TouchMiner – Mining for Usability

Source code for v. 0.03 Alpha

Abstract

TouchMiner is a command-line data mining tool for analyzing sequential patterns of characters in a preprocessed text data file. This is the source code for alpha version 0.03. Note that **TouchMiner is not Open Source** and using this code to fork TouchMiner is prohibited.

How the code is organized

The source contains two classes: **AnalyzeMatrices** and **SmallMatrices**. Most of the operations are placed in the latter, since for the moment, we only have the analysis for text patterns made of small characters.

Methods and subroutines are commented quite thoroughly and there should be no difficulty in figuring out the relationships between entities. Also note that this is by no means a "final version" and there is much work to be done.

1 AnalyzeMatrices class

```
1
2  /**
3   * The package for Phase one of the research, i.e. meaning the creation of
4   * text pattern matrices and printing the data into text files for plotting.
5   * The minimization problem (creating the actual keyboard layout) is addressed
6   * separately
7   *
8   * @author rierjarv
9   * @version 12.2.2012
10  *
11  */
12 package phaseone;
13
14 import java.io.BufferedReader;
15 import java.io.File;
16 import java.io.FileNotFoundException;
17 import java.io.FileReader;
18 import java.io.IOException;
19 import java.util.InputMismatchException;
20 import java.util.Scanner;
21
22 /**
23  * Class for running the TouchMiner program and to perform basic utilities
24  *  -reading a file for analysis
25  *  -name of files to write to
26  *  -whether analysis operation is done or not
27  *
28  * @author rierjarv
29  * @version 0.03 14.2.2012
30  *
31  * TODO:
32  *   JUnit testing
33  *   parse utilities (zvalues files)
34  *   most probable patterns analysis
35  *   keyboard construction, full character analysis,
36  *   special char analysis etc.
37  */
38 public class AnalyzeMatrices {
39
40    // read user input
41    private final static Scanner reader = new Scanner(System.in);
```

```
42
43    private boolean smallDone = false; //if smallanalysis done changed to true
44 // private boolean filterFlag = false; // if files written with filter
45 // private boolean fullDone = false;
46 // private boolean wordsDone = false;
47
48    // text file to read for analysis
49    private String readFile;
50
51    // folder where smallChar analysis files are saved: specified by user
52    private String writeSmallFolder;
53
54 // private String writeFullFolder; // directory, save full analysis TODO
55 // private String writeWordsFolder; // directory, save words analysis TODO
56
57    // root directory where all files are saved
58    private static final String rootDirectory = "C:\\TouchMiner\\";
59
60    // subdirectories for different dimensions.
61    protected static final String subDir1D = "\\1D";
62    protected static final String subDir2D = "\\2D";
63    protected static final String subDir3D = "\\3D";
64    protected static final String subDir4D = "\\4D";
65
66    /** Constructs AnalyzeMatrices object */
67    public AnalyzeMatrices () {}
68
69    /** high-level decisions based on user input */
70     private static void start() {
71       AnalyzeMatrices matrices = new AnalyzeMatrices();
72       boolean quit = false;
73       while (!quit) {
74          quit = matrices.choices();
75       }
76    }
77
78     /** choices of main menu: see start() */
79     private boolean choices () {
80       int choice = intro();
81       if ( choice == 0 ) {
82          System.out.println("\nProgram terminated. Thank You for " +
83              "using TouchMiner.");
84          return true;
85       }
86       else if ( choice == 1 ) help();
87       else if ( choice == 2 && smallDone == false ) {
88          return analyzeSmall(); // return carryOn() value
89       }
90       else if ( choice == 2 && smallDone == true) {
91          System.out.println("\nSmall character analysis already done.");
92       }
93       else return false;
94       return false;
95     }
96
97    /**
98     * Prints the introduction text and asks for user interaction
```

```
99      * returns -1 if input does not exist.
100     * returns 0 and terminates program if user input makes no sense
101     * @return number that user chose for interaction
102     */
103    private int intro() {
104       introText();
105       return introChoose();
106    }
107
108    /** show introtext to user: see intro()  */
109    private void introText() {
110       System.out.println("\n----------------------------------------\n"+
111       "[[[ TouchMiner - Mining for Usability ]]]\n--------------------"+
112       "--------------------\n");
113       System.out.println("Version 0.03 alpha\n");
114       System.out.println("Choose action from the list below:\n" +
115             "\n 0 == Quit" +
116             "\n 1 == Help");
117       if ( !smallDone ) System.out.println(" 2 == Small characters raw" +
118             " data analysis");
119       if ( smallDone ) System.out.println(" 3 == Do nothing");
120       System.out.print("\n>> ");
121    }
122
123    /** actions for user choices: see introText() */
124    private int introChoose () {
125       try {
126          int choice = reader.nextInt();
127          if ( choice == 0 || choice == 1 ) return choice;
128          if ( choice == 2 && this.smallDone == false ) return choice;
129          if ( choice == 2 && this.smallDone == true ) return -1;
130          if ( choice == 3 && smallDone == false ) {
131             System.out.println("\nChoice does not exist! Try again...");
132             return -1; // loop
133          }
134          if ( choice == 3 && smallDone == true ) return choice;
135          if ( (choice < 0) || (choice > 3) ) {
136             System.out.println("\nChoice does not exist! Try again...");
137             return -1;
138          }
139       } catch (InputMismatchException e) {
140          System.out.println("\n\nYour input was incorrect.");
141          return 0; // also in child methods
142       }
143       return 0;
144    }
145
146     /**
147      * small characters analysis
148      * @return false if user decides to quit
149      */
150    private boolean analyzeSmall() {
151       smallDone = true; // small analysis flag is on
152       SmallMatrices smallM = new SmallMatrices();
153       createDirectories();
154       setReadFile(); // TODO: refct setReadFile()
155         return smallReadWrites(smallM);
```

```
156     }
157
158     /** details of analyzeSmall() method */
159     private boolean smallReadWrites (SmallMatrices smallM) {
160        smallRead(smallM);
161        return smallWrite(smallM);
162     }
163
164     /** reading the smallAnalysis file and doing calculations */
165     private void smallRead (SmallMatrices smallM) {
166        try {
167           String file = read(readFile); // read in Analyzematrices
168           smallM.analysis(file); // if we continue, return true
169        } catch (Exception e1) {
170           e1.printStackTrace();
171           System.out.println("Problem reading file (analyzeSmall).");
172        }
173     }
174
175     /** writing the smallAnalysis file and results */
176     private boolean smallWrite (SmallMatrices smallM) {
177        try {
178           smallM.write(writeSmallFolder);
179           System.out.println("Analysis done! Files saved in "+
180                 writeSmallFolder + "\n");
181           if ( !carryOn() ) return true;
182           else return false;
183        } catch (Exception e) {
184           System.out.println("Problem writing file (analyzeSmall).");
185           e.printStackTrace();
186        }
187        return false;
188     }
189
190     /**
191       * Checks whether users wants to continue using the program
192       * @return true if continues, false if does not continue
193       */
194     protected static boolean carryOn () {
195        System.out.println("Do you wish to continue?  y/n");
196        System.out.print("\n>> ");
197        String answer = reader.nextLine();
198        if ( answer.equals("n") ) {
199           System.out.println("\nProgram terminated. Thank You for "+
200                "using TouchMiner.");
201           return false;
202        }
203        else if ( answer.equals("y") ) return true;
204        else return true;
205     }
206
207     /**
208      * Prints help, which has short comments on program features.
209      * For more information, see TouchMiner manual.
210      */
211     private static void help()  {
212        String msg ="\n(Press any number to go to previous menu...)";
```

```
213         while ( helpMenu(msg) ); // until user inputs 0
214     }
215 
216     /** show top level of help menu */
217     protected static boolean helpMenu (String msg) {
218         showHelpMenu();
219         return makeHelpChoice(msg);
220     }
221 
222     /** shows help menu choices */
223     private static void showHelpMenu () {
224         System.out.println("\n[TouchMiner help]\n" +
225                 "\nPlease choose one of the following:\n" +
226                 "\n 0 == Back to main menu" +
227                 "\n 1 == About TouchMiner" +
228                 "\n 2 == Analysis of small characters");
229         System.out.print("\n>> ");
230     }
231 
232     /** make a help menu choice */
233     private static boolean makeHelpChoice (String msg) {
234         try {
235             int input = reader.nextInt() ;
236             if ( input == 0 ) return false; // quit at once
237             else if ( input == 1 ) helpAbout(msg);
238             else if ( input == 2 ) helpSmallChar(msg);
239             return true; // if user input not 0
240         } catch (Exception e) {
241             return false;
242         }
243     }
244 
245     /** General information about TouchMiner */
246     private static boolean helpAbout (String msg) {
247         while (true) {
248             helpAboutText(msg);
249             return helpSubmenuBack();
250         }
251     }
252 
253     /** Prints the text in helpAbout() - About TouchMiner */
254     private static void helpAboutText (String msg) {
255         System.out.println("\n[1 - About TouchMiner]\n" +
256             "\nTouchMiner is a software for the analysis of text" +
257             " patterns.\n" +
258             "Created by Riku Jarvinen in spring 2012.\n\n" +
259             "Latest version can be downloaded at \n\n"+
260             " http://rikun.net/touchminer\n"+
261             "\nKeep in mind that you can only analyze one [large] "+
262             "text file\nthat should be preprocessed!\n");
263         System.out.println(msg);
264         System.out.print("\n>> ");
265     }
266 
267     /** choices for help submenu to go back to help main menu*/
268     private static boolean helpSubmenuBack () {
269         try {
```

```
270             @SuppressWarnings("unused")
271             int input = reader.nextInt();
272         } catch (Exception e) {
273             return false;
274         }
275         return false;
276     }
277 
278     /**
279      * Help menu for smallChars analysis
280      * @return false if user wants to go back, otherwise true
281      */
282     private static boolean helpSmallChar (String msg) {
283         while (true) {
284             helpSmallCharText(msg);
285             return helpSubmenuBack();
286         }
287     }
288 
289     /** Prints helpSmallChar() text*/
290     private static void helpSmallCharText(String msg) {
291         System.out.println("\n[2 - Analysis of small characters]\n");
292         System.out.println("Selecting this option creates text" +
293             " files containing:\n " +
294             "\n -frequencies of patterns of 2, 3 and 4 chars" +
295             "\n -corresponding normalized values.\n" +
296             "\nYou need to:\n" +
297             "\n -name the subfolder where the files are saved" +
298             "\n -give relative path to the text file used for" +
299             " analysis.\n"+
300             "\nTouchMiner will create the necessary folders " +
301             "in \"C:\\TouchMiner\\\"." +
302             "\n\nSee TouchMiner manual for more information.\n");
303         System.out.println(msg);
304         System.out.print("\n>> ");
305     }
306 
307     /**
308      * Read a file into a String
309      * @param file the name of the file to read
310      * @return line file contents as a string
311      * @throws Exception if something goes wrong
312      * TODO: create parse method that works with this one SmallMatrices
313      */
314     public static String read (String filename) throws Exception {
315         BufferedReader fi;
316         try {
317             fi = new BufferedReader(new FileReader(filename)); // read file
318         } catch (FileNotFoundException e) {
319             throw new Exception( e.getMessage() );
320         }
321         StringBuilder text = new StringBuilder();
322         String line ="";
323         try {
324             // read
325             while ( ( line = fi.readLine() ) != null ) {
326                 // if line is empty or commented with '#' do nothing
```

```
327             if ( line.isEmpty() || line.charAt(0) == '#' ) continue;
328             text.append(line);
329          }
330       } catch (IOException e) {
331          throw new Exception( "Problem reading file." + e.getMessage() );
332       } finally {
333          try {
334             fi.close();
335          } catch (IOException e) {
336             throw new Exception( "Problem close file." + e.getMessage() );
337          }
338       }
339       return text.toString();
340    }
341
342     /**
343      * Create directories for patterns analysis.
344      * If directory exists, user is prompted for a different dirname.
345      * Sets the name of writeSmallFolder to parent directory specified
346      * by user.
347      */
348     private void createDirectories () {
349       System.out.println("\nSpecify a directory name. It will be" +
350             " created under \"C:\\TouchMiner\\\"." +
351             "\nAll analysis files are saved there.");
352       System.out.print("\n>> ");
353       reader.nextLine(); // needed for some unspecified reason
354       writeSmallFolder = createParentDirectory(); // save directory
355       createSubDirectories(); // different dimensions
356     }
357
358     /**
359      * Creates the parent directory under which different pattern
360      * directories are created. This is saved as writeSmallFolder
361      * @return name of the parent directory
362      */
363     private static String createParentDirectory () {
364       String dirName = reader.nextLine();
365       StringBuilder sb = new StringBuilder(rootDirectory+dirName);
366       try {
367          boolean dirExists = ( new File( sb.toString() ) ).mkdirs();
368          if (dirExists) return giveDirName(sb);
369          return getNewDir(sb);
370       } catch (Exception e) {
371          System.out.println("Problem (createParentDirectory).");
372       }
373       return null; // this shouldn't happen unless exception
374     }
375
376     private static String giveDirName (StringBuilder sb) {
377       System.out.println("\nNew directory: " + "\""
378             + sb.toString() + "\"" + " created.\n");
379             return sb.toString();
380     }
381
382     private static String getNewDir (StringBuilder sb){
383       while ( true ) { // directory doesn't exist
```

```
384           String answer = promptNewDir(sb);
385           if ( answer.equals("y") ) {
386              return createNewParent(rootDirectory);
387           }
388           else if ( answer.equals("n") ) {
389              System.out.println("\nFiles will be overwritten...\n");
390              return sb.toString();
391           }
392           else System.out.println("\nYou didn't answer correctly.");
393        }
394     }
395
396     /** prompt user for new directory name: see createParentDirectory() */
397     private static String promptNewDir (StringBuilder sb) {
398        System.out.println("\nDirectory \"" + sb.toString() +
399              "\" already exists.\n" +
400              "Would You like to create a new directory?\n" +
401              "\n y = create a new directory" +
402              "\n n = overwrite existing files in " +
403              sb.toString() + "\n");
404        System.out.print("\n>> ");
405        String answer = reader.nextLine();
406        return answer;
407     }
408
409     /**
410      * if user answered yes in createParentDirectory to new parent
411      * directory creation, we do what is required of us...
412      * @return true if new parent directory creation was successful
413      */
414     private static String createNewParent (String rootDir) {
415        while (true) {
416           StringBuilder sc = new StringBuilder();
417           System.out.println("\nSpecify a new directory name " +
418                 "(it will be created under \"C:\\TouchMiner\\\")\n");
419           System.out.print("\n>> ");
420           String newDirName = reader.nextLine();
421           sc.append(rootDir+newDirName);
422           String newDir = sc.toString();
423           boolean newDirectory = ( new File(newDir) ).mkdirs();
424           if ( newDirectory ) {
425              System.out.println("\nNew directory: "
426                    + "\"" + newDir + "\"" + " created.\n");
427              return newDir;
428           }
429        }
430     }
431
432     /**
433      * Creates subdirectories. Since the creation of the parent directory
434      * already has error checking, these directories do not exist before and
435      * there should be no problem with their creation.
436      */
437     private void createSubDirectories () {
438        createSubDirectory(subDir1D);
439        createSubDirectory(subDir2D);
440        createSubDirectory(subDir3D);
```

```
441         createSubDirectory(subDir4D);
442       }
443
444       /** subdirectory for patterns */
445       private void createSubDirectory (String subDir) {
446           StringBuilder dir = new StringBuilder(writeSmallFolder);
447         dir.append(subDir);
448           @SuppressWarnings("unused") // error checking ?
449           boolean file =  new File( dir.toString() ).mkdir();
450       }
451
452      /**
453       * Set the name of the file to read, asks from user
454       * TODO: refactor, organize
455       * @param readfile name of the file to read
456       */
457      public void setReadFile() {
458         System.out.println("Specify relative path to text file (including"+
459               " the name of the file):\n");
460         System.out.print(">> ");
461         String filename = reader.nextLine();
462         StringBuilder sb = new StringBuilder(filename);
463         // check whether the given filename is empty
464         while ( sb.toString().isEmpty() ) {
465            System.out.println("\nYou did not specify a filename... " +
466                "Could You Please specify a filename to read?");
467            System.out.print("\n>> ");
468            String newFileName = reader.nextLine();
469            sb.append(newFileName);
470         }
471         // TODO: fix this error checking!
472         // check whether filename is a directory name
473         while ( sb.toString().charAt( sb.toString().length()-1 ) == '/' ) {
474               sb.setLength(0);
475               System.out.println("\nSpecify a filename instead of directory name!");
476               String fname = reader.nextLine();
477               sb.append(fname);
478         }
479         File file = new File(filename);
480         boolean exists = file.exists();
481         // check whether file exists
482         while (!exists) {
483            System.out.println("\nThe file \"" + sb.toString() + "\" does not exist!"+
484                " Specify a new filename.");
485            System.out.print("\n>> ");
486            sb.setLength(0);
487            String newFileName = reader.nextLine();
488            File newFile = new File(newFileName);
489            // if newFile exists, we add to sb and close the loop
490            if (newFile.exists()) exists = true;
491            sb.append(newFileName);
492         }
493         this.readFile = sb.toString();
494      }
495
496      /**
497       * Runs the TouchMiner program
```

```
498      * @param args not used
499      */
500     public static void main(String[] args) {
501         start();
502     }
503 }
```

2 Smallmatrices class

```
1  package phaseone;
2
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.io.FileOutputStream;
6  import java.io.PrintStream;
7  import java.util.Locale;
8  import java.util.StringTokenizer;
9
10 /**
11  * Creates matrices needed for analysis of small characters
12  * SmallMatrices object stores information about the number
13  * of patterns in different dimensions.
14  *
15  * Basic read/write utilities are also here except for read()
16  * in AnalyzeMatrices.
17  *
18  * @author rierjarv
19  * @version 14.2.2012
20  *
21  */
22 public class SmallMatrices {
23
24     private int[] table1D; // standard table for 1D
25     private int[] table1Dorg; // organized 1D table
26     private int[] chars1Dorg; // indices for organized 1D table
27     private double[] nTable1D; // normalized "int"-type table
28     private double[] nTable1Dorg; // normalized organized "double"-type table
29
30     // no organized entries for 2D-> make filter, (String,value) objects and
31     // organize wrt. value with quicksort or similar algorithm, same with
32     // 3D & 4D but this should be done in StringPatterns class
33     private int [][] table2D;
34     private double [][] nTable2D;
35     private int [] distributionTable2D;
36     private int[][][] table3D;
37     private double[][][] nTable3D;
38     private int [] distributionTable3D;
39     private int[][][][] table4D;
40     private double[][][][] nTable4D;
41     private int [] distributionTable4D;
42
43     public final int smallDim = 29; // #of analyzed small characters
44     private int smallPatterns1D; // #of char patterns
45     private int smallPatterns2D;
46     private int smallPatterns3D;
47     private int smallPatterns4D;
```

```
48
49      // multiplier for smallDim to determine division in distribution classes
50      // increasing this gives a smoother distribution
51      private final int distConst = 4;
52
53      // TODO: loops for writing all dims efficiently?
54      protected final static String norm1D = "1Dnorm";
55      protected final static String nzval1D = "1Dnzval";
56      protected final static String nlet1D = "1Dnlet";
57      protected final static String basic1D = "1Dindex";
58      protected final static String zval1D = "1Dzval";
59      protected final static String let1D = "1Dlet";
60
61      // 2D filenames
62      protected final static String norm2D = "2Dnorm";
63      protected final static String nzval2D = "2Dnzval";
64      protected final static String nlet2D = "2Dnlet";
65      protected final static String basic2D = "2Dindex";
66      protected final static String zval2D = "2Dzval";
67      protected final static String let2D = "2Dlet";
68
69      // 3D filenames
70      protected final static String norm3D = "3Dnorm";
71      protected final static String nzval3D = "3Dnzval";
72      protected final static String nlet3D = "3Dnlet";
73      protected final static String basic3D = "3Dindex";
74      protected final static String zval3D = "3Dzval";
75      protected final static String let3D = "3Dlet";
76
77      // 4D filenames
78      protected final static String norm4D = "4Dnorm";
79      protected final static String nzval4D = "4Dnzval";
80      protected final static String nlet4D = "4Dnlet";
81      protected final static String basic4D = "4Dindex";
82      protected final static String zval4D = "4Dzval";
83      protected final static String let4D = "4Dlet";
84
85      // attributes that are needed later a written here
86      protected final static String attributeFile = "attributes";
87      protected final static String distributionFile2D = "2Ddistribution";
88      protected final static String distributionFile3D = "3Ddistribution";
89      protected final static String distributionFile4D = "4Ddistribution";
90
91      // file extensions
92      protected final static String txt = ".txt";
93      protected final static String dat = ".dat";
94
95
96      /** constructs SmallMatrices object */
97      public SmallMatrices () {
98          table1D = new int[smallDim];
99          table1Dorg = new int[smallDim];
100         nTable1D =  new double[smallDim];
101          nTable1Dorg = new double [smallDim];
102         chars1Dorg = new int[smallDim];
103         table2D = new int[smallDim][smallDim];
104         nTable2D = new double[smallDim][smallDim];
```

```
105         distributionTable2D = new int[smallDim*distConst];
106         table3D = new int[smallDim][smallDim][smallDim];
107         nTable3D = new double[smallDim][smallDim][smallDim];
108         distributionTable3D = new int[smallDim*distConst];
109         table4D = new int[smallDim][smallDim][smallDim][smallDim];
110         nTable4D = new double[smallDim][smallDim][smallDim][smallDim];
111         distributionTable4D = new int[smallDim*distConst];
112         smallPatterns1D = 0;
113         smallPatterns2D = 0;
114         smallPatterns3D = 0;
115         smallPatterns4D = 0;
116     }
117 
118     /**
119      * Analysis of small characters
120      * @param data String to be analyzed
121      */
122     public void analysis (String data) {
123         StringTokenizer st = new StringTokenizer(data);
124         while (st.hasMoreTokens()) {
125             StringBuilder sb = new StringBuilder(st.nextToken());
126             sb.append(' '); String word = sb.toString();
127             findPatterns(word);
128         }
129         distributions();
130         normalize(); // can be done anywhere before write
131         organizeDescending1D (); // 1D chars organized
132         printToScreen ();
133     }
134 
135 // /** Tests normalization of patterns */
136 // public boolean normalizationTest () {
137 //     return true;
138 // }
139 
140     /**
141      * Perform pattern finding algorithms
142      */
143     private void findPatterns (String word) {
144         small1char(word);
145         small2char(word);
146         small3char(word);
147         small4char(word);
148     }
149 
150     /**
151      * small character analysis for single characters
152      * @param word word to be analyzed
153      */
154     public void small1char ( String word ) {
155         int j = 0;
156         for (int k = 0; k < word.length(); k++) {
157             char a = word.charAt(k);
158             if (getIndex(a) == -1) continue;
159             j = getIndex(a);
160             smallPatterns1D++;
161             table1D [j]++;
```

```
162         }
163     }
164 
165     /**
166      * search 2-character patterns
167      * @param word to be analyzed
168      */
169     public void small2char (String word) {
170         int[] mat = new int[2];
171         if (word.length() == 1) return; // individual chars not analyzed
172         else {
173             for (int i = 1, k = 0; i < word.length(); i++, k++) {
174                 char a = word.charAt(i);
175                 char b = word.charAt(k);
176                 if (getIndex(a) == -1 || getIndex(b) == -1) {
177                     mat[0] = 0; mat[1] = 0;
178                     continue;
179                 }
180                 mat[0] = getIndex(b); mat[1] = getIndex(a);
181                 smallPatterns2D++;
182                 table2D [ mat[0] ] [ mat[1] ]++;
183             }
184         }
185     }
186 
187     /**
188      * search 3-character patterns
189      * @param word to be analyzed
190      * */
191     public void small3char (String word) {
192         int[] mat = new int[3];
193         if (word.length() < 2) return;
194         else {
195             for (int i=2, j=1, k=0; i<word.length(); i++,j++,k++) {
196                 char a = word.charAt(i);
197                 char b = word.charAt(j);
198                 char c = word.charAt(k);
199                 if (getIndex(a) == -1 || getIndex(b) == -1 ||
200                         getIndex(c) == -1) {
201                     mat[0] = 0; mat[1] = 0; mat[2] = 0;
202                     continue;
203                 }
204                 mat[0] = getIndex(c); mat[1] = getIndex(b);
205                 mat[2] = getIndex(a);
206                 smallPatterns3D++;
207                 table3D [ mat[0] ] [ mat[1] ] [ mat[2] ]++;
208             }
209         }
210     }
211 
212     /**
213      * search 4-character patterns
214      * @param word to be analyzed
215      * */
216     public void small4char (String word) {
217         int[] mat = new int[4];
218         if ( word.length() < 3 ) return;
```

```
219         else {
220            for (int i=3,j=2,k=1,l=0; i<word.length(); i++,j++,k++,l++) {
221               char a = word.charAt(i); char b = word.charAt(j);
222               char c = word.charAt(k); char d = word.charAt(l);
223               if (getIndex(a) == -1 || getIndex(b) == -1 ||
224                     getIndex(c) == -1 || getIndex(d) == -1) {
225                  mat[0] = 0; mat[1] = 0; mat[2] = 0; mat[3] = 0;
226                  continue;
227               }
228               mat[0] = getIndex(d); mat[1] = getIndex(c);
229               mat[2] = getIndex(b); mat[3] = getIndex(a);
230               smallPatterns4D++;
231               table4D [ mat[0] ] [ mat[1] ] [ mat[2] ] [ mat[3] ]++;
232            }
233         }
234      }
235
236      /** Print analysis results onscreen */
237      public void printToScreen () {
238         System.out.println("\nCharacters: " + smallPatterns1D);
239         System.out.println("2-char patterns: "+smallPatterns2D);
240         System.out.println("3-char patterns: "+smallPatterns3D);
241         System.out.println("4-char patterns: "+smallPatterns4D+"\n");
242         System.out.println("Analyzing and writing files " +
243            "(this takes a few minutes) ...\n");
244      }
245
246      /**
247       * Organized 1D table in descending order using bubble sort
248       * table1Dorg is different from the table1D, in which the order
249       * remains alphabetical.
250       * @return table with proper indices
251       */
252      public void organizeDescending1D () {
253         int [] chars = new int [smallDim];
254         for (int i=0; i < smallDim; i++) {
255            chars[i] = i;
256            table1Dorg[i] = table1D[i];
257            nTable1Dorg[i] = nTable1D[i];
258         }
259         for (int k=1; k < smallDim; k++) {
260            for (int j=0; j < smallDim-k; j++) {
261               if (table1Dorg[j] < table1Dorg[j+1]) {
262                  int tmp=table1Dorg[j];
263                  table1Dorg[j]=table1Dorg[j+1];
264                  table1Dorg[j+1]=tmp;
265                  int itmp=chars[j];
266                  chars[j]=chars[j+1];
267                  chars[j+1]=itmp;
268                  double ntmp=nTable1Dorg[j];
269                  nTable1Dorg[j]=nTable1Dorg[j+1];
270                  nTable1Dorg[j+1]=ntmp;
271               }
272            }
273         } // set indices accordingly
274         for (int i=0; i < smallDim; i++) chars1Dorg[i] = chars[i];
275      }
```

```
276
277     /** Normalizes smallAnalysis matrices */
278     private void normalize () {
279        normalize1();
280        normalize2();
281        normalize3();
282        normalize4();
283     }
284
285     /** Normalizes 1D character table values */
286     private void normalize1 () {
287        int count = smallPatterns1D;
288        double normFactor = 1 / (double) count;
289        for ( int i = 0; i < smallDim; i++ ) {
290           nTable1D[i] = (double) normFactor * table1D[i];
291        }
292     }
293
294     /** Normalizes 2D character table values */
295     private void normalize2 () {
296        int count = smallPatterns2D;
297        double normFactor = 1 / (double) count;
298        for ( int i = 0; i < smallDim; i++ ) {
299           for ( int j = 0; j < smallDim; j++ ) {
300              nTable2D[i][j] = (double) normFactor * table2D[i][j];
301           }
302        }
303     }
304
305     /** Normalizes 3D character table values */
306     private void normalize3 () {
307        int count = smallPatterns3D;
308        double normFactor = 1 / (double) count;
309        for ( int i = 0; i < smallDim; i++ ) {
310           for ( int j = 0; j < smallDim; j++ ) {
311              for ( int k = 0; k < smallDim; k++ ) {
312                 nTable3D[i][j][k] = (double) normFactor*table3D[i][j][k];
313              }
314           }
315        }
316     }
317
318     /** Normalizes 4D character table values
319      * TODO: filter applied before normalization?
320      * Does this take a long time to compute?
321      * Problem with very small values in 2D, 3D and 4D?
322      */
323     private void normalize4 () {
324        int count = smallPatterns4D;
325        double normFactor = 1 / (double) count;
326        for ( int i = 0; i < smallDim; i++ ) {
327           for ( int j = 0; j < smallDim; j++ ) {
328              for ( int k = 0; k < smallDim; k++ ) {
329                 for ( int l = 0; l < smallDim; l++ ) {
330                    nTable4D[i][j][k][l] =
331                       (double) normFactor*table4D[i][j][k][l];
332                 }
```

```
333             }
334         }
335     }
336 }
337 
338 /**
339  * TODO: normalized distributions?
340  * calculate distributions of patterns frequencies
341  * NOTE: zero values of patterns are filtered since they
342  * contribute to nothing.
343  * NO 1D analysis since there is no point.
344  */
345 private void distributions () {
346     distribution2D();
347     distribution3D();
348     distribution4D();
349 }
350 
351 /**
352  * divides the values of 2D table into classes according to
353  * their frequencies. This method is not visible to user and
354  * was used only temporarily to verify that the distribution
355  * of 2D values is not normal -> cannot apply Gaussian statistics
356  */
357 private void distribution2D () {
358     int largest = findLargest2D();
359     double separator = (double) largest / (double) (smallDim*distConst);
360     // this is ridiculously slow
361     for (int i = 0; i < smallDim; i++) {
362         for (int j = 0; j < smallDim; j++) {
363             for (int k = 0, h=1; k < (smallDim*distConst); k++, h++) {
364                 if ( ( table2D[i][j] > (k*separator) ) &&
365                         ( table2D[i][j] <= (h*separator) ) ) {
366                     distributionTable2D[k]++;
367                     break; // inner loop breaks
368                 }
369             }
370         }
371     }
372 }
373 
374 /** forms a distribution of 3D pattern frequencies. */
375 private void distribution3D () {
376     int largest = findLargest3D();
377     double separator = (double) largest / (double) (smallDim*distConst);
378     // this is even more ridiculously slow
379     for (int i = 0; i < smallDim; i++) {
380         for (int j = 0; j < smallDim; j++) {
381             for (int l = 0; l < smallDim; l++) {
382                 for (int k = 0, h=1; k < (smallDim*distConst); k++, h++) {
383                     if ( ( table3D[i][j][l] > (k*separator) ) &&
384                             ( table3D[i][j][l] <= (h*separator) ) ) {
385                         distributionTable3D[k]++;
386                         break;
387                     }
388                 }
389             }
```

```
390             }
391          }
392       }
393
394       /** forms a distribution of 4D pattern frequencies. */
395       private void distribution4D () {
396          int largest = findLargest4D();
397          double separator = (double) largest / (double) (smallDim*distConst);
398          // this is the most ridiculously slow... do not copy!
399          for (int i=0; i < smallDim; i++) {
400             for (int j=0; j < smallDim; j++) {
401                for (int l=0; l < smallDim; l++) {
402                   for (int m=0; m < smallDim; m++) {
403                      for (int k=0, h=1; k<(smallDim*distConst); k++, h++) {
404                         if ( ( table4D[i][j][l][m] > (k*separator) ) &&
405                            ( table4D[i][j][l][m] <= (h*separator) ) ) {
406                               distributionTable4D[k]++;
407                               break;
408                         }
409                      }
410                   }
411                }
412             }
413          }
414       }
415
416       /** find largest value of a 2D int table */
417       private int findLargest1D () {
418          int largest = 0;
419          for (int i = 0; i < smallDim; i++) {
420             if ( table1D[i] > largest )
421                largest = table1D[i];
422          }
423          return largest ;
424       }
425
426       /** find largest value of a 2D int table */
427       private int findLargest2D () {
428          int largest = 0;
429          for (int i = 0; i < smallDim; i++) {
430             for (int j = 0; j < smallDim; j++) {
431                if ( table2D[i][j] > largest )
432                   largest = table2D[i][j];
433             }
434          }
435          return largest ;
436       }
437
438       /** find largest value of a 3D int table */
439       private int findLargest3D () {
440          int largest = 0;
441          for (int i = 0; i < smallDim; i++) {
442             for (int j = 0; j < smallDim; j++) {
443                for (int k = 0; k < smallDim; k++) {
444                   if ( table3D[i][j][k] > largest )
445                      largest = table3D[i][j][k];
446                }
```

```
447             }
448         }
449         return largest ;
450     }
451 
452     /** find largest value of a 3D int table */
453     private int findLargest4D () {
454         int largest = 0;
455         for (int i = 0; i < smallDim; i++) {
456             for (int j = 0; j < smallDim; j++) {
457                 for (int k = 0; k < smallDim; k++) {
458                     for (int l = 0; l < smallDim; l++) {
459                         if ( table4D[i][j][k][l] > largest )
460                             largest = table4D[i][j][k][l];
461                     }
462                 }
463             }
464         }
465         return largest ;
466     }
467 
468      /**
469       * Writes all files to user-specified directory
470       * Automatic file naming because of the large number of files
471       * TODO: add filter and parse writing constraints
472       * @param pathname pathname where to write
473       */
474     public void write (String pathname)  {
475         try {
476             write1D(pathname);
477             write2D(pathname);
478             write3D(pathname);
479             write4D(pathname);
480             writeAttributes(pathname);
481             writeDistribution(pathname);
482         } catch (Exception e) {
483          System.out.println("File not found (writeSmall)");
484         }
485     }
486 
487     /**
488      * write attributes of SmallMatrices object, such as
489      * #of patterns for each dimension etc.
490      * @param pathname where to write the values
491      */
492     private void writeAttributes (String pathname) {
493         File file = new File (pathname, attributeFile+txt); // in rootdir
494         PrintStream fi = null;
495         try {
496             fi = new PrintStream(new FileOutputStream(file, false));
497             try { // add patterns here as needed
498                 fi.println("# Number of SmallChar patterns for " +
499                    "1D, 2D, 3D and 4D:");
500                 fi.println( smallPatterns1D + "|"+ smallPatterns2D + "|" +
501                     smallPatterns3D + "|" + smallPatterns4D);
502                 fi.println("# largest single values for a pattern were: ");
503                 fi.println( findLargest1D() + "|" + findLargest2D() + "|" +
```

```
504             findLargest3D() + "|" + findLargest4D());
505          } finally {
506             fi.close();
507          }
508       } catch (FileNotFoundException e) {
509          System.out.println("File not found" + e.getMessage());
510       }
511    }
512
513    /**
514     * writes distribution files
515     * values are saved in subdirectory for each dimension with
516     * the corresponding pattern analysis data.
517     * @param pathname where to write the values
518     */
519    private void writeDistribution (String pathname) {
520       writeDistribution2D(pathname);
521       writeDistribution3D(pathname);
522       writeDistribution4D(pathname);
523    }
524
525    /** writes 2D distribution to a dat file in 2D subfolder */
526    private void writeDistribution2D (String pathname) {
527       File file = new File (pathname+AnalyzeMatrices.subDir2D,
528             distributionFile2D+dat);
529       PrintStream fi = null;
530       try {
531          fi = new PrintStream(new FileOutputStream(file, false));
532          try {
533             writeDistributionComments(fi, findLargest2D() );
534             for (int i = 0; i< smallDim*distConst; i++) {
535                fi.println(distributionTable2D[i]);
536             }
537          } finally {
538             fi.close();
539          }
540       } catch (FileNotFoundException e) {
541          System.out.println("File not found" + e.getMessage());
542       }
543    }
544
545    /** writes 3D distribution */
546    private void writeDistribution3D (String pathname) {
547       File file = new File (pathname+AnalyzeMatrices.subDir3D,
548             distributionFile3D+dat);
549       PrintStream fi = null;
550       try {
551          fi = new PrintStream(new FileOutputStream(file, false));
552          try {
553             writeDistributionComments(fi, findLargest3D() );
554             for (int i = 0; i< smallDim*distConst; i++) {
555                fi.println(distributionTable3D[i]);
556             }
557          } finally {
558             fi.close();
559          }
560       } catch (FileNotFoundException e) {
```

```
561             System.out.println("File not found" + e.getMessage());
562         }
563     }
564
565     /** writes 4D distribution */
566     private void writeDistribution4D (String pathname) {
567         File file = new File (pathname+AnalyzeMatrices.subDir4D,
568                 distributionFile4D+dat);
569         PrintStream fi = null;
570         try {
571             fi = new PrintStream(new FileOutputStream(file, false));
572             try {
573                 writeDistributionComments(fi, findLargest4D() );
574                 for (int i = 0; i< smallDim*distConst; i++) {
575                     fi.println(distributionTable4D[i]);
576                 }
577             } finally {
578                 fi.close();
579             }
580         } catch (FileNotFoundException e) {
581             System.out.println("File not found" + e.getMessage());
582         }
583     }
584
585     /** write comments for distribution files
586      *  @param fi PrintStream where to write to
587      */
588     private void writeDistributionComments (PrintStream fi, int j)  {
589         int classSize = j / (smallDim*distConst);
590         fi.println("# Classes are uniform width of approximately "
591         + classSize + "." );
592         fi.println("# There are " + (smallDim*distConst) + " classes.");
593     }
594
595     /**
596      * writes files for 1D in largest frequency order
597      * @param pathname where the files are written
598      * @throws Exception if file not found
599      */
600     private void write1D(String pathname) throws Exception {
601         writeNormalized1D(pathname);
602         writeAbsolute1D(pathname);
603         System.out.println("1% complete");
604     }
605
606     /**
607      * writes "double"-type 1D results, i.e. the normalized files
608      * @param pathname where to write
609      * @param loc locale to replace comma with a dot
610      */
611     private void writeNormalized1D (String pathname) {
612         File file = new File (pathname+AnalyzeMatrices.subDir1D,
613                 norm1D+txt );
614         File zfile = new File (pathname+AnalyzeMatrices.subDir1D,
615                 nzval1D+dat ); // z files with .dat extension
616         File letfile = new File (pathname+AnalyzeMatrices.subDir1D,
617                 nlet1D+txt );
```

```
618         PrintStream fi = null, fe = null, fu = null;
619         try {
620             fi = new PrintStream(new FileOutputStream(file , false));
621             fe = new PrintStream(new FileOutputStream(zfile , false));
622             fu = new PrintStream(new FileOutputStream(letfile , false));
623             writeComments1D(fi ,fe ,fu);
624             normWriteDetails1D(fi ,fe ,fu);
625         } catch (FileNotFoundException e) {
626             System.out.println("File not found" + e.getMessage());
627         }
628     }
629
630     /** normalized files writing in 1D: see writeNormalized1D */
631     private void normWriteDetails1D( PrintStream fi, PrintStream fe,
632           PrintStream fu) {
633        Locale loc = new Locale ("en"); // replace comma with dot
634        try {
635           for (int i=0; i < smallDim; i++) {
636              fi.printf(loc, "%2d" + " " + "%f%n",
637                 chars1Dorg[i], nTable1Dorg[i] );
638              fe.printf(loc, "%f%n", nTable1D[i] );
639              fu.printf(loc, "%2c" + " " + "%f%n",
640                 getChar(chars1Dorg[i]), nTable1Dorg[i] );
641           }
642        } finally {
643           fi.close(); fe.close(); fu.close();
644        }
645     }
646
647     /**
648      * writes "int"-type 1D analysis results, i.e. non-normalized files
649      * @param pathname where to write
650      * @throws Exception if file is not found
651      */
652     private void writeAbsolute1D (String pathname) {
653          File file = new File (pathname+AnalyzeMatrices.subDir1D,
654              basic1D+txt );
655          File zfile = new File (pathname+AnalyzeMatrices.subDir1D,
656              zval1D+dat );
657          File letfile = new File (pathname+AnalyzeMatrices.subDir1D,
658              let1D+txt );
659          PrintStream fi = null, fe = null, fu = null;
660          try {
661              fi = new PrintStream(new FileOutputStream(file , false));
662              fe = new PrintStream(new FileOutputStream(zfile , false));
663              fu = new PrintStream(new FileOutputStream(letfile , false));
664              writeComments1D (fi, fe, fu);
665              absWriteDetails1D (fi ,fe ,fu);
666          } catch (FileNotFoundException e) {
667              System.out.println("File not found" + e.getMessage() );
668          }
669     }
670
671     /** absolute frequency files writing in 1D: see writeAbsolute1D */
672     private void absWriteDetails1D (PrintStream fi, PrintStream fe,
673           PrintStream fu) {
674        Locale loc = new Locale ("en"); // replace comma with dot
```

```
675         try {
676          for (int i=0; i < smallDim; i++) {
677              fi.printf(loc, "%2d" + " " + "%d%n",
678                    chars1Dorg [i], table1Dorg[i] );
679              fe.printf(loc, "%d%n", table1D[i] );
680              fu.printf(loc, "%2c" + " " + "%d%n",
681                 getChar(chars1Dorg[i]), table1Dorg[i] );
682          }
683         } finally {
684          fi.close(); fe.close(); fu.close();
685         }
686     }
687
688     /** Writes comments for 1D: see writeabsolute1D(); */
689     private void writeComments1D (PrintStream fi, PrintStream fe, PrintStream fu){
690         fi.println("# These are in \"highest first\" order");
691         fe.println("# These are in alphabetical order");
692         fu.println("# These are in \"highest first\" order");
693     }
694
695     /**
696      * writes files for 2D
697      * @param pathname where the files are written
698      * @throws Exception of file not found
699      */
700     private void write2D(String pathname) throws Exception {
701        writeNormalized2D(pathname);
702        System.out.println("5% complete");
703        writeAbsolute2D(pathname);
704        System.out.println("10% complete");
705     }
706
707     /**
708      * writes "double"-type 2D results, i.e. the normalized files
709      * @param pathname where to write
710      * @param loc locale use english to replace comma with a dot
711      */
712     private void writeNormalized2D (String pathname) {
713          File file = new File (pathname+AnalyzeMatrices.subDir2D,
714              norm2D+txt );
715          File zfile = new File (pathname+AnalyzeMatrices.subDir2D,
716              nzval2D+dat ); // z files with .dat extension
717          File letfile = new File (pathname+AnalyzeMatrices.subDir2D,
718              nlet2D+txt );
719          PrintStream fi = null, fe = null, fu = null;
720          try {
721              fi = new PrintStream(new FileOutputStream(file, false));
722              fe = new PrintStream(new FileOutputStream(zfile, false));
723              fu = new PrintStream(new FileOutputStream(letfile, false));
724           writeMultidimComments(fi,fe,fu);
725              normWriteDetails2D (fi,fe,fu);
726          } catch (FileNotFoundException e) {
727              System.out.println("File not found" + e.getMessage());
728          }
729     }
730
731     /** normalized files writing in 2D: see writeNormalized2D  */
```

```
732     private void normWriteDetails2D( PrintStream fi, PrintStream fe,
733           PrintStream fu) {
734        Locale loc = new Locale ("en"); // replace comma
735        try {
736           for (int i=0; i < smallDim; i++) {
737              for (int j=0; j < smallDim; j++) {
738                 fi.printf(loc, "%2d" + " " + "%2d" + " " + "%f%n",
739                       i, j, nTable2D[i][j] ); //indexvalues
740                 fe.printf(loc, "%f%n", nTable2D[i][j] ); // zvalues
741                 fu.printf(loc, "%2c" + " " + "%2c" + " " + "%f%n", //letter
742                       getChar(i), getChar(j), nTable2D[i][j] );
743              }
744           }
745        } finally {
746           fi.close(); fe.close(); fu.close();
747        }
748     }
749
750     /**
751      * writes "int"-type 2D analysis results, i.e. non-normalized files
752      * @param pathname where to write
753      * @throws Exception if file is not found
754      */
755     private void writeAbsolute2D (String pathname) {
756           File file = new File (pathname+AnalyzeMatrices.subDir2D,
757               basic2D+txt );
758           File zfile = new File (pathname+AnalyzeMatrices.subDir2D,
759               zval2D+dat );
760           File letfile = new File (pathname+AnalyzeMatrices.subDir2D,
761               let2D+txt );
762           PrintStream fi = null, fe = null, fu = null;
763           try {
764               fi = new PrintStream(new FileOutputStream(file, false));
765               fe = new PrintStream(new FileOutputStream(zfile, false));
766               fu = new PrintStream(new FileOutputStream(letfile, false));
767            writeMultidimComments(fi,fe,fu);
768               absWriteDetails2D(fi,fe,fu);
769           } catch (FileNotFoundException e) {
770               System.out.println("File not found" + e.getMessage() );
771           }
772     }
773
774     /** absolute frequency files writing in 2D: see writeAbsolute2D */
775     private void absWriteDetails2D (PrintStream fi, PrintStream fe,
776           PrintStream fu) {
777        Locale loc = new Locale ("en"); // replace comma
778          try {
779            for (int i=0; i < smallDim; i++) {
780              for (int j=0; j < smallDim; j++) {
781                 fi.printf(loc, "%2d" + " " + "%2d" + " " + "%d%n",
782                       i, j, table2D[i][j] );
783                 fe.printf(loc, "%d%n", table2D[i][j] );
784                 fu.printf(loc, "%2c" + " " + "%2c" + " " + "%d%n",
785                       getChar(i), getChar(j), table2D[i][j] );
786              }
787            }
788          } finally {
```

```
789             fi.close(); fe.close(); fu.close();
790           }
791     }
792
793     /** Write standard comments to multidimension (2D,3D and 4D) files */
794     private void writeMultidimComments(PrintStream fi, PrintStream fe,
795           PrintStream fu) {
796         fi.println("# These are in alphabetical order");
797         fe.println("# These are in alphabetical order");
798         fu.println("# These are in alphabetical order");
799     }
800
801     /**
802      * writes files for 3D
803      * @param pathname where the files are written
804      * @throws Exception if file not found
805      */
806     private void write3D(String pathname) throws Exception {
807        writeNormalized3D(pathname);
808        System.out.println("15% complete");
809        writeAbsolute3D(pathname);
810        System.out.println("20% complete");
811     }
812
813     /**
814      * writes "double"-type 3D results, i.e. the normalized files
815      * @param pathname where to write
816      * @param loc locale use english to replace comma with a dot
817      */
818     private void writeNormalized3D (String pathname) {
819           File file = new File (pathname+AnalyzeMatrices.subDir3D,
820               norm3D+txt );
821           File zfile = new File (pathname+AnalyzeMatrices.subDir3D,
822               nzval3D+dat ); // z files with .dat extension
823           File letfile = new File (pathname+AnalyzeMatrices.subDir3D,
824               nlet3D+txt );
825           PrintStream fi = null, fe = null, fu = null;
826           try {
827               fi = new PrintStream(new FileOutputStream(file, false));
828               fe = new PrintStream(new FileOutputStream(zfile, false));
829               fu = new PrintStream(new FileOutputStream(letfile, false));
830            writeMultidimComments(fi,fe,fu);
831               normWriteDetails3D (fi,fe,fu);
832           } catch (FileNotFoundException e) {
833               System.out.println("File not found" + e.getMessage());
834           }
835     }
836
837     /** normalized files writing in 3D: see writeNormalized3D */
838     private void normWriteDetails3D( PrintStream fi, PrintStream fe,
839           PrintStream fu) {
840        Locale loc = new Locale ("en"); // replace comma with dot
841        try {
842           for (int i=0; i < smallDim; i++) {
843              for (int j=0; j < smallDim; j++) {
844                 for (int k=0; k < smallDim; k++) {
845                    fi.printf(loc, "%2d" + " " + "%2d" + " " + "%2d" +
```

```
846                     " " + "%f%n",
847                     i, j, k, nTable3D[i][j][k] ); //indexvalues
848                 fe.printf(loc, "%f%n", nTable3D[i][j][k] ); // zvalues
849                 fu.printf(loc, "%2c" + " " + "%2c" + " " + "%2c" +
850                      " " + "%f%n", //letter
851                     getChar(i), getChar(j),
852                     getChar(k), nTable3D[i][j][k] );
853             }
854           }
855         }
856     } finally {
857         fi.close(); fe.close(); fu.close();
858     }
859   }
860
861   /**
862    * writes "int"-type 3D analysis results, i.e. non-normalized files
863    * @param pathname where to write
864    * @throws Exception if file is not found
865    */
866   private void writeAbsolute3D (String pathname) {
867       File file = new File (pathname+AnalyzeMatrices.subDir3D,
868           basic3D+txt );
869       File zfile = new File (pathname+AnalyzeMatrices.subDir3D,
870           zval3D+dat );
871       File letfile = new File (pathname+AnalyzeMatrices.subDir3D,
872           let3D+txt );
873       PrintStream fi = null, fe = null, fu = null;
874       try {
875           fi = new PrintStream(new FileOutputStream(file, false));
876           fe = new PrintStream(new FileOutputStream(zfile, false));
877           fu = new PrintStream(new FileOutputStream(letfile, false));
878        writeMultidimComments(fi,fe,fu);
879           absWriteDetails3D(fi,fe,fu);
880       } catch (FileNotFoundException e) {
881           System.out.println("File not found" + e.getMessage() );
882       }
883   }
884
885   /** absolute frequency files writing in 3D: see writeAbsolute2D */
886   private void absWriteDetails3D (PrintStream fi, PrintStream fe,
887         PrintStream fu) {
888     Locale loc = new Locale ("en"); // replace comma with dot
889     try {
890         for (int i=0; i < smallDim; i++) {
891           for (int j=0; j < smallDim; j++) {
892             for (int k=0; k < smallDim; k++) {
893               fi.printf(loc, "%2d" + " " + "%2d" + " " + "%2d" +
894                    " " + "%d%n",
895                    i, j, k, table3D[i][j][k] );
896               fe.printf(loc, "%d%n", table3D[i][j][k] );
897               fu.printf(loc, "%2c" + " " + "%2c" + " " + "%2c" +
898                    " " + "%d%n",
899                    getChar(i), getChar(j),
900                    getChar(k), table3D[i][j][k] );
901             }
902           }
```

```
903             }
904         } finally {
905          fi.close(); fe.close(); fu.close();
906         }
907     }
908 
909     /**
910      * writes files for 4D
911      * TODO: organize writing better, merge with other dimensions
912      * @param pathname where the files are written
913      * @throws Exception of file not found
914      */
915     public void write4D(String pathname) throws Exception {
916        writeNormalized4D(pathname);
917        System.out.println("55% complete");
918        writeAbsolute4D(pathname);
919     }
920 
921     /**
922      * writes "double"-type 4D results, i.e. the normalized files
923      * @param pathname where to write
924      * @param loc locale use english to replace comma with a dot
925      */
926     private void writeNormalized4D (String pathname) {
927          File file = new File (pathname+AnalyzeMatrices.subDir4D,
928              norm4D+txt );
929          File zfile = new File (pathname+AnalyzeMatrices.subDir4D,
930              nzval4D+dat ); // z files with .dat extension
931          File letfile = new File (pathname+AnalyzeMatrices.subDir4D,
932              nlet4D+txt );
933          PrintStream fi = null, fe = null, fu = null;
934          try {
935              fi = new PrintStream(new FileOutputStream(file, false));
936              fe = new PrintStream(new FileOutputStream(zfile, false));
937              fu = new PrintStream(new FileOutputStream(letfile, false));
938           writeMultidimComments(fi,fe,fu);
939              normWriteDetails4D (fi,fe,fu);
940          } catch (FileNotFoundException e) {
941              System.out.println("File not found" + e.getMessage());
942          }
943     }
944 
945     /** normalized files writing in 4D: see writeNormalized4D */
946     private void normWriteDetails4D( PrintStream fi, PrintStream fe,
947           PrintStream fu) {
948        Locale loc = new Locale ("en"); // replace comma
949        try {
950           for (int i=0; i < smallDim; i++) {
951              for (int j=0; j < smallDim; j++) {
952                 for (int k=0; k < smallDim; k++) {
953                    for (int l=0; l < smallDim; l++) {
954                       fi.printf(loc, "%2d" + " " + "%2d" + " " +
955                             "%2d" + " " + "%2d" + " " + "%f%n",
956                              i,j,k,l, nTable4D[i][j][k][l] );
957                       fe.printf(loc, "%f%n", nTable4D[i][j][k][l] );
958                       fu.printf(loc, "%2c" + " " + "%2c" + " " +
959                           "%2c" + " " + "%2c" + " " + "%f%n",
```

```
960                             getChar(i), getChar(j),
961                             getChar(k), getChar(l),
962                             nTable4D[i][j][k][l] );
963                     }
964                 }
965             }
966             if (i == 4) System.out.println("25% complete");
967             if (i == 8) System.out.println("30% complete");
968             if (i == 12) System.out.println("35% complete");
969             if (i == 16) System.out.println("40% complete");
970             if (i == 20) System.out.println("45% complete");
971             if (i == 24) System.out.println("50% complete");
972         }
973     } finally {
974         fi.close(); fe.close(); fu.close();
975     }
976 }
977
978 /**
979  * writes "int"-type 4D analysis results, i.e. non-normalized files
980  * @param pathname where to write
981  * @throws Exception if file is not found
982  * TODO: kuormita metodin alkuosa ja tee abswritesta & normwritesta
983  * omat erikoismetodinsa sekä erillinen chooseWrite-metodi, anna
984  * parametrina tiedostonimet (jaa ne osiin
985  * alkumäärittelyissä)
986  */
987 private void writeAbsolute4D (String pathname) {
988       File file = new File (pathname+AnalyzeMatrices.subDir4D,
989           basic4D+txt );
990       File zfile = new File (pathname+AnalyzeMatrices.subDir4D,
991           zval4D+dat );
992       File letfile = new File (pathname+AnalyzeMatrices.subDir4D,
993           let4D+txt );
994       PrintStream fi = null, fe = null, fu = null;
995       try {
996           fi = new PrintStream(new FileOutputStream(file, false));
997           fe = new PrintStream(new FileOutputStream(zfile, false));
998           fu = new PrintStream(new FileOutputStream(letfile, false));
999        writeMultidimComments(fi,fe,fu);
1000          absWriteDetails4D(fi,fe,fu);
1001      } catch (FileNotFoundException e) {
1002          System.out.println("File not found" + e.getMessage() );
1003      }
1004  }
1005
1006  /** absolute frequency files writing in 4D: see writeAbsolute4D */
1007  private void absWriteDetails4D (PrintStream fi, PrintStream fe,
1008        PrintStream fu) {
1009     Locale loc = new Locale ("en"); // replace comma
1010       try {
1011         for (int i=0; i < smallDim; i++) {
1012           for (int j=0; j < smallDim; j++) {
1013             for (int k=0; k < smallDim; k++) {
1014               for (int l=0; l < smallDim; l++) {
1015                 fi.printf(loc, "%2d" + " " + "%2d" + " " +
1016                     "%2d" + " " + "%2d" + " " + "%d%n",
```

```
1017                        i,j,k,l, table4D[i][j][k][l] );
1018                    fe.printf(loc, "%d%n", table4D[i][j][k][l] );
1019                    fu.printf(loc, "%2c" + " " + "%2c" + " " +
1020                        "%2c" + " " + "%2c" + " " + "%d%n",
1021                          getChar(i), getChar(j),
1022                          getChar(k), getChar(l),
1023                          table4D[i][j][k][l] );
1024                }
1025             }
1026          }
1027          if (i == 4) System.out.println("60% complete");
1028          if (i == 8) System.out.println("65% complete");
1029          if (i == 12) System.out.println("70% complete");
1030          if (i == 16) System.out.println("75% complete");
1031          if (i == 20) System.out.println("80% complete");
1032          if (i == 24) System.out.println("85% complete");
1033          if (i == 26) System.out.println("90% complete\n");
1034        }
1035      } finally {
1036        fi.close(); fe.close(); fu.close();
1037      }
1038  }
1039
1040  /**
1041   * Only indices for small letters. See getAllIndex() in
1042   * FullMatrices Class for capital letters and special characters.
1043   * Indices 0-28 correspond to letters a to ö and empty space
1044   *
1045   * @param a character for which the index will be determined
1046   * @return i the index of the ith field (corresponding to the letter)
1047   * </pre>
1048   */
1049  public static int getIndex(char a){
1050     switch (a){
1051     case 'a':
1052        return 0;
1053     case 'b':
1054        return 1;
1055     case 'c':
1056        return 2;
1057     case 'd':
1058        return 3;
1059     case 'e':
1060        return 4;
1061     case 'f':
1062        return 5;
1063     case 'g':
1064        return 6;
1065     case 'h':
1066        return 7;
1067     case 'i':
1068        return 8;
1069     case 'j':
1070        return 9;
1071     case 'k':
1072        return 10;
1073     case 'l':
```

```
1074            return 11;
1075        case 'm':
1076            return 12;
1077        case 'n':
1078            return 13;
1079        case 'o':
1080            return 14;
1081        case 'p':
1082            return 15;
1083        case 'q':
1084            return 16;
1085        case 'r':
1086            return 17;
1087        case 's':
1088            return 18;
1089        case 't':
1090            return 19;
1091        case 'u':
1092            return 20;
1093        case 'v':
1094            return 21;
1095        case 'w':
1096            return 22;
1097        case 'x':
1098            return 23;
1099        case 'y':
1100            return 24;
1101        case 'z':
1102            return 25;
1103        case 'å':
1104            return 26;
1105        case 'ä':
1106            return 27;
1107        case 'ö':
1108            return 28;
1109        default:
1110            return -1; // if not found return negative value
1111        }
1112    }
1113
1114
1115    /**
1116     * Only for small letters. Returns character corresponding to
1117     * index as defined in getIndex(). For all chars, see
1118     * getAllChars() in FullMatrices Class
1119     *
1120     * Fields and indices in order :
1121     * 0-28    a to ö
1122     *
1123     * @param index for which the character will be determined
1124     * @return char of ith index corresponding to the index i
1125     * </pre>
1126     */
1127    public static char getChar(int index){
1128        switch (index){
1129        case 0:
1130            return 'a';
```

```
1131        case 1:
1132            return 'b';
1133        case 2:
1134            return 'c';
1135        case 3:
1136            return 'd';
1137        case 4:
1138            return 'e';
1139        case 5:
1140            return 'f';
1141        case 6:
1142            return 'g';
1143        case 7:
1144            return 'h';
1145        case 8:
1146            return 'i';
1147        case 9:
1148            return 'j';
1149        case 10:
1150            return 'k';
1151        case 11:
1152            return 'l';
1153        case 12:
1154            return 'm';
1155        case 13:
1156            return 'n';
1157        case 14:
1158            return 'o';
1159        case 15:
1160            return 'p';
1161        case 16:
1162            return 'q';
1163        case 17:
1164            return 'r';
1165        case 18:
1166            return 's';
1167        case 19:
1168            return 't';
1169        case 20:
1170            return 'u';
1171        case 21:
1172            return 'v';
1173        case 22:
1174            return 'w';
1175        case 23:
1176            return 'x';
1177        case 24:
1178            return 'y';
1179        case 25:
1180            return 'z';
1181        case 26:
1182            return 'å';
1183        case 27:
1184            return 'ä';
1185        case 28:
1186            return 'ö';
1187        default:
```

```
1188            return '§'; // if not found
1189        }
1190    }
1191
1192
1193    /**
1194     * Tests for getChar() and getIndex()
1195     * Prints all chars and their corresponding indices
1196     * Also prints indices for chars not analyzed
1197     */
1198     public void testChars () {
1199        System.out.println();
1200        System.out.println("RECOGNIZED CHARACTERS AND INDICES");
1201        System.out.println();
1202        for( int i = 0; i < this.smallDim; i++ ) {
1203            // print all chars
1204            System.out.println(i+" "+getChar(i)+" "+getIndex(getChar(i)) );
1205        }
1206        System.out.println();
1207        System.out.println("NOT RECOGNIZED INDICES");
1208        System.out.println("Special characters");
1209
1210        System.out.println("@, " + getIndex('@'));
1211        System.out.println("§, " + getIndex('§'));
1212        System.out.println("½, " + getIndex('½'));
1213        System.out.println("\", " + getIndex('\"'));
1214        System.out.println("#, " + getIndex('#'));
1215        System.out.println("£, " + getIndex('£'));
1216        System.out.println("¤, " + getIndex('¤'));
1217        System.out.println("$, " + getIndex('$'));
1218        System.out.println("/, " + getIndex('/'));
1219        System.out.println("{, " + getIndex('{'));
1220        System.out.println("[, " + getIndex('['));
1221        System.out.println("], " + getIndex(']'));
1222        System.out.println("=, " + getIndex('='));
1223        System.out.println("}, " + getIndex('}'));
1224        System.out.println("+, " + getIndex('+'));
1225        System.out.println("\\, " + getIndex('\\'));
1226        System.out.println("´, " + getIndex('´'));
1227        System.out.println("‘, " + getIndex('‘'));
1228        System.out.println("^, " + getIndex('^'));
1229        System.out.println("~, " + getIndex('~'));
1230        System.out.println("¨, " + getIndex('¨'));
1231        System.out.println();
1232
1233        System.out.println("Numbers");
1234        System.out.println("1, " + getIndex('1'));
1235        System.out.println("2, " + getIndex('2'));
1236        System.out.println("3, " + getIndex('3'));
1237        System.out.println("4, " + getIndex('4'));
1238        System.out.println("5, " + getIndex('5'));
1239        System.out.println("6, " + getIndex('6'));
1240        System.out.println("7, " + getIndex('7'));
1241        System.out.println("8, " + getIndex('8'));
1242        System.out.println("9, " + getIndex('9'));
1243        System.out.println("0, " + getIndex('0'));
1244    }
```

```
1245 }
```