Вариант 19

Задание 5 (метод 4, задача 2)

Задание:

Задание 5. Итерационные методы для разреженных СЛАУ особого вида

1. Написать программу, которая при данном решает СЛАУ указанным в варианте методом. Здесь разреженные матрицы размерности из списка 2 (см. ниже), указанные в варианте.

* Матрицу следует либо хранить в одном из форматов для разреженных матриц, либо сразу реализовать итерационный метод, учитывая известную структуру матрицы. Хранить в памяти матрицу целиком со всеми нулями запрещено!
* Вектор b выбирать таким образом, чтобы он соответствовал некоторому заранее заданному решению.
* Критерий остановки итераций: 

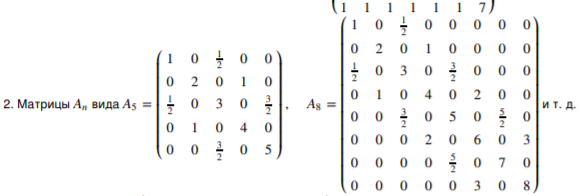
2. Подвердить правильность работы программы на примере нескольких СЛАУ размерности 5-10.

3. Построить диаграмму сходимости (общую) для n = 100, 1000, 10000.

4. Построить диаграмму, в которой по оси абсцисс изменяется n = [10k/2], k = 1, … ,12, а на оси ординат отложено время работы, которое требуется, чтобы норма невязки не превышала 10-8.

Метод 4: Симметричный метод Гаусса-Зейделя

Задача 2:



Решение

1. Для решения реализовал алгоритм Гаусса-Зейделя и класс для хранения матрицы моего вида. В самом начале по невнимательности реализовал класс для хранения симметричной ленточной матрицы. Но позже понял, что из-за особенностей приведенной матрицы можно полностью отказаться от хранения каких-либо элементов.

Класс для вычисления по алгоритму адаптирован для работы с классами матриц реализующих интерфейс содержащий методы getLength() и get(int i, int j)

1. Результат работы программы на матрице размером 7х7

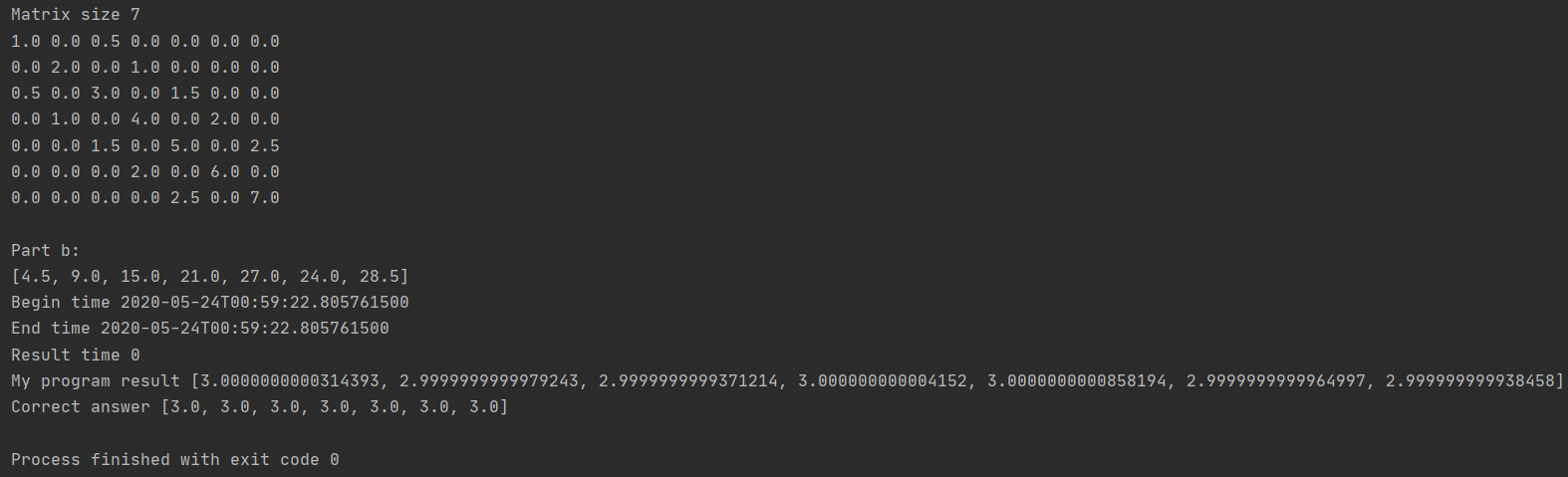


Рисунок 1 Скриншот работы программы на матрице 7x7

Результат работы программы на матрице размером 10х10

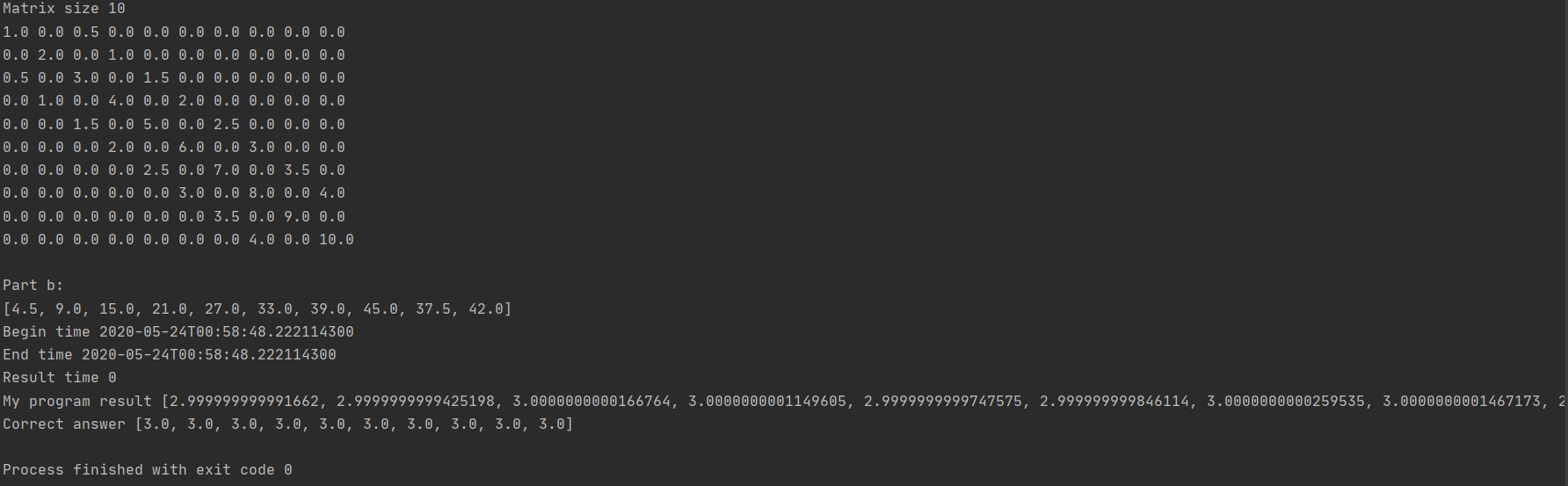


Рисунок 2 Скриншот работы программы на матрице 10x10

3)Результаты вычислений:

|  |  |  |  |
| --- | --- | --- | --- |
| Итерации\размер | 100 | 1000 | 10000 |
| 100 | 3.473412575799601E-5 | 0.014068198966513472 | 0.15701660161237757 |
| 200 | 2.1580030663761773E-8 | 0.003464079135167473 | 0.045343283639246314 |
| 300 | 1.3542148724457002E-11 | 0.0013598100550964538 | 0.021946135847971428 |
| 500 | 1.5781608441359134E-13 | 3.194417874235375E-4 | 0.008750053767643104 |

Диаграмма сходимости:

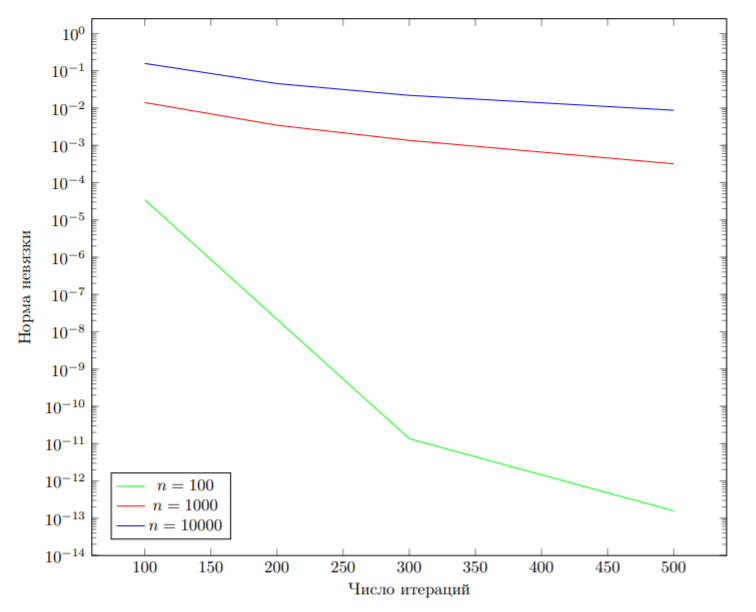


Рисунок 3 Диаграмма сходимости

4)Диаграмма по следующим данным:

|  |  |  |
| --- | --- | --- |
| K | Размер матрицы | Время вычисления при норме не более чем 10-8 |
| 1 | 3 | 58200 |
| 2 | 10 | 177700 |
| 3 | 31 | 2250400 |
| 4 | 100 | 10314500 |
| 5 | 316 | 35070800 |
| 6 | 1000 | 416553600 |
| 7 | 3162 | 7350379500 |
| 8 | 10000 | 143151055800 |
| 9 | 31622 | 2807320614800 |

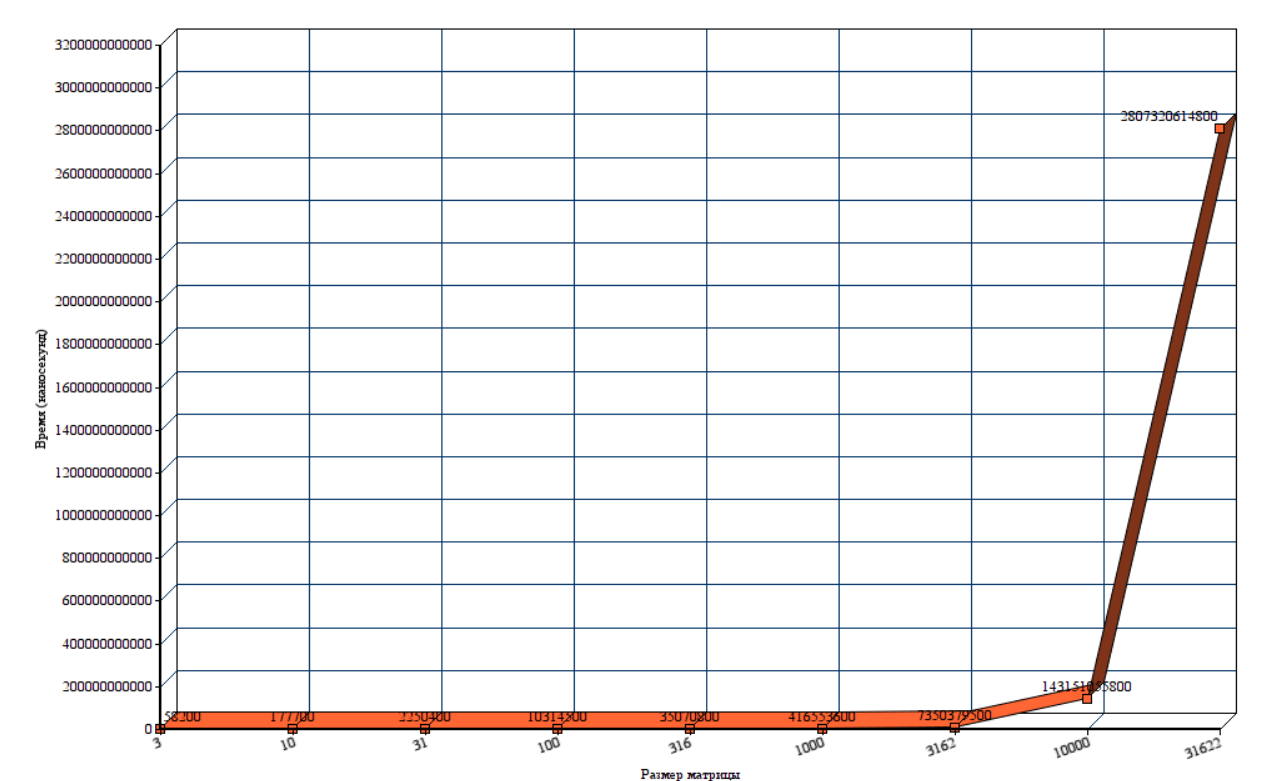


Рисунок 4 Временная диаграмма

Код программы на Java:

Класс для вычисления по методу Гаусса-Зейделя

public class ApliedGouseSeidel {  
 private final int MAX\_ITERATION;  
 private final double EPSILON;  
 private final Matrixable a;  
 private final double[] b;  
  
 private double lastInfelicity;  
  
 public ApliedGouseSeidel(int maxIteration, double epsilon, Matrixable a, double[] b) {  
 this.a = a;  
 this.b = b;  
 this.EPSILON = epsilon;  
 this.MAX\_ITERATION = maxIteration;  
 }  
  
 public static double[] getPartB(Matrixable a, double[] answer){  
 return *multiplyMatrix*(a, answer);  
 }  
  
 public double getLastInfelicity() {  
 return lastInfelicity;  
 }  
  
 //region straight  
  
 public double[] countStraight(double[] startX) {  
  
 var ans = Arrays.*copyOf*(startX, startX.length);  
  
 for (int i = 0; i < MAX\_ITERATION && !isConverge(ans); i++) {  
 ans = straightIteration(ans);  
 }  
  
 return ans;  
 }  
  
 */\*\*  
 \** ***@return*** *new x values  
 \*/* private double[] straightIteration(double[] previousValues) {  
 double[] ans = new double[previousValues.length];  
  
 for (int i = 0; i < a.getLength(); i++) {  
  
 ans[i] = 1.0 / a.get(i,i) \* (b[i] - getSumBefore(ans, i) - getSumAfter(previousValues, i));  
  
 }  
  
 return ans;  
 }  
  
 private double getSumBefore(double[] newValues, int iteration) {  
 var sum = 0.0;  
 for (int i = 0; i < iteration; i++) {  
 sum+= a.get(iteration, i) \* newValues[i];  
 }  
 return sum;  
 }  
  
 private double getSumAfter(double[] previousValues, int iteration){  
 var sum = 0.0;  
 for (int i = iteration +1; i < previousValues.length; i++) {  
 sum += a.get(iteration,i) \* previousValues[i];  
 }  
 return sum;  
 }  
  
 //endregion  
  
 //region reverse  
 public double[] countReverse(double[] startX) {  
  
 var ans = Arrays.*copyOf*(startX, startX.length);  
  
 for (int i = 0; i < MAX\_ITERATION && !isConverge(ans); i++) {  
  
 ans = reverseIteration(ans);  
  
 }  
  
 return ans;  
 }  
  
 private double[] reverseIteration(double[] previousValues) {  
 double[] ans = new double[previousValues.length];  
  
 for (int i = previousValues.length -1; i >=0 ; i--) {  
  
 double sumBeforeReverse = getSumBefore(previousValues, i);  
 double sumAfterReverse = getSumAfter(ans, i);  
  
 ans[i] = 1.0 / a.get(i,i) \* (b[i] - sumBeforeReverse - sumAfterReverse);  
  
 }  
 return ans;  
 }  
  
  
 //endregion  
  
 //region symmetric  
  
 public double[] countSymmetric(double[] startX){  
  
 var ans = Arrays.*copyOf*(startX, startX.length);  
  
 for (int i = 0; i < MAX\_ITERATION && !isConverge(ans); i++) {  
 ans = straightIteration(ans);  
 ans = reverseIteration(ans);  
 }  
  
 return ans;  
 }  
  
 //endregion  
  
 //region isConverge  
  
 private boolean isConverge(double[] x){  
 double[] axMatrix = *multiplyMatrix*(a, x);  
 double infelicity = countInfelicity(axMatrix, b);  
 lastInfelicity = infelicity;  
 return infelicity < EPSILON;  
 }  
  
 static private double[] multiplyMatrix(Matrixable a, double[] b) {  
 var ans = new double[a.getLength()];  
 for (int i = 0; i < a.getLength(); i++) {  
 double sum = 0;  
 for (int j = 0; j < a.getLength(); j++) {  
 sum += a.get(i,j) \* b[j];  
 }  
 ans[i] = sum;  
 }  
 return (ans);  
 }  
  
 private double countInfelicity(double[] ax, double[] bx ){  
 if(ax.length!=bx.length)  
 throw new RuntimeException("ax.length!= bx.length");  
 double sum=0;  
 for(int i=0;i<ax.length;i++){  
 sum+=(ax[i]-bx[i])\*(ax[i]-bx[i]);  
 }  
 return Math.*sqrt*(sum);  
 }  
  
 //endregion  
}

Класс для матрицы, определенного в задании вида:

public class TaskMatrix implements Matrixable {  
 private int rank;  
  
 public TaskMatrix(int rank) {  
 this.rank = rank;  
 }  
  
 public int getLength() {  
 return rank;  
 }  
  
 public double get(int i, int j) {  
 if(i>=rank || j >= rank)  
 throw new IndexOutOfBoundsException();  
  
 if (j < i) {  
 int t = i;  
 i = j;  
 j = t;  
 }  
  
 if (i == j) return i + 1;  
  
 if(j -i ==2) return (j-1)/2.0;  
  
 return 0.0;  
 }  
  
  
 @Override  
 public String toString() {  
  
 var ans = new StringBuilder();  
  
 for (int i = 0; i < getLength(); i++) {  
 for (int j = 0; j < getLength(); j++) {  
 ans.append(get(i, j)).append("\t");  
 }  
 ans.append("\n");  
 }  
  
 return ans.toString();  
 }  
}

Интерфейс, которому должны следовать матрицы, передаваемые в алгоритм Гаусса –Зейделя

public interface Matrixable {  
 double get(int i, int j);  
 int getLength();  
}

Класс main

public static void main(String[] args) {  
  
  
 int size = 5;  
 System.*out*.println("Matrix size " + size);  
  
 TaskMatrix matrix = new TaskMatrix(size);  
 var benchAns = *fillAnswer*(matrix.getLength(), 3);  
 double[] partB = ApliedGouseSeidel.*getPartB*(matrix, benchAns);  
  
 System.*out*.println(matrix);  
 System.*out*.println("Part b:");  
 System.*out*.println(Arrays.*toString*(partB));  
  
 var ags = new ApliedGouseSeidel(2000000000, 0.01, matrix, partB);  
  
  
 var beginDate = System.*nanoTime*();  
 double[] ans = ags.countSymmetric(*fillAnswer*(matrix.getLength(), 0));  
 var endDate = System.*nanoTime*();  
  
 System.*out*.println("Begin time " + beginDate);  
 System.*out*.println("End time " + endDate);  
  
 System.*out*.println("Result time " + (endDate - beginDate));  
  
 System.*out*.println("My program result " + Arrays.*toString*(ans));  
 System.*out*.println("Correct answer " + Arrays.*toString*(benchAns));  
}  
  
private static double[] fillAnswer(int n, double val) {  
 var answer = new double[n];  
 for (int i = 0; i < n; i++) {  
 answer[i] = val;  
 }  
 return answer;  
}

Результаты работы программы:

