Министерство науки и высшего образования РФ

Федеральное государственное бюджетное

образовательное учреждение высшего образования

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Отчет по лабораторной работе №4

**Разработка библиотеки на языке программирования Java**

дисциплина «Технология разработки программного обеспечения»

|  |  |
| --- | --- |
| Выполнил: студент группы ИВТм-1301 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Савин Д.А. / |
|  |  |
| Проверил: доцент кафедры ЭВМ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Чистяков Г.А. / |

Киров 2022

**Цель**: Целью работы является получение практических навыков разработки эффективных алгоритмических библиотек на языке программирования Java.

**Задачи**:

1. Выбрать вариант задания и согласовать тему разработки с преподавателем.
2. Реализовать библиотеку.
3. Разработать несколько типичных сценариев использования библиотеки.

Вариант №1. Разработать библиотеку для работы с матрицами, реализовать несколько матричных алгоритмов.

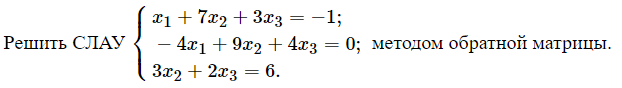
Код библиотеки:

|  |
| --- |
| **package** MatrLib;  **import** **java.math.BigDecimal**;  **import** **java.math.RoundingMode**;  **public** **class** **MatrixLibrary**  {  **public** **static** **int** SCALE = **3**;  /\*\*  \* @param source исходная матрица.  \* @return Детерминант матрицы.  \*/  **public** **static** **float** **GetDeterminant**(**float**[][] source)  {  **int** n = source.length;  **if** (n == **1**) **return** source[**0**][**0**]; //Вернуть единственный элемент матрицы  **float** result = **0**;  **for** (**int** i = **0**; i < n; i++)  {  **if** ((i % **2**) == **0**)  {  result += source[**0**][i] \* GetDeterminant(GetMinor(source, i, **0**));  }  **else**  {  result -= source[**0**][i] \* GetDeterminant(GetMinor(source, i, **0**));  }  }  **return** result;  }  /\*\*  \* @param source исходная матрица.  \* @param x строка.  \* @param y столбец.  \* @return Матрица без строки x и столбца y.  \*/  **public** **static** **float**[][] **GetMinor**(**float**[][] source, **int** x, **int** y)  {  **int** n = source.length;  **int** m = source[**0**].length;  **float**[][] minorMatrix = **new** **float**[n-**1**][m-**1**];  **for** (**int** i = **0**, ln = **0**; i < n; i++)  {  **for** (**int** j = **0**, cm = **0**; j < m; j++)  {  **if** (!((i == y) || (j == x)))  {  minorMatrix[ln][cm] = source[i][j];  cm++;  **if** (cm == n - **1**)  {  ln++;  }  }  }  }  **return** minorMatrix;  }  /\*\*  \* O(n^2)\*O\_determinant  \* @param source Исходная двумерная матрица.  \* @return Обратная матрица, полученная с помощью метода алгебраических дополнений.  \*/  **public** **static** **float**[][] **InverseMatrix**(**float**[][] source, **boolean** scaling) **throws** Exception  {  **int** n = source.length;  **if** (n != source[**0**].length) **throw** **new** Exception("не квадратная матрица");  **float**[][] matrix = **new** **float**[n][n];  **float** determinant = GetDeterminant(source); //Получение определителя входной матрицы  source = Transpose(source); //Транспонирование матрицы  **for** (**int** i = **0**; i < n; i++)  {  **for** (**int** t = **0**; t < n; t++)  {  **float**[][] tmpMatrix = GetMinor(source, i, t); //Получение матрицы меньшего порядка для вычисления её определителя  matrix[t][i] = (**float**) ((**1** / determinant) \* Math.pow(-**1**, i + t) \* GetDeterminant(tmpMatrix)); //Вычисление присоединённой матрицы и деление на определитель  **if** (scaling) matrix[t][i] = (BigDecimal.valueOf(matrix[t][i]).setScale(SCALE,RoundingMode.HALF\_DOWN)).floatValue();  }  }  **return** matrix;  }  /\*\*  \* @param source Исходная матрица.  \* @return Транспонированная матрица.  \*/  **public** **static** **float**[][] **Transpose**(**float**[][] source)  {  **int** n = source.length;  **int** m = source[**0**].length;  **float**[][] tmpMatrix = **new** **float**[n][m];  **for** (**int** i = **0**; i < n; i++)  {  **for** (**int** j = **0**; j < m; j++)  {  tmpMatrix[j][i] = source[i][j];  }  }  **return** tmpMatrix;  }  /\*\*  \* O(n^3)  \* Только квадратные матрицы.  \* @param source исходная матрица.  \* @return Обратная матрица, полученная с помощью метода Гаусса-Жордано.  \*/  **public** **static** **float**[][] **Jordan\_Gauss**(**float**[][] source, **boolean** scaling) **throws** Exception  {  **int** n = source.length;  **if** (n != source[**0**].length) **throw** **new** Exception("Неквадратная матрица");  **float**[][] e = **new** **float**[n][n]; //Создание единичной матрицы  **for** (**int** i = **0**; i < n; i++)  {  **for** (**int** j = **0**; j < n; j++)  {  e[i][j] = (i == j) ? **1** : **0**; //Заполнение единичной матрицы  }  }  **float** element;  **for** (**int** k = **0**; k < n; k++) //Операции получения нулей ниже главной диагонали  {  **if** (source[k][k] == **0**) //На каждой итерации берётся диагональный элемент  {  **boolean** swaped = **false**;  **for** (**int** i = k + **1**; (i < n) && !swaped; i++) //Поиск строки с не нулевым элементом  {  **if** (source[i][k] != **0**) //Не нулевой диагональный элемент найден  {  Swap(source, k, i);//Меняются местами строки k и i  Swap(e, k, i);//Нельзя забывать про единичную матрицу  swaped = **true**; //Выход из цикла  }  }  }  element = source[k][k];  **for** (**int** j = **0**; j < n; j++) //Текущая строка делится на этот элемент чтобы получить единицу на диагонали этой строки  {  source[k][j] /= element; //Операции проводятся как с основной матрицей, так и с единичной  e[k][j] /= element;  }  **for** (**int** i = k + **1**; i < n; i++)  {  element = source[i][k];  **for** (**int** j = **0**; j < n; j++)  {  /\*Из i-й строки матрицы вычитается k-я строка умноженная на элемент, который приводится к нулю.  k-я строка имеет в этот момент в столбце элемента единицу, что позволяет при вычитании этого  элемента, умноженного на единицу, из самого себя получить нуль вне диагонали\*/  source[i][j] -= source[k][j] \* element;  e[i][j] -= e[k][j] \* element;  }  }  }  **for** (**int** k = n - **1**; k > **0**; k--) //Операции получения нулей выше главной диагонали  {  **for** (**int** i = k - **1**; i >= **0**; i--)  {  element = source[i][k];  **for** (**int** j = **0**; j < n; j++)  {  source[i][j] -= source[k][j] \* element;  e[i][j] -= e[k][j] \* element;  }  }  }  **if** (scaling)  {  **for** (**int** i = **0**; i < n; i++)  {  **for** (**int** j = **0**; j < n; j++)  {  e[i][j] = (BigDecimal.valueOf(e[i][j]).setScale(SCALE, RoundingMode.HALF\_DOWN)).floatValue();  }  }  }  **return** e; //Возвращает единичную матрицу после всех преобразований  }  **static** **void** **Swap**(**float**[][] source, **int** i, **int** j) //Меняет местами строки i и j местами  {  **int** n = source.length;  **float**[] tmpMatrix = **new** **float**[n];//Временная матрица, чтобы записать в неё строку  **for** (**int** k = **0**; k < n; k++)  {  tmpMatrix[k] = source[i][k]; //Строка i будет храниться во временной матрице  source[i][k] = source[j][k]; //Запись строки j в строку i  }  //Запись строки i в строку j  System.arraycopy(tmpMatrix, **0**, source[j], **0**, n);  }  /\*\*  \* O(n^3)  \* @param source\_1 исходная матрица[x][y].  \* @param source\_2 исходная матрица[y][k].  \* @return Произведение матриц[x][k].  \*/  **public** **static** **float**[][] **MultiplyMatrix**(**float**[][] source\_1, **float**[][] source\_2) **throws** Exception  {  **int** m = source\_1.length;  **int** n = source\_2[**0**].length;  **int** o = source\_2.length;  **if** (source\_1[**0**].length != o) **throw** **new** Exception("Неверный размер матриц.");  **float** [][] result = **new** **float**[m][n];  **for** (**int** i = **0**; i < m; i++) {  **for** (**int** j = **0**; j < n; j++) {  **for** (**int** k = **0**; k < o; k++) {  result[i][j] += Math.round(source\_1[i][k] \* source\_2[k][j]);  }  }  }  **return** result;  }  } |

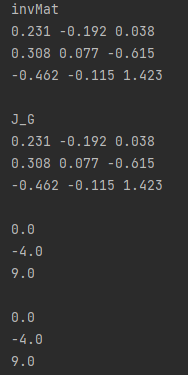
Доступные методы:

1. Нахождение обратной матрицы с помощью метода алгебраических дополнений.
2. Нахождение обратной матрицы с помощью метода Гаусса-Жордано.
3. Нахождение детерминанта матрицы.
4. Транспонирование матрицы.
5. Перемножение матриц.

Сценарий использования:



|  |
| --- |
| **import** **java.util.Random**;  **import** **static** MatrLib.MatrixLibrary.\*;  **import** **static** java.lang.System.out;  **public** **class** **Main**  {  // SLAU  // x1 + 7x2 + 3x3 =-1  // -4x1 + 9x2 + 4x3 = 0  // 0x1 + 3x2 + 2x3 = 6  **static** **float** [][]A = { {**1**,**7**,**3**}, {-**4**,**9**,**4**}, {**0**,**3**,**2**} };  **static** **float** [][]B = { {-**1**}, {**0**}, {**6**} };  **static** Random random = **new** Random();  **public** **static** **void** **main**(String[] args) **throws** Exception  {  SCALE = **5**;  **float**[][] invMat = InverseMatrix(A, **true**);  **float**[][] J\_G = Jordan\_Gauss(A, **true**);  out.println("invMat");  showMatrix(invMat);  out.println("J\_G");  showMatrix(J\_G);  showMatrix(MultiplyMatrix(invMat, B));  showMatrix(MultiplyMatrix(J\_G, B));  }  **public** **static** **void** **showMatrix**(**float** [][]matrix)  {  **for** (**float**[] doubles : matrix)  {  **for** (**int** j = **0**; j < matrix[**0**].length; j++)  {  out.print(doubles[j] + " ");  }  out.println();  }  out.println();  }  **public** **static** **float** [][] fillMatrix (**float** [][]matrix)  {  **for** (**int** i=**0**; i< matrix.length; i++)  {  **for** (**int** j=**0**; j<matrix[**0**].length; j++)  matrix[i][j] = random.nextInt(**100**);  }  **return** matrix;  }  } |



Вывод:

Благодаря данной библиотеке не требуется тратить время на изучение и реализацию алгоритмов нахождения обратной матрицы и других вспомогательных алгоритмов.