные математические операции, необходимые для работы алгоритма решения систем линейных уравнений методом обратной матрицы.

Далее представлен список методов, полей и свойств, к которым вы можете получить доступ (их реализация приведена в пункте «Приложение»):

**class “Matrix”:**

public double this[int row, int col]

public double Determinant { get => ... }

public Matrix Multiply(Vector vector)

public static Matrix operator \*(Matrix matrix, Matrix other)

public Matrix Inverse()

public Matrix ElementwiseOperation(Func<double, double> func)

**class “Vector”:**

public int Length => ...

public double this[int index]

public Vector ElementwiseOperation(Func<double, double> func)

public Matrix ToVerticalMatrix()

**class “InverseMatrixSolver”:**

public (LinearSystemSolveResult result, Vector answer) Solve(Matrix coeff, Vector vector)

# **Инструкция для программиста**

Исходный код представлен на GitHub по ссылке выше. Вы можете сделать форк репозитория и внести свой вклад в разработку этой библиотеки. Все методы в коде задокументированы и имеют пояснения в виде комментариев.

В библиотеке присутствует интерфейс **ILinearSystemSolver,** который необходимо имплементировать в классах, предназначенных для решения уравнений каким-либо способом.

Также, при необходимости, можно реализовывать новые методы для работы с матрицами и векторами, так как на данный момент существует лишь необходимый минимум для работы с алгоритмом решения системы линейных уравнений.

**Тестирование**

Для тестов была использована библиотека, предоставляющая функционал для юнит-тестирования NUnit. Тесты охватывают все основные методы для арифметических операций, операций сравнения и служебных методов. Обрабатывают также граничные случаи, такие как: матрица не содержит элементов или в качестве массива элементов передано значение , нулевой, положительный и отрицательный определитель, умножение матриц с несовпадающим количеством строк и столбцов, попытка нахождения решения системы линейных уравнений в случае, когда определитель основной матрицы равен нулю и другие. Всего в проекте 21 тестовый случай.

Некоторые выдержки из кода тестирования представлены в разделе «Приложение».

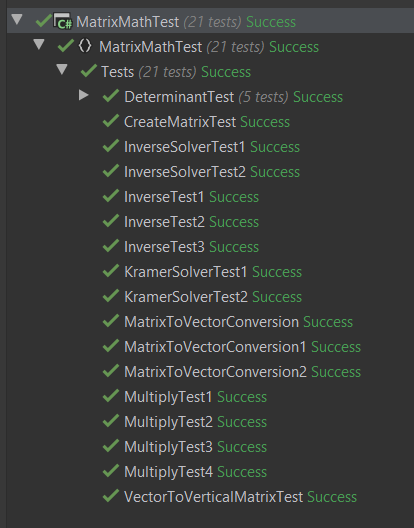


Рисунок 1. Результаты прохождения юнит-тестов - 100%

**Вывод**

В ходе работы мною были изучены и реализованы на языке C# алгоритмы работы с матрицами и векторами, алгоритм нахождения определителя матрицы, нахождения обратной матрицы и алгоритм нахождения решения системы линейных уравнений с помощью обратной матрицы. Результатом работы стало написание библиотеки, которую можно подключить к любому проекту и использовать в своих целях.

# **Литература**

1. Решение систем линейных алгебраических уравнений с помощью обратной матрицы [Электронный ресурс]. Режим доступа: <https://math1.ru/education/sys_lin_eq/invmatrix.html>
2. Нахождение обратной матрицы: три алгоритма и примеры [Электронный ресурс]. Режим доступа: <https://function-x.ru/return_matrix.html>
3. Понижение порядка определителя. Разложение определителя по строке (столбцу). [Электронный ресурс]. Режим доступа: <https://math1.ru/education/matrix/detr.html>
4. Формулы для вычисления определителей второго и третьего порядков. Примеры вычисления определителей второго и третьего порядков. [Электронный ресурс]. Режим доступа: <https://math1.ru/education/matrix/det23.html>

**Приложение**

Тесты

[TestFixture]  
public class Tests  
{  
 [Test]  
 public void CreateMatrixTest()  
 {  
 Assert.Throws(typeof(ArgumentException), () => new Matrix(new double[,] { }));  
 Assert.Throws(typeof(ArgumentException), () => new Matrix(null));  
 var elements = new double[,] {{1, 2, 3}, {5, 6, 7}};  
 Assert.AreEqual(elements, new Matrix(elements).Elements);  
 }  
  
 [TestCase("1 2 3 9 8 7 4 5 6", 0)]  
 [TestCase("2 10 8 2", -76)]  
 [TestCase("4 6 10 7 1 4 9 8 10 2 10 7 10 7 6 10 3 6 1 10 10 7 1 8 3", 52)]  
 [TestCase("6 4 10 1 10 5 1 9 3 2 9 10 6 2 6 2", 1718)]  
 [TestCase("10 2 6 1 5 10 3 6 7 2 9 7 4 6 8 9", -648)]  
 public void DeterminantTest(string matrix, double det)  
 {  
 var determinant = Utils.ArrayToSquareMatrix(matrix.Split(' ').Select(double.Parse).ToArray())  
 .Determinant;  
 Assert.AreEqual(determinant, det);  
 }  
  
 [Test]  
 public void MultiplyTest1()  
 {  
 var matrix1 = new Matrix(new double[,] {{3, 2, 1}, {6, 5, 4}, {3, 8, 1}});  
 var matrix2 = new Matrix(new double[,] {{12, 8, 1}, {4, 9, 2}, {3, 6, 0}});  
 var result = new double[,] {{47, 48, 7}, {104, 117, 16}, {71, 102, 19}};  
  
 Assert.AreEqual(result, (matrix1 \* matrix2).Elements);  
 }  
  
  
 [Test]  
 public void MultiplyTest2()  
 {  
 var matrix1 = new Matrix(new double[,] {{1, 12}, {38, 11}, {9, 21}});  
 var matrix2 = new Matrix(new double[,] {{13, 4, 1, 8, 4}, {3, 17, 21, 0, 9}});  
 var result = new double[,]  
 {  
 {49, 208, 253, 8, 112},  
 {527, 339, 269, 304, 251},  
 {180, 393, 450, 72, 225}  
 };  
  
 Assert.AreEqual(result, (matrix1 \* matrix2).Elements);  
 }  
  
  
 [Test]  
 public void MultiplyTest3()  
 {  
 var matrix1 = new Matrix(new double[,] {{1}, {38}});  
 var matrix2 = new Matrix(new double[,] {{13, 4}});  
 var result = new double[,]  
 {  
 {13, 4},  
 {494, 152}  
 };  
  
 Assert.AreEqual(result, (matrix1 \* matrix2).Elements);  
 }  
  
 [Test]  
 public void MultiplyTest4()  
 {  
 var matrix1 = new Matrix(new double[,] {{31, 209, 21}, {9, 8, 31}, {6, 72, 81}});  
 var matrix2 = new Matrix(new double[,] {{7}, {18}, {29}});  
 var result = new double[,] {{4588}, {1106}, {3687}};  
  
 Assert.AreEqual(result, (matrix1 \* matrix2).Elements);  
 }  
  
 [Test]  
 public void InverseTest1()  
 {  
 var matrix = new Matrix(new double[,]  
 {  
 {43, 129, 9},  
 {67, 13, 23},  
 {0, 191, 18}  
 });  
  
 var inverse = new Matrix(new double[,]  
 {  
 {-4159, -603, 2850},  
 {-1206, 774, -386},  
 {12797, -8213, -8084}  
 }).ElementwiseOperation(e => e / -219238.0);  
  
 Assert.AreEqual(inverse, matrix.Inverse());  
 }  
  
 [Test]  
 public void InverseTest2()  
 {  
 var matrix = new Matrix(new double[,]  
 {  
 {98, 103, 887},  
 {301, 345, 13},  
 {39, 91, 3}  
 });  
  
 var inverse = new Matrix(new double[,]  
 {  
 {-148, 80408, -304676},  
 {-396, -34299, 265713},  
 {13936, -4901, 2807}  
 }).ElementwiseOperation(e => e / 12305940.0);  
  
 Assert.AreEqual(inverse, matrix.Inverse());  
 }  
  
 [Test]  
 public void InverseTest3()  
 {  
 var matrix = new Matrix(new double[,] {{1, 2}, {3, 6}});  
 Assert.Throws(typeof(ArgumentException), () => matrix.Inverse());  
 }  
  
 [Test]  
 public void KramerSolverTest1()  
 {  
 var matrix = new Matrix(new double[,] {{21, 82, 1}, {47, 19, 37}, {8, 91, 12}});  
 var vector = new Vector(new double[] {81, 3, 54});  
 var result = new Vector(new double[] {-94128, -60396, 143790}).ElementwiseOperation(e => e / -83770.0);  
  
 var (kramerResult, answer) = new KramerSolver().Solve(matrix, vector);  
 Assert.AreEqual(kramerResult, LinearSystemSolveResult.**Single**);  
 Assert.AreEqual(answer, result);  
 }  
  
 [Test]  
 public void KramerSolverTest2()  
 {  
 var matrix = new Matrix(new double[,] {{1, 2}, {3, 6}});  
 var vector = new Vector(new double[] {8, 12});  
  
 var (result, answer) = new KramerSolver().Solve(matrix, vector);  
 Assert.AreEqual(result, LinearSystemSolveResult.**InfinityOrAbsence**);  
 Assert.AreEqual(answer, new Vector(0));  
 }  
  
 [Test]  
 public void MatrixToVectorConversion()  
 {  
 var matrix = new Matrix(new double[,] {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}});  
 Assert.Throws(typeof(InvalidOperationException), () => matrix.ToVector());  
 }  
  
 [Test]  
 public void MatrixToVectorConversion1()  
 {  
 var matrix = new Matrix(new double[,] {{1}, {2}, {3}, {4}, {5}}).ToVector();  
 var vector = new Vector(new double[] {1, 2, 3, 4, 5});  
 Assert.AreEqual(vector, matrix);  
 }  
  
 [Test]  
 public void MatrixToVectorConversion2()  
 {  
 var matrix = new Matrix(new double[,] {{1, 2, 3, 4, 5}}).ToVector();  
 var vector = new Vector(new double[] {1, 2, 3, 4, 5});  
 Assert.AreEqual(vector, matrix);  
 }  
  
 [Test]  
 public void InverseSolverTest1()  
 {  
 var matrix = new Matrix(new double[,] {{21, 82, 1}, {47, 19, 37}, {8, 91, 12}});  
 var vector = new Vector(new double[] {81, 3, 54});  
 var result = new Vector(new double[] {-94128, -60396, 143790}).ElementwiseOperation(e => e / -83770.0);  
  
 var (kramerResult, answer) = new InverseMatrixSolver().Solve(matrix, vector);  
 Assert.AreEqual(kramerResult, LinearSystemSolveResult.**Single**);  
 Assert.AreEqual(answer, result);  
 }  
  
 [Test]  
 public void InverseSolverTest2()  
 {  
 var matrix = new Matrix(new double[,] {{1, 2}, {3, 6}});  
 var vector = new Vector(new double[] {8, 12});  
  
 var (result, answer) = new InverseMatrixSolver().Solve(matrix, vector);  
 Assert.AreEqual(result, LinearSystemSolveResult.**InfinityOrAbsence**);  
 Assert.AreEqual(answer, new Vector(0));  
 }  
  
 [Test]  
 public void VectorToVerticalMatrixTest()  
 {  
 var vector = new Vector(new double[] {1, 2, 3, 4, 5});  
 var matrix = new Matrix(new double[,] {{1}, {2}, {3}, {4}, {5}});  
 Assert.AreEqual(matrix, vector.ToVerticalMatrix());  
 }  
}

Нахождение определителя

public double Determinant  
{  
 get {  
 var size = RowCount;  
 if (size == 1) return this[0, 0];  
 if (size == 2) return this[0, 0] \* this[1, 1] - this[0, 1] \* this[1, 0];  
  
 double determinant = 0;  
 var sign = 1;  
 for (var k = 0; k < size; ++k)  
 {  
 var mat0 = RowCoFactor(k);  
 determinant += sign \* this[k, 0] \* mat0.Determinant;  
 sign \*= -1;  
 }  
  
 return determinant;  
 }  
}

Нахождение обратной матрицы

public Matrix Inverse()  
{  
 if (RowCount != ColumnCount)  
 throw new ArgumentException(  
 "Обратная матрица существует только для квадратных, невырожденных, матриц.");  
  
 var matrix = new Matrix(RowCount, ColumnCount);  
 var determinant = Determinant;  
  
 if (Math.Abs(determinant) < 1e-9)  
 throw new ArgumentException(  
 "Обратная матрица существует только для квадратных, невырожденных, матриц.");  
  
 var sign = 1;  
  
 for (var i = 0; i < RowCount; i++)  
 for (var t = 0; t < ColumnCount; t++)  
 {  
 var tmp = Exclude(i, t);  
 matrix[t, i] = sign / determinant \* tmp.Determinant;  
 sign \*= -1;  
 }  
  
 return matrix;  
}

Умножение матрицы на вектор

public Matrix Multiply(Vector vector) =>  
 Multiply(this, new Matrix(vector.Length, 1, vector.Elements));

private static Matrix Multiply(Matrix matrix, Matrix other)  
{  
 if (matrix.ColumnCount != other.RowCount)  
 throw new ArgumentException("Matrices should have same size");  
  
 var result = new Matrix(matrix.RowCount, other.ColumnCount);  
  
 for (var row = 0; row < result.RowCount; row++)  
 for (var col = 0; col < result.ColumnCount; col++)  
 {  
 var tmp = .0;  
 for (var i = 0; i < matrix.ColumnCount; i++)  
 tmp += matrix[row, i] \* other[i, col];  
  
 result[row, col] = tmp;  
 }  
  
 return result;  
}

Решение системы линейных уравнений методом обратной матрицы

public (LinearSystemSolveResult result, Vector answer) Solve(Matrix coeff, Vector vector)  
{  
 var matrix = new Matrix(coeff.Elements);  
 var det = matrix.Determinant;  
 if (Math.Abs(det) < 1e-9)  
 return (LinearSystemSolveResult.**InfinityOrAbsence**, new Vector(0));  
  
 var result = matrix.Inverse().Multiply(vector).ToVector();  
 return (LinearSystemSolveResult.**Single**, result);  
}