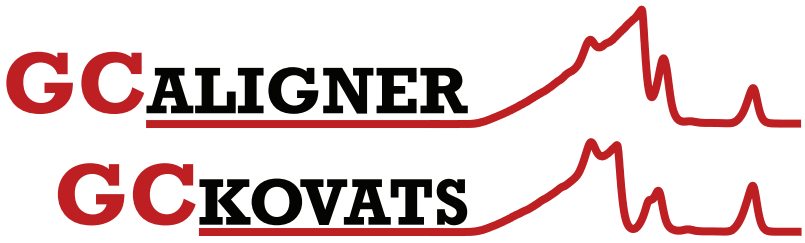
GCALIGNER 1.0 and GCKOVATS 1.0

-

Manual of a software suite to compute a multiple sample comparison data matrix from eco-chemical datasets obtained by gas chromatography



Simon Dellicour

Thomas Lecocq

UMONS

GCALIGNER 1.0 and GCKOVATS 1.0
-
Manual of a software suite to compute a multiple sample comparison data matrix from eco-chemical datasets obtained by gas chromatography

Simon Dellicour
Evolutionary Biology and Ecology, Free University of Brussels, av. FD Roosevelt 50, 1050 Brussels, Belgium.

Thomas Lecocq
Laboratoire de Zoologie, University of Mons, Place du Parc 20, 7000 Mons, Belgium.

2013

Publisher:
Université de Mons
Faculté des Sciences
Place du Parc, 23
7000 Mons
Belgium

ISBN : 978-2-87325-079-9
***Dépôt légal* : D/2013/970/5**

Contents

A disk of GCALIGNER 1.0 and GCKOVATS 1.0 is joined to this manual.

Foreword

Chemical ecology is the discipline that deals with molecularly mediated biological interactions (Meinwald & Eisner 2008). It is a multifaceted discipline, intent on deciphering both the chemical structure and the information content of the mediating molecules. It covers a broad range of chemical interactions like chemical communication or chemical defences of organisms (Hartmann 2008) and also includes synthesis, emission, transmission and detection of chemicals. This is also a discipline widely applied in ecology (Medeiros & Simoneit 2007; Martin *et al.* 2008; Youngsteadt *et al.* 2008; Lecocq *et al.* 2013), ethology (Vereecken *et al.* 2007; Lhomme *et al.* 2012), and taxonomy (Bertsch *et al.* 2005; Lecocq *et al.* 2011) and in which discovery is still very much in order, for the interactions themselves remain in large measure to be uncovered.

The characterization of chemicals is the first step of any modern chemo-ecological study that requires furthermore the comparison of many samples by chemical analyses. Chemical ecology has made major progress in recent decades mainly thanks to advances in analytical chemistry (Meinwald & Eisner 2008). Current analytic instruments like gas chromatography allow a fast and easy generation of large high-dimensional datasets. Within the framework of biology, and particularly in chemo-ecology, rapid comparison of many samples is needed in order to support hypothesis testing (Meinwald & Eisner 2008). However, the huge amount of samples makes the computing of a sample comparison data matrix unworkable without automated electronic methods (Bloemberg *et al.* 2013).

GCALIGNER 1.0 and GCKOVATS 1.0 a software suite allowing the alignment of samples analyzed by gas chromatography based on retention time or Kovats retention indices but without mass spectrometry information. This software suite is specifically designed to align chemical compounds detected by gas chromatography in different samples, thereby allowing the comparison of the composition of a high number of samples.

Acknowledgements

We thank our collaborators and co-workers for their help in developing and testing the software suite.

We especially are grateful to Brigitte Frérot, Irena Valterová, Pierre Rasmont, Nora Elvinger, Loïc Dohet, Laurent Grumiau, Louis Hautier, Patrick Lhomme, Nicolas Meurisse, Nicolas Brasero, Thomas Vantorre, Dimitri Evrard, Sarah Vray, and Sophie Lambert.

SD and TL acknowledges the financial support from the Belgian Fonds National pour la Recherche Scientifique (FNRS) and the Fonds pour la Recherche dans l'Industrie et l'Agriculture (FRIA).

1. GCALIGNER method

GCALIGNER aligns data files from several independent samples (e.g. different specimens or different injections) analyzed by gas chromatography (GC) on the basis of the retention times or the Kovats retention indices of each chemical compound detected in each independent sample (Dellicour & Lecocq 2013). By "align data files", we mean find for each compound of a sample the homolog compounds in other samples. GCALIGNER is designed to allow the comparison of the chemical composition of multiple samples. The alignment algorithm is based on the comparison between each retention time (RT) or Kovats retention indices (KRI), the following RT/KRI in the same sample and its closest RT/KRI in other samples. The alignment algorithm is fully published in Dellicour & Lecocq (2013).

2. Software availability

The software suite GCALIGNER & GCKOVATS is a free and open-source software available from the website ebe.ulb.ac.be/ebe/Software.html. The GCALIGNER and GCKOVATS are written in Java programming language (see GCALIGNER source code and GCKOVATS source code). Both programs have a graphical user interface (GUI) and runs on any operating system (Fig. 1).

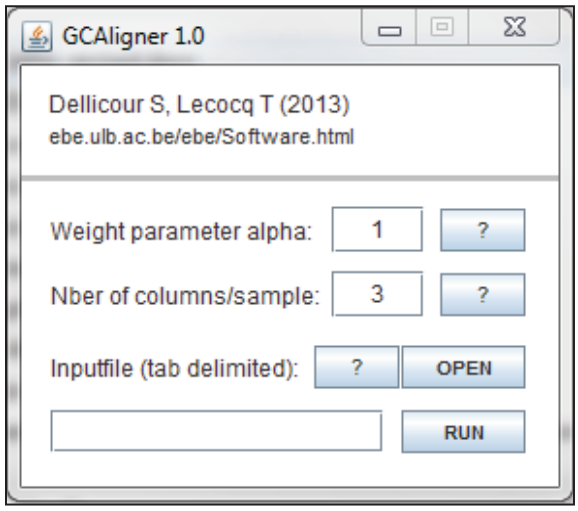
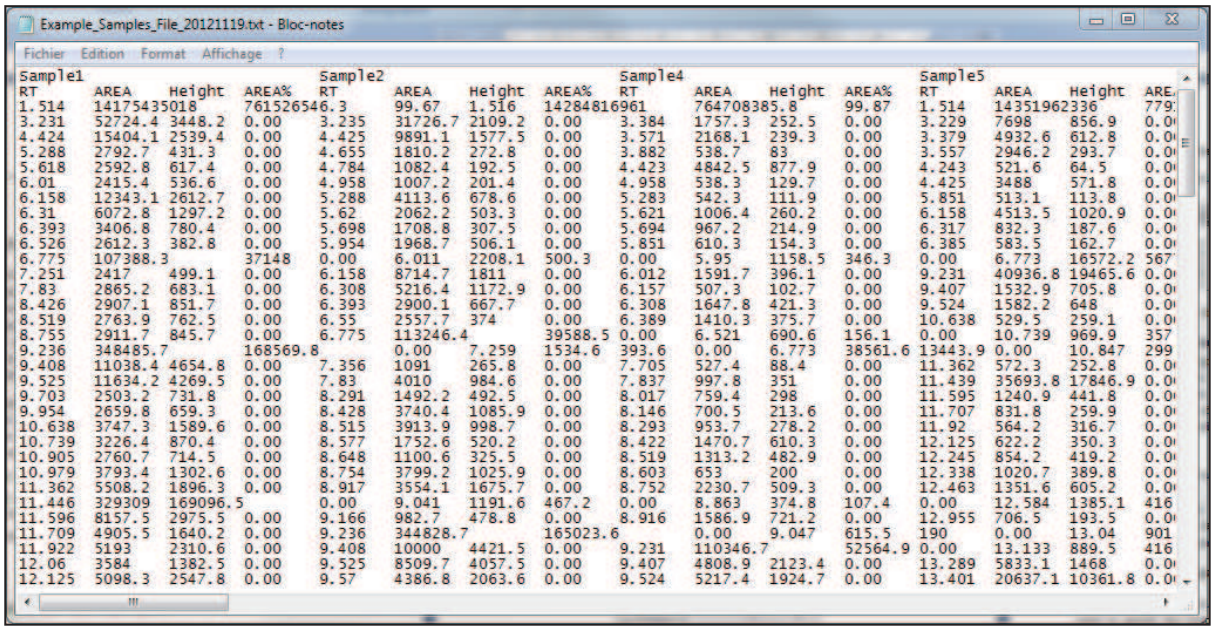


Figure 1: GCALIGNER GUI

3. GCALIGNER input file

GCALIGNER needs an input file that gathers all data files to align. For each data file, GCALIGNER requires the retention time or the Kovats retention indices of each chemical compound to align. The input file must be a tab delimited text file (Fig. 2).



Sample1				Sample2				Sample4				Sample5			
RT	AREA	Height	AREA%	RT	AREA	Height	AREA%	RT	AREA	Height	AREA%	RT	AREA	Height	ARE
1.514	14175435018		761526546.3		99.67	1.516	14284816961		764708385.8		99.87	1.514	14351962336		779
3.231	52724.4	3448.2	0.00	3.235	31726.7	2109.2	0.00	3.384	1757.3	252.5	0.00	3.229	7698	856.9	0.0
4.424	15404.1	2539.4	0.00	4.425	9891.1	1577.5	0.00	3.571	2168.1	239.3	0.00	3.379	4932.6	612.8	0.0
5.288	2792.7	431.3	0.00	4.655	1810.2	272.8	0.00	3.882	538.7	83	0.00	3.557	2946.2	293.7	0.0
5.618	2592.8	617.4	0.00	4.784	1082.4	192.5	0.00	4.423	4842.5	877.9	0.00	4.243	521.6	64.5	0.0
6.01	2415.4	536.6	0.00	4.958	1007.2	201.4	0.00	4.958	538.3	129.7	0.00	4.425	3488	571.8	0.0
6.158	12343.1	2612.7	0.00	5.288	4113.6	678.6	0.00	5.283	542.3	111.9	0.00	5.851	513.1	113.8	0.0
6.31	6072.8	1297.2	0.00	5.62	2062.2	503.3	0.00	5.621	1006.4	260.2	0.00	6.158	4513.5	1020.9	0.0
6.393	3406.8	780.4	0.00	5.698	1708.8	307.5	0.00	5.694	967.2	214.9	0.00	6.317	832.3	187.6	0.0
6.526	2612.3	382.8	0.00	5.954	1968.7	506.1	0.00	5.851	610.3	154.3	0.00	6.385	583.5	162.7	0.0
6.775	107388.3		37148	0.00	6.011	2208.1	500.3	0.00	5.95	1158.5	346.3	0.00	6.773	16572.2	567
7.251	2417	499.1	0.00	6.158	8714.7	1811	0.00	6.012	1591.7	396.1	0.00	9.231	40936.8	19465.6	0.0
7.83	2865.2	683.1	0.00	6.308	5216.4	1172.9	0.00	6.157	507.3	102.7	0.00	9.407	1532.9	705.8	0.0
8.426	2907.1	851.7	0.00	6.393	2900.1	667.7	0.00	6.308	1647.8	421.3	0.00	9.524	1582.2	648	0.0
8.519	2763.9	762.5	0.00	6.55	2557.7	374	0.00	6.389	1410.3	375.7	0.00	10.638	529.5	259.1	0.0
8.755	2911.7	845.7	0.00	6.775	113246.4		39588.5	0.00	6.521	690.6	156.1	0.00	10.739	969.9	357
9.236	348485.7		168569.8		0.00	7.259	1534.6	393.6	0.00	6.773	38561.6	13443.9	0.00	10.847	299
9.408	11038.4	4654.8	0.00	7.356	1091	265.8	0.00	7.705	527.4	88.4	0.00	11.362	572.3	252.8	0.0
9.525	11634.2	4269.5	0.00	7.83	4010	984.6	0.00	7.837	997.8	351	0.00	11.439	35693.8	17846.9	0.0
9.703	2503.2	731.8	0.00	8.291	1492.2	492.5	0.00	8.017	759.4	298	0.00	11.595	1240.9	441.8	0.0
9.954	2659.8	659.3	0.00	8.428	3740.4	1085.9	0.00	8.146	700.5	213.6	0.00	11.707	831.8	259.9	0.0
10.638	3747.3	1589.6	0.00	8.515	3913.9	998.7	0.00	8.293	953.7	278.2	0.00	11.92	564.2	316.7	0.0
10.739	3226.4	870.4	0.00	8.577	1752.6	520.2	0.00	8.422	1470.7	610.3	0.00	12.125	622.2	350.3	0.0
10.905	2760.7	714.5	0.00	8.648	1100.6	325.5	0.00	8.519	1313.2	482.9	0.00	12.245	854.2	419.2	0.0
10.979	3793.4	1302.6	0.00	8.754	3799.2	1025.9	0.00	8.603	653	200	0.00	12.338	1020.7	389.8	0.0
11.362	5508.2	1896.3	0.00	8.917	3554.1	1675.7	0.00	8.752	2230.7	509.3	0.00	12.463	1351.6	605.2	0.0
11.446	329309	169096.5		0.00	9.041	1191.6	467.2	0.00	8.863	374.8	107.4	0.00	12.584	1385.1	416
11.596	8157.5	2975.5	0.00	9.166	982.7	478.8	0.00	8.916	1586.9	721.2	0.00	12.955	706.5	193.5	0.0
11.709	4905.5	1640.2	0.00	9.236	344828.7		165023.6		0.00	9.047	615.5	190	0.00	13.04	901
11.922	5193	2310.6	0.00	9.408	10000	4421.5	0.00	9.231	110346.7		52564.9	0.00	13.133	889.5	416
12.06	3584	1382.5	0.00	9.525	8509.7	4057.5	0.00	9.407	4808.9	2123.4	0.00	13.289	5833.1	1468	0.0
12.125	5098.3	2547.8	0.00	9.57	4386.8	2063.6	0.00	9.524	5217.4	1924.7	0.00	13.401	20637.1	10361.8	0.0

Figure 2: example of an input file.

- The **first column of each sample/injection must always be the RT** column. The program does not read the other columns, which will only be copied in the output file (i.e. the three others columns in Figure 2). The number of additional columns to the RT column is not limited but has to be specified before running the program (see Running GCALIGNER). However the number of columns must be the same for all samples.

- **Empty data are not allowed**.

- The two first lines are not read. The users can fill the two first lines as they please (e.g. the title of each column of the input data matrix). GCALIGNER **needs that the input file presents these two lines** before the different columns of each sample.

4. Running GCALIGNER

A double click on the executable file (".jar") starts GCALIGNER. The users then must specify the input file, the number of columns of each sample, and the weight parameter alpha (Fig. 1). A help popup windows is available for each GCALIGNER field (? buttons).

- The OPEN button of the **Inputfile** field allows the location of the input file.

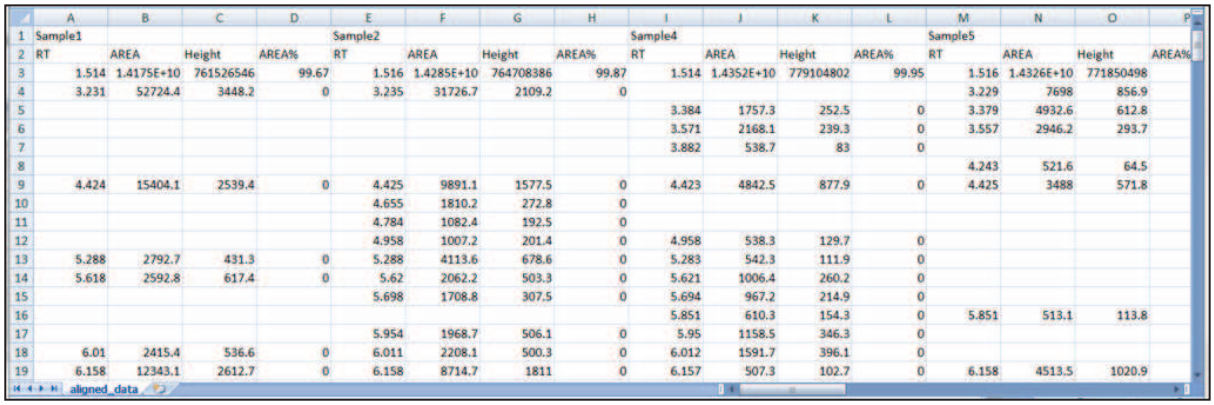
- The **Nber of columns/sample** field defines the number of columns of each sample (including the compulsory first columns with the retention times; see 3 GCALIGNER input file).

- The **weight parameter alpha** field defines the weight parameter alpha. The weight parameter alpha (α) allows to increase or decrease the acceptance rate of a RT "equivalence" (corresponding to the same compound signal) to another RT (for more details see Dellicour & Lecocq 2013). Users are invited to begin with the default value of $\alpha = 1$ and then to decrease (i.e. $\alpha = 0.5, 0.25, 0.125$) or increase (i.e. $\alpha = 2, 3, 4$) this parameter in order to check if it can give better alignment results on their own initial dataset. The choice of α depends on the average difference between compounds' retention times.

When all fields are filled, the user must click on the **RUN** button.

5. GCALIGNER output file

The output file is in an Excel format (".xls "; Fig. 3). **The last line of each injection is not aligned** because the algorithm is based on the comparison between each RT and the following RT (see software limitations in Dellicour & Lecocq 2013)**.** These last lines are simply added at the end of the output file.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Sample1				Sample2				Sample4				Sample5			
2	RT	AREA	Height	AREA%	RT	AREA	Height	AREA%	RT	AREA	Height	AREA%	RT	AREA	Height	AREA%
3	1.514	1.4175E+10	761526546	99.67	1.516	1.4285E+10	764708386	99.87	1.514	1.4352E+10	779104802	99.95	1.516	1.4326E+10	771850498	
4	3.231	52724.4	3448.2	0	3.235	31726.7	2109.2	0					3.229	7698	856.9	
5									3.384	1757.3	252.5	0	3.379	4932.6	612.8	
6									3.571	2168.1	239.3	0	3.557	2946.2	293.7	
7									3.882	538.7	83	0				
8													4.243	521.6	64.5	
9	4.424	15404.1	2539.4	0	4.425	9891.1	1577.5	0	4.423	4842.5	877.9	0	4.425	3488	571.8	
10					4.655	1810.2	272.8	0								
11					4.784	1082.4	192.5	0								
12					4.958	1007.2	201.4	0	4.958	538.3	129.7	0				
13	5.288	2792.7	431.3	0	5.288	4113.6	678.6	0	5.283	542.3	111.9	0				
14	5.618	2592.8	617.4	0	5.62	2062.2	503.3	0	5.621	1006.4	260.2	0				
15					5.698	1708.8	307.5	0	5.694	967.2	214.9	0				
16									5.851	610.3	154.3	0	5.851	513.1	113.8	
17					5.954	1968.7	506.1	0	5.95	1158.5	346.3	0				
18	6.01	2415.4	536.6	0	6.011	2208.1	500.3	0	6.012	1591.7	396.1	0				
19	6.158	12343.1	2612.7	0	6.158	8714.7	1811	0	6.157	507.3	102.7	0	6.158	4513.5	1020.9	

aligned_data

Figure 3: example of an output file.

6. GCALIGNER limitations and authors recommendations

Alignment based on RT clearly shows limits in cases of large gaps between two samples corresponding RT (for more details see Dellicour & Lecocq 2013). This is the case for example when sample GC analyses are performed on different columns or on the same column at different degree of column usury. A way to solve this problem is to use Kovats retention indices (Kovats 1958) instead of retention times (see 7. Kovats indices and GCKOVATS 1.0).

GCALIGNER only performs a first preliminary alignment on large GC dataset and accelerate analysis of multiple GC datasets. We strongly recommend to users to check all the alignment results obtained with GCALIGNER. The authors decline any responsibility in case of errors/misalignments.

7. Kovats indices and GCKOVATS 1.0

Alignment based on RT with large RT gaps between samples for identical compounds (e.g. samples analyzed at different degrees of columns usury or in different columns) has no meaning. This problem can be avoided by converting all the RT into Kovats retention indices (Kovats 1958). Kovats retention indices (KRI) are normalised system-independent measures. The KRI of a given compound is computed with:

$$KRI = 100\left(n_s + (n_l - n_s)\frac{RT_{sample} - RT_s}{RT_l - RT_s}\right)$$

with:

- n_s: the number of carbon atoms in the smaller n-alkane.
- n_l: the number of carbon atoms in the larger n-alkane.

GCKOVATS 1.0 converts RT into KRI (Fig. 4; for more details see Dellicour & Lecocq 2013). This conversion requires prior calibration points obtained by GC analyses of a standard solution of several alkanes (e.g. from C6 to C26). This calibration is used to convert RT of samples to KRI. A new calibration should be performed after any changes in methods or parameter (typically each day, as the users please).

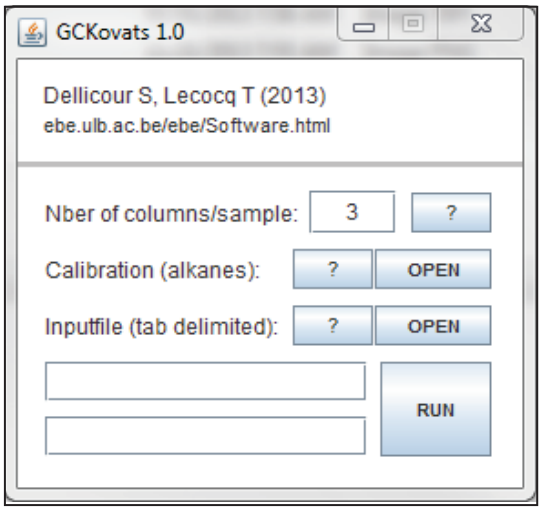


Figure 4. GCKOVATS GUI

GCKOVATS needs the same input file as GCALIGNER and an additional tab-delimited text file with the RT of all the n-alkanes from the calibration solution (first column) and the number of carbon atoms of each alkane (second column) (see Fig. 5). Please note that the two first lines are not read. They can contain any useful information for users but, as for GCALIGNER, this input file needs to present two lines before the n-alkanes RT and number of carbon atoms. **Moreover, we underline that the range of samples RT have to be included into the range of n-alkanes RT**.

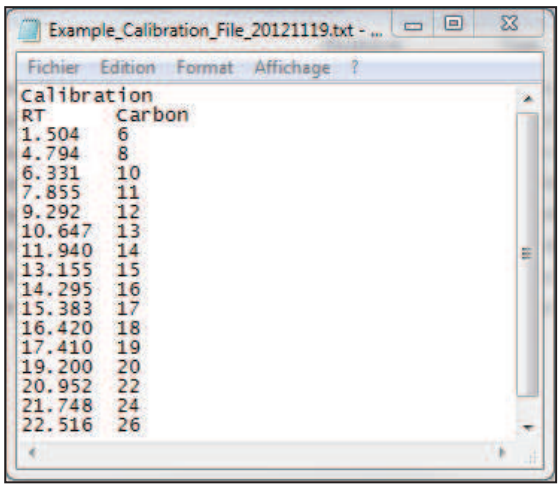


Figure 5. Second input file for GCKOVATS

How to cite GCALIGNER & GCKOVATS

Dellicour S, Lecocq T (2013) GCALIGNER 1.0: an aligment program to compute a multiple sample comparison data matrix from large eco-chemical datasets obtained by gas chromatography. *Journal of Separation Science*, **36**, 3206–3209.

GCALIGNER source code

Available at http://ebe.ulb.ac.be/ebe/GCAligner.html

Individual.java

```
package gcaligner;

import java.util.ArrayList;

// Class created by Simon Dellicour (April 2011).

public class Individual
{
	public int individualID;
	public ArrayList<String> columns;
	public ArrayList<String> retensionTimesString;

	public Individual (int i)
		{
			individualID = i;
			columns = new ArrayList<String>();
			retensionTimesString = new ArrayList<String>();
		}
}
```

InputFileReader.java

```
package gcaligner;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

// Class created by Simon Dellicour (April 2011).

public class InputFileReader
{
	public ArrayList<String> lines; // this ArrayList has all the information
coming from the input text file.
	public String inputName;
	public double maximumNumberOfDataRows;
	public double numberOfDataColumns;
	public double numberOfColumnsPerInjection;
	public double numberOfIndividuals;
	public ArrayList<Individual> individuals;
	public String individualsList;
	public String origins;

	public void readInput(String inputName) throws IOException
		{
			BufferedReader in = new BufferedReader(new
FileReader(inputName));
			String s1;
			ArrayList<String> lines0 = new ArrayList<String>();
			while ((s1 = in.readLine()) != null)
				lines0.add(s1);
			lines = lines0;
```

```
                    maximumNumberOfDataRows = lines.size() - 2;
                    in.close();

                    individualsList = lines.get(0);
                    origins = lines.get(1);
                    String[] line2 = lines.get(3).split("  ");
                    numberOfDataColumns = line2.length;
                    numberOfColumnsPerInjection =
MainClass.numberOfColumnsPerInjection;
                    numberOfIndividuals =
numberOfDataColumns/numberOfColumnsPerInjection;

                    individuals = new ArrayList<Individual>();
                    for (int i = 0; i < (int) numberOfIndividuals; i++)
                          {
                                Individual individual = new Individual(i + 1);
                                individuals.add(individual);
                          }

                    for (int i = 2; i < maximumNumberOfDataRows + 2; i++)
                          {
                                String s2 = new String();
                                s2 = lines.get(i);
                                String[] line = s2.split("");
                                for (int j = 0; j < numberOfIndividuals; j++)
                                      {
                                            if (j <
(line.length/numberOfColumnsPerInjection))
                                                  {
                                                        int k = (int)
(((j+1)*numberOfColumnsPerInjection) - numberOfColumnsPerInjection);
                                                        String string = new
String(); // string = line[k+0] + "    " + line[k+1]  + "  " + line[k+2];

                                                        for (int l = 0; l <
(int) numberOfColumnsPerInjection; l++)
                                                              {
                                                                    string =
string + line[k+l];
                                                                    if (l <
((int) numberOfColumnsPerInjection - 1))

      {

            string = string + " ";

      }
                                                              }

      individuals.get(j).columns.add(string);

      individuals.get(j).retensionTimesString.add(line[k+0]);
                                                  }
                                            else
                                                  {
                                                        String string = new
String(); // "      " + "  " + "  ";
                                                        for (int l = 0; l <
(int) numberOfColumnsPerInjection; l++)
                                                              {
                                                                    string =
string + "   "; // System.out.println(l+1);
                                                              }

      individuals.get(j).columns.add(string);

      individuals.get(j).retensionTimesString.add("");
                                                  }
                                      }
```

```
                        }
                }
}
```

Interface.java

```
package gcaligner;

import javax.swing.JFileChooser;
import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JTextField;
import java.awt.Color;
import javax.swing.border.LineBorder;
import javax.swing.SwingConstants;
import java.awt.Font;
import javax.swing.JPopupMenu;
import java.awt.Component;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Dimension;
import javax.swing.JTextPane;
import java.io.File;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Interface extends JFrame
{
	private static final long serialVersionUID = 1L;
	private JTextField inputFileName;
	private JTextField intervalWeight;
	private JTextField numberOfColumns;
	private JButton run = null;
	public Interface()
		{
			setResizable(false);
			//
setIconImage(Toolkit.getDefaultToolkit().getImage(Interface.class.getResource("/ima
ges/test.jpg")));
			getContentPane().setBackground(Color.WHITE);
			setTitle("GCAligner 1.0");
			setSize(283,243);
			getContentPane().setLayout(null);

			JPanel panel1 = new JPanel();
			panel1.setBorder(new LineBorder(new Color(192, 192, 192), 3));
			panel1.setBackground(Color.WHITE);
			panel1.setBounds(-11, -20, 296, 79);
			getContentPane().add(panel1);
			panel1.setLayout(null);

			JLabel lblDellicourSLecocq = new
JLabel("ebe.ulb.ac.be/ebe/Software.html");
			lblDellicourSLecocq.setBounds(25, 48, 167, 16);
			panel1.add(lblDellicourSLecocq);
			lblDellicourSLecocq.setFont(new Font("Arial", Font.PLAIN, 11));

			JLabel lblDellicourSLecocq_1 = new JLabel("Dellicour S, Lecocq T
(2013)");
			lblDellicourSLecocq_1.setBounds(25, 31, 167, 16);
			panel1.add(lblDellicourSLecocq_1);
			lblDellicourSLecocq_1.setFont(new Font("Arial", Font.PLAIN,
12));

			JPanel panel2 = new JPanel();
```

```
                    panel2.setBorder(new LineBorder(Color.LIGHT_GRAY, 3));
                    panel2.setBackground(Color.WHITE);
                    panel2.setBounds(-11, 51, 296, 173);
                    getContentPane().add(panel2);
                    panel2.setLayout(null);

                    JLabel lblNewLabel = new JLabel("Inputfile (tab delimited): ");
                    lblNewLabel.setFont(new Font("Arial", Font.PLAIN, 12));
                    lblNewLabel.setBounds(26, 95, 180, 16);
                    panel2.add(lblNewLabel);

                    inputFileName = new JTextField();
                    inputFileName.setFont(new Font("Arial", Font.PLAIN, 10));
                    // inputFileName.setText("(without a \".txt\" extension)");
                    inputFileName.setBounds(26, 125, 172, 22);
                    panel2.add(inputFileName);
                    inputFileName.setColumns(10);

                    run = new JButton("RUN");
                    run.setFont(new Font("Arial", Font.BOLD, 10));
                    run.setBounds(207, 125, 64, 22);
                    panel2.add(run);

                    JButton open = new JButton("OPEN");
                    open.addActionListener(new ActionListener() {
                          public void actionPerformed(ActionEvent arg0) {
                          }
                    });
                    open.setFont(new Font("Arial", Font.BOLD, 10));
                    open.setBounds(207, 92, 64, 22);
                    panel2.add(open);

                    JButton interrogationWeight = new JButton("?");
                    interrogationWeight.setBounds(227, 22, 44, 22);
                    panel2.add(interrogationWeight);
                    interrogationWeight.setFont(new Font("Arial", Font.PLAIN, 11));

                    final JPopupMenu popupMenu1 = new JPopupMenu();
                    popupMenu1.setBorderPainted(false);
                    popupMenu1.setFont(new Font("Arial", Font.PLAIN, 11));
                    popupMenu1.setPopupSize(new Dimension(500, 75));
                    addPopup(interrogationWeight, popupMenu1);

                    JTextPane text1 = new JTextPane();
                    text1.setFont(new Font("Arial", Font.PLAIN, 11));
                    text1.setText("This is the weight parameter alpha (\u03B1)
allowing to systematically increase or decrease the acceptance rate of a RT
\u201Cequivalence\u201D (corresponding to the same compound signal) to another RT.
Users are invited to begin with the default value of \u03B1 = 1 and then to
decrease (i.e. \u03B1 = 0.5, 0.25, 0.125) or increase (i.e. \u03B1 = 2, 3, 4) this
parameter in order to check if it can give better alignment results on their own
initial dataset.");
                    text1.setEditable(false);
                    popupMenu1.add(text1);

                    intervalWeight = new JTextField();
                    intervalWeight.setFont(new Font("Arial", Font.PLAIN, 12));
                    intervalWeight.setBounds(171, 21, 48, 24);
                    panel2.add(intervalWeight);
                    intervalWeight.setHorizontalAlignment(SwingConstants.CENTER);
                    intervalWeight.setText("1");
                    intervalWeight.setColumns(10);

                    JLabel lblIntervalWeight = new JLabel("Weight parameter alpha:
");
                    lblIntervalWeight.setBounds(26, 25, 189, 16);
                    panel2.add(lblIntervalWeight);
                    lblIntervalWeight.setFont(new Font("Arial", Font.PLAIN, 12));
```

```
                    JLabel lblNberOfColumnsinjection = new JLabel("Nber of
columns/sample:");
                    lblNberOfColumnsinjection.setBounds(26, 58, 193, 16);
                    panel2.add(lblNberOfColumnsinjection);
                    lblNberOfColumnsinjection.setFont(new Font("Arial", Font.PLAIN,
12));

                    numberOfColumns = new JTextField();
                    numberOfColumns.setFont(new Font("Arial", Font.PLAIN, 12));
                    numberOfColumns.setBounds(171, 54, 48, 24);
                    panel2.add(numberOfColumns);
                    numberOfColumns.setHorizontalAlignment(SwingConstants.CENTER);
                    numberOfColumns.setText("3");
                    numberOfColumns.setColumns(10);

                    JButton interrogationNberOfColumns = new JButton("?");
                    interrogationNberOfColumns.setBounds(227, 55, 44, 22);
                    panel2.add(interrogationNberOfColumns);
                    interrogationNberOfColumns.setFont(new Font("Arial", Font.PLAIN,
11));

                    final JPopupMenu popupMenu2 = new JPopupMenu();
                    popupMenu2.setBorderPainted(false);
                    popupMenu2.setFont(new Font("Arial", Font.PLAIN, 11));
                    popupMenu2.setPopupSize(new Dimension(500, 35));
                    addPopup(interrogationNberOfColumns, popupMenu2);

                    JTextPane text2 = new JTextPane();
                    text2.setFont(new Font("Arial", Font.PLAIN, 11));
                    text2.setEditable(false);
                    text2.setText("This the number of columns per injection/sample
in the input file (including the compulsory first columns with the retention
times).");
                    popupMenu2.add(text2);

                    JButton interrogationInputFile = new JButton("?");
                    interrogationInputFile.setFont(new Font("Arial", Font.PLAIN,
11));
                    interrogationInputFile.setBounds(162, 92, 44, 22);
                    panel2.add(interrogationInputFile);

                    final JPopupMenu popupMenu3 = new JPopupMenu();
                    popupMenu3.setBorderPainted(false);
                    popupMenu3.setFont(new Font("Arial", Font.PLAIN, 11));
                    popupMenu3.setPopupSize(new Dimension(500, 90));
                    addPopup(interrogationInputFile, popupMenu3);

                    JTextPane text3 = new JTextPane();
                    text3.setFont(new Font("Arial", Font.PLAIN, 11));
                    text3.setText("GCAligner needs input files in a \u00AB.txt\u00BB
tab delimited format. The first column of each sample/injection must always be the
RT column. The program does not read the other columns, which will only be copied
in the output file. The number of additional columns to the RT column is not
limited but has to be specified before running the program (cfr. GCAligner manual).
Empty data are not allowed. The two first lines are not read. They can contain any
useful information for users but input files need to present two lines before the
different columns per sample.\r");
                    text3.setEditable(false);
                    popupMenu3.add(text3);

                    interrogationNberOfColumns.addMouseListener(new MouseAdapter() {
                          @Override
                          public void mouseClicked(MouseEvent arg0) {
                                popupMenu2.show(arg0.getComponent(),
                                            arg0.getX(), arg0.getY());
                          }
                    });

                    interrogationWeight.addMouseListener(new MouseAdapter() {
```

```
                        @Override
                        public void mouseClicked(MouseEvent arg0) {
                                popupMenu1.show(arg0.getComponent(),
                                                arg0.getX(), arg0.getY());
                        }
                });

                interrogationInputFile.addMouseListener(new MouseAdapter() {
                        @Override
                        public void mouseClicked(MouseEvent arg0) {
                                popupMenu3.show(arg0.getComponent(),
                                                arg0.getX(), arg0.getY());
                        }
                });

                final JFileChooser fileChooser = new JFileChooser();

                open.addMouseListener(new MouseAdapter() {
                        public void mouseClicked(MouseEvent arg0) {
                        int returnVal = fileChooser.showOpenDialog(null);
                        if (returnVal == JFileChooser.APPROVE_OPTION)
                                {
                                        File selection =
fileChooser.getSelectedFile();

        inputFileName.setText(selection.getAbsolutePath());
                                }
                        }
                });
        }

        JButton getJButton()
                {
                        return run;
                }

        public String getInputNameText()
                {
                        String inputNameText = inputFileName.getText();
                        return inputNameText;
                }

        public String getIntervalWeightText()
                {
                        String intervalWeightText = intervalWeight.getText();
                        return intervalWeightText;
                }

        public String getNumberOfColumnsPerInjectionText()
                {
                        String getNumberOfColumnsPerInjectionText =
numberOfColumns.getText();
                        return getNumberOfColumnsPerInjectionText;
                }

        private static void addPopup(Component component, final JPopupMenu popup)
                {
                }
}
```

MainClass.java

```
package gcaligner;

// Class created by Simon Dellicour (April 2011).

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.util.ArrayList;
import javax.swing.JButton;
import javax.swing.JFrame;

public class MainClass
{
	public static ArrayList<String> finalArray;
	// data coming from the interface :
	public static Interface window;
	public static JButton start;
	public static String inputName;
	public static String intervalWeightText;
	public static double intervalWeight;
	public static String numberOfColumnsPerInjectionText;
	public static double numberOfColumnsPerInjection;

	// data coming from the inputFileReader :
	public static ArrayList<ArrayList <Double>> originalArray;

	public static void main(String[] args) {
		window = new Interface(); // creation of the interface.
		window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		window.setVisible(true); // necessary line to add to have a nice
window.
		// window.setIconImage(new ImageIcon("Icon.PNG").getImage());
		start = window.getJButton();

		start.addActionListener(new ActionListener()
			{
				public void actionPerformed(ActionEvent e)
					{
						// inputName = (window.getInputNameText() +
".txt");
						inputName = window.getInputNameText();
						intervalWeightText =
window.getIntervalWeightText();
						intervalWeight =
Double.valueOf(intervalWeightText);
						numberOfColumnsPerInjectionText =
window.getNumberOfColumnsPerInjectionText();
						numberOfColumnsPerInjection =
Double.valueOf(numberOfColumnsPerInjectionText);

						InputFileReader input = new
InputFileReader();
							try {
								input.readInput(inputName);
							} catch (IOException e3) {
								// TODO Auto-generated catch
block
								e3.printStackTrace();
							}

						finalArray = new ArrayList<String>();
						finalArray.add(input.individualsList);
						finalArray.add(input.origins);

						double highestGoodRetentionTime = 0;
```

```
                                        int c = 0; // (counter).
                                        int e2 = 0;
                                        for (int i = 0; i <
input.numberOfIndividuals; i ++)
                                              {
                                                    if
(input.individuals.get(i).columns.size() > 1)
                                                          {
                                                                e2 = e2 + 1;
                                                          }
                                              }

                                        while (e2 > 1) // while all the individuals'
first lines are not analyzed.
                                              {
                                                    String row = new String();
                                                    ArrayList<Double>
rowRetentionTimes = new ArrayList<Double>(); // this ArrayList will contain all the
retention times of the considered row.
                                                    ArrayList<Double>
nextRowRetentionTimes = new ArrayList<Double>(); // this ArrayList will contain all
the retention times of the considered row.
                                                    ArrayList<Double>
rowIndividualsID = new ArrayList<Double>(); // this ArrayList will contain all the
individual ID's of the considered row.
                                                    ArrayList<Double>
goodRetentionID = new ArrayList<Double>(); // Arraylist which will contain only
individuals ID with a "correct" retention times.

                                                    // construction of the
rowRententionTimes and rowIndividuals ArrayList's :
                                                    for (int j = 0; j <
input.numberOfIndividuals; j++)
                                                          {
                                                                if
(input.individuals.get(j).retensionTimesString.size() > 1
                                                                                        &&
input.individuals.get(j).retensionTimesString.get(0).length() > 0
                                                                                        &&
input.individuals.get(j).retensionTimesString.get(1).length() > 0)
                                                                                  {

      rowRetentionTimes.add(Double.valueOf(input.individuals.get(j).retensionTimesS
tring.get(0)));

      nextRowRetentionTimes.add(Double.valueOf(input.individuals.get(j).retensionTi
mesString.get(1)));

      rowIndividualsID.add(Double.valueOf(input.individuals.get(j).individualID));
                                                                                        }
                                                          }

                                                    // (1) determination of the
weakest retention time :
                                                    double weakestTime =
Double.valueOf(rowRetentionTimes.get(0)).doubleValue();
                                                    double weakestTimeID =
Double.valueOf(rowIndividualsID.get(0)).doubleValue();
                                                    double tf1 =
Double.valueOf(input.individuals.get((int) weakestTimeID -
1).retensionTimesString.get(1)).doubleValue(); // NOUVEAU !!!
                                                    for (int j = 1; j <
rowIndividualsID.size(); j++)
                                                          {
                                                                if
(rowRetentionTimes.get(j).doubleValue() < weakestTime)
                                                                                  {

      weakestTime = rowRetentionTimes.get(j).doubleValue();
```

```
	weakestTimeID = rowIndividualsID.get(j).doubleValue();
                                                                        }
                                                                if
(nextRowRetentionTimes.get(j).doubleValue() < tf1) // NOUVEAU !!!
                                                                {
                                                                        tf1 =
nextRowRetentionTimes.get(j).doubleValue();
                                                                }
                                                        }

	goodRetentionID.add(Double.valueOf(weakestTimeID));

                                                        for (int j = 0; j <
rowIndividualsID.size(); j++) // to remove the weakestTimeID of the
rowIndividualsID;
                                                        {
                                                                if
(rowIndividualsID.get(j).doubleValue() == weakestTimeID)
                                                                {

	rowIndividualsID.remove(j);

	rowRetentionTimes.remove(j);
                                                                }
                                                        }

                                                        double tf0 =
Double.valueOf(input.individuals.get((int) weakestTimeID -
1).retensionTimesString.get(0)).doubleValue(); // NOUVEAU !!!

                                                        // comparison with the other
retention times on the same row :
                                                        int d = (int)
rowIndividualsID.size();
                                                        while (d > 0)
                                                                {
                                                                        double nearestTime
= rowRetentionTimes.get(0).doubleValue(); // initialization.
                                                                        double
nearestTimeID = rowIndividualsID.get(0).doubleValue(); // initialization.
                                                                        double delta =
nearestTime - weakestTime;

                                                                        // (2)
determination of the nearest rentention time :
                                                                        for (int j = 1; j
< rowIndividualsID.size(); j++)
                                                                        {


                                                                                if
(rowRetentionTimes.get(j).doubleValue() - weakestTime < delta)

	{

		delta = rowRetentionTimes.get(j).doubleValue() - weakestTime;

		nearestTimeID = (int) rowIndividualsID.get(j).doubleValue();

	}
                                                                        }

                                                                        // (3) comparison
between the nearest and weakest retention times (+ cfr. external notes about the
algorithm) :
                                                                        if
(input.individuals.get((int) weakestTimeID -
```

```
1).retensionTimesString.get(1).length() > 0

                                                                                &&
input.individuals.get((int) nearestTimeID - 1).retensionTimesString.get(1).length()
> 0)
                                                                          {
                                                                                //
double tf1 = Double.valueOf(input.individuals.get((int) weakestTimeID -
1).retensionTimesString.get(1)).doubleValue();
                                                                                //
double tf0 = Double.valueOf(input.individuals.get((int) weakestTimeID -
1).retensionTimesString.get(0)).doubleValue();

      double tp1 = Double.valueOf(input.individuals.get((int) nearestTimeID -
1).retensionTimesString.get(1)).doubleValue();

      double tp0 = Double.valueOf(input.individuals.get((int) nearestTimeID -
1).retensionTimesString.get(0)).doubleValue();
                                                                                if
((tf1-tp0) > ((tp0-tf0)*intervalWeight))

      {                                                          //
System.out.println(tf1 + "-" + tp0 + " " + tf0 + "-" + tp0);

            // (2.2)

            if ((tp1-tp0) > ((tp0-tf0)*intervalWeight))

                  {

                        // (2.3)

                        if ((tf0-highestGoodRetentionTime) > ((tp0-
tf0)*intervalWeight)) // NOUVEAU !!!

                              {


      goodRetentionID.add(Double.valueOf(nearestTimeID));



                                    for (int k = 0; k < rowIndividualsID.size();
k++) // to remove the nearestTimeID of the rowIndividualsID;

                                          {

                                                if
(rowIndividualsID.get(k).doubleValue() == nearestTimeID)

                                                {


      rowIndividualsID.remove(k);


      rowRetentionTimes.remove(k);

                                                }

                                          }

                                    d = d - 1;

                              }

                        else // NOUVEAU !!!

                              {
```

```
                                        for (int k = 0; k < rowIndividualsID.size();
k++) // to remove the nearestTimeID of the rowIndividualsID;
                                              {
                                                    if
(rowIndividualsID.get(k).doubleValue() == nearestTimeID)
                                                          {

      rowIndividualsID.remove(k);

      rowRetentionTimes.remove(k);
                                                          }
                                              }
                                        d = d - 1;
                                  }
                  }
            else
                  {
                        for (int k = 0; k < rowIndividualsID.size(); k++) // to
remove the nearestTimeID of the rowIndividualsID;
                                  {
                                        if (rowIndividualsID.get(k).doubleValue() ==
nearestTimeID)
                                              {
                                                    rowIndividualsID.remove(k);
                                                    rowRetentionTimes.remove(k);
                                              }
                                  }
                        d = d - 1;
                  }
      }
                                                                              else
      {
            for (int k = 0; k < rowIndividualsID.size(); k++) // to remove the
nearestTimeID of the rowIndividualsID;
                  {
                        if (rowIndividualsID.get(k).doubleValue() ==
nearestTimeID)
                              {
                                    rowIndividualsID.remove(k);
```

```
                                        rowRetentionTimes.remove(k);

                                }

                        }

                d = d - 1;

        }

                                                                                if
(tp1 < tf1) // NOUVEAU !!! : if (tp1 > tf1), the alignment consider bigger
intervals and if (tp1 < tf1), smallest intervals.

        {

                tf1 = tp1;

        }
                                                                                }

                                                                        // this step
shouldn't be necessary because last individuals' line are not analyzed.
                                                                        else if
(input.individuals.get((int) nearestTimeID -
1).retensionTimesString.get(1).length() > 0)
                                                                                {
        System.out.println("inside 1");
                                                                                ////
double tf0 = Double.valueOf(input.individuals.get((int) weakestTimeID -
1).retensionTimesString.get(0)).doubleValue();

        double tp1 = Double.valueOf(input.individuals.get((int) nearestTimeID -
1).retensionTimesString.get(1)).doubleValue();

        double tp0 = Double.valueOf(input.individuals.get((int) nearestTimeID -
1).retensionTimesString.get(0)).doubleValue();


                if ((tp1-tp0) > (tp0-tf0))

                        {

                                goodRetentionID.add(Double.valueOf(nearestTimeID));


                                for (int k = 0; k < rowIndividualsID.size(); k++) // to
remove the nearestTimeID of the rowIndividualsID;

                                        {

                                                if (rowIndividualsID.get(k).doubleValue() ==
nearestTimeID)

                                                        {

                                                                rowIndividualsID.remove(k);

                                                                rowRetentionTimes.remove(k);

                                                        }

                                        }

                                d = d - 1;
```

```
                    }
             else
                    {
                          for (int k = 0; k < rowIndividualsID.size(); k++) // to
remove the nearestTimeID of the rowIndividualsID;
                                {
                                      if (rowIndividualsID.get(k).doubleValue() ==
nearestTimeID)
                                            {
                                                  rowIndividualsID.remove(k);
                                                  rowRetentionTimes.remove(k);
                                            }
                                }
                          d = d - 1;
                    }
      }
                                                                  // this step
shouldn't be necessary because last individuals' line are not analyzed.
                                                                  else if
(input.individuals.get((int) weakestTimeID -
1).retensionTimesString.get(1).length() > 0)
                                                                        {
System.out.println("inside 2");
                                                                        ////
double tf1 = Double.valueOf(input.individuals.get((int) weakestTimeID -
1).retensionTimesString.get(1)).doubleValue();
                                                                        ////
double tf0 = Double.valueOf(input.individuals.get((int) weakestTimeID -
1).retensionTimesString.get(0)).doubleValue();
                                                                        //
double tp1 = Double.valueOf(input.individuals.get((int) nearestTimeID -
1).retensionTimesString.get(1)).doubleValue();
      double tp0 = Double.valueOf(input.individuals.get((int) nearestTimeID -
1).retensionTimesString.get(0)).doubleValue();
                                                                        if
((tf1-tp0) > (tp0-tf0))
      {
             for (int j = 0; j < rowIndividualsID.size(); j++) // to remove the
weakestTimeID of the rowIndividualsID;
                    {
                          goodRetentionID.add(Double.valueOf(nearestTimeID));

                          for (int k = 0; k < rowIndividualsID.size(); k++) // to
remove the nearestTimeID of the rowIndividualsID;
                                {
```

```
                                        if (rowIndividualsID.get(k).doubleValue() ==
nearestTimeID)

                                              {

                                                    rowIndividualsID.remove(k);

                                                    rowRetentionTimes.remove(k);

                                              }

                                  }

                            d = d - 1;

                      }

       }
                                                                                  else

       {

              for (int k = 0; k < rowIndividualsID.size(); k++) // to remove the
nearestTimeID of the rowIndividualsID;

                      {

                            if (rowIndividualsID.get(k).doubleValue() ==
nearestTimeID)

                                  {

                                        rowIndividualsID.remove(k);

                                        rowRetentionTimes.remove(k);

                                  }

                      }

              d = d - 1;

       }
                                                                        }
                                                                  } //
System.out.println();

                                                      // (4) individuals' lines with
an "accepted" (corresponding) retention times are put at the end of a the output
ArrayList ("finalArray").
                                                      for (int j = 0; j <
input.numberOfIndividuals; j++)
                                                            {
                                                                  int a = 0;
                                                                  for (int k = 0; k
< goodRetentionID.size(); k++)
                                                                        {
                                                                              if
(input.individuals.get(j).individualID == goodRetentionID.get(k)

              && input.individuals.get(j).columns.size() > 1)

       {

              if
(Double.valueOf(input.individuals.get(j).retensionTimesString.get(0)) >
highestGoodRetentionTime)
```

```
                    {

                          highestGoodRetentionTime =
Double.valueOf(input.individuals.get(j).retensionTimesString.get(0));

                    }

             row = row + input.individuals.get(j).columns.get(0) + "    ";

             input.individuals.get(j).columns.remove(0);

             input.individuals.get(j).retensionTimesString.remove(0);

             a = a + 1;

      }
                                                                  }
                                                            if (a == 0)
                                                                  {
                                                                        //
row = row + "" + "  " + "  ";                                           for
(int l = 0; l < (int) numberOfColumnsPerInjection; l++)

      {

             row = row + "";

      }
                                                                  }
                                                      }

                                                   e2 = 0; // "e2" counter update.
                                                   for (int j = 0; j <
input.numberOfIndividuals; j ++)
                                                   {
                                                      if
(input.individuals.get(j).retensionTimesString.size() > 1
                                                                  &&
input.individuals.get(j).retensionTimesString.get(0).length() > 0
                                                                  &&
input.individuals.get(j).retensionTimesString.get(1).length() > 0)
                                                            {
                                                                  e2 = e2 +
1;
                                                            }
                                                   }

                                                   finalArray.add(row); c = c + 1;
// System.out.println(e2);
                                             }

                                       // construction of the .txt or .xls output
file :
                                       String finalRow = new String();
                                       for (int i = 0; i <
input.numberOfIndividuals; i ++)
                                             {
                                                   finalRow = finalRow +
input.individuals.get(i).columns.get(0) + "   ";
                                             }
                                       finalArray.add(finalRow);

                                       OutputFileWriter output1 = new
OutputFileWriter();
                                             try {

      output1.writeOutput1(finalArray, "GCAligner_output_file.xls");
```

```
                                                        } catch (IOException e1) {
                                                              // TODO Auto-generated
catch block
                                                              e1.printStackTrace();
                                                        }
                                    } // end of "addActionListener".
                              });
       } // end of "main".
}
```

OutputFileWriter.java

```
package gcaligner;

// Class created by Simon Dellicour (March 2010).

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;

public class OutputFileWriter
{
       public void writeOutput1(ArrayList<String> string0, String name0) throws
IOException
              {
                     ArrayList<String> string = string0;
                     String name = name0;

                     FileWriter file = new FileWriter(name);
                     BufferedWriter buffer = new BufferedWriter(file);
                     PrintWriter out = new PrintWriter(buffer);

                     for (int i = 0; i < string.size(); i++)
                            {
                                   out.println(string.get(i));
                            }

                     out.close();
                     buffer.close();
                     file.close();
              }

       public static void writeOutput2(ArrayList<String> string0, String name0)
throws IOException
              {
                     ArrayList<String> string = string0;
                     String name = name0;

                     FileWriter file = new FileWriter(name);
                     BufferedWriter buffer = new BufferedWriter(file);
                     PrintWriter out = new PrintWriter(buffer);

                     for (int i = 0; i < string.size(); i++)
                            {
                                   out.print(string.get(i) + " ");
                            }

                     out.close();
                     buffer.close();
                     file.close();
              }
}
```

GCKOVATS source code

Available at http://ebe.ulb.ac.be/ebe/GCAligner.html

Individual.java

```
package gcaligner;
package gckovats;

import java.util.ArrayList;

// Class created by Simon Dellicour (April 2011).

public class Individual
{
	public int individualID;
	public ArrayList<String> columns;
	public ArrayList<String> retensionTimesString;
	public ArrayList<String> kovatsI;
	public ArrayList<String> columnsWithoutRetentionTimes;

	public Individual (int i)
		{
			individualID = i;
			columns = new ArrayList<String>();
			retensionTimesString = new ArrayList<String>();
			kovatsI = new ArrayList<String>();
			columnsWithoutRetentionTimes = new ArrayList<String>();
		}
}
```

InputFileReader.java

```
package gckovats;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;

// Class created by Simon Dellicour (April 2011 & November 2012).

public class InputFileReader
{
	public ArrayList<String> lines1;
	public ArrayList<String> lines2;
	public String inputName;
	public double maximumNumberOfDataRows;
	public double numberOfDataColumns;
	public double numberOfColumnsPerInjection;
	public double numberOfIndividuals;
	public ArrayList<Individual> individuals;
	public String individualsList;
	public String origins;
	public ArrayList<String> calibrationRetentionTimes;
	public ArrayList<String> numberOfCarbons;

	// IMPORTANT NOTE : to easily understand these distinctions, it is quite
better to look at the input file structure...
```

```
	public void readInput(String inputName) throws IOException
		{
			BufferedReader in = new BufferedReader(new
FileReader(inputName));
			String s1;
			ArrayList<String> lines0 = new ArrayList<String>();
			while ((s1 = in.readLine()) != null)
				lines0.add(s1);
			lines1 = lines0;
			maximumNumberOfDataRows = lines1.size() - 2;
			in.close();

			individualsList = lines1.get(0);
			origins = lines1.get(1);
			String[] line2 = lines1.get(2).split(" ");
			numberOfDataColumns = line2.length;
			numberOfColumnsPerInjection =
MainClass.numberOfColumnsPerInjection;
			numberOfIndividuals =
numberOfDataColumns/numberOfColumnsPerInjection;

			individuals = new ArrayList<Individual>();
			for (int i = 0; i < (int) numberOfIndividuals; i++)
				{
					Individual individual = new Individual(i + 1);
					individuals.add(individual);
				}

			for (int i = 2; i < maximumNumberOfDataRows + 2; i++)
				{
					String s2 = new String();
					s2 = lines1.get(i);
					String[] line = s2.split("");
					for (int j = 0; j < numberOfIndividuals; j++)
						{
							if (j <
(line.length/numberOfColumnsPerInjection))
								{
									int k = (int)
(((j+1)*numberOfColumnsPerInjection) - numberOfColumnsPerInjection);
									String string1 = new
String();
									String string2 = new
String();
									for (int l = 0; l <
(int) numberOfColumnsPerInjection; l++)
										{
											string1 =
string1 + line[k+l];
											if (l <
((int) numberOfColumnsPerInjection - 1))

	{

		string1 = string1 + "     ";

	}
										}
									for (int l = 1; l <
(int) numberOfColumnsPerInjection; l++)
										{
											string2 =
string2 + line[k+l];
											if (l <
((int) numberOfColumnsPerInjection - 1))

	{

		string2 = string2 + "     ";
```

```
	}
                                                                  }

	individuals.get(j).columns.add(string1);

	individuals.get(j).retensionTimesString.add(line[k+0]);

	individuals.get(j).columnsWithoutRetentionTimes.add(string2);
                                                  }
                                            else
                                                  {
                                                        String string = new
String(); // "      " + "  " + "  ";

                                                        for (int l = 0; l <
(int) numberOfColumnsPerInjection; l++)
                                                        {
                                                              string =
string + "    "; // System.out.println(l+1);
                                                        }


	individuals.get(j).columns.add(string);

	individuals.get(j).retensionTimesString.add("");
                                                  }
                                      }
                              }
                  }

	public void readCalibration(String calibrationName) throws IOException
                  {
                        BufferedReader in = new BufferedReader(new
FileReader(calibrationName));
                        String s1;
                        ArrayList<String> lines0 = new ArrayList<String>();
                        while ((s1 = in.readLine()) != null)
                              lines0.add(s1);
                        lines2 = lines0;
                        double maximumNumberOfCalibrationRows = lines2.size() - 2;
                        in.close();

                        calibrationRetentionTimes = new ArrayList<String>();
                        numberOfCarbons = new ArrayList<String>();

                        for (int i = 2; i < maximumNumberOfCalibrationRows + 2; i++)
                              {
                                    String s2 = new String();
                                    s2 = lines2.get(i);
                                    String[] line = s2.split("");
                                    calibrationRetentionTimes.add(line[0]);
                                    numberOfCarbons.add(line[1]);
                              }
                  }
}
```

Interface.java

```
package gckovats;

import javax.swing.JFileChooser;
import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JTextField;
import java.awt.Color;
import javax.swing.border.LineBorder;
import javax.swing.SwingConstants;
import java.awt.Font;
import javax.swing.JPopupMenu;
import java.awt.Component;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.Dimension;
import javax.swing.JTextPane;
import java.io.File;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Interface extends JFrame
{
	private static final long serialVersionUID = 1L;
	private JTextField inputFileName;
	private JTextField numberOfColumns;
	private JButton run = null;
	private JTextField calibrationFileName;
	public Interface()
		{
			setResizable(false);
			//
setIconImage(Toolkit.getDefaultToolkit().getImage(Interface.class.getResource("/ima
ges/test.jpg")));
			getContentPane().setBackground(Color.WHITE);
			setTitle("GCKovats 1.0");
			setSize(283, 263);
			getContentPane().setLayout(null);

			JPanel panel1 = new JPanel();
			panel1.setBorder(new LineBorder(new Color(192, 192, 192), 3));
			panel1.setBackground(Color.WHITE);
			panel1.setBounds(-11, -20, 304, 79);
			getContentPane().add(panel1);
			panel1.setLayout(null);

			JLabel lblDellicourSLecocq = new
JLabel("ebe.ulb.ac.be/ebe/Software.html");
			lblDellicourSLecocq.setBounds(25, 48, 167, 16);
			panel1.add(lblDellicourSLecocq);
			lblDellicourSLecocq.setFont(new Font("Arial", Font.PLAIN, 11));

			JLabel lblDellicourSLecocq_1 = new JLabel("Dellicour S, Lecocq T
(2013)");
			lblDellicourSLecocq_1.setBounds(25, 31, 167, 16);
			panel1.add(lblDellicourSLecocq_1);
			lblDellicourSLecocq_1.setFont(new Font("Arial", Font.PLAIN,
12));

			JPanel panel2 = new JPanel();
			panel2.setBorder(new LineBorder(Color.LIGHT_GRAY, 3));
			panel2.setBackground(Color.WHITE);
			panel2.setBounds(-11, 51, 304, 201);
			getContentPane().add(panel2);
```

```
                    panel2.setLayout(null);

                    JLabel label1 = new JLabel("Inputfile (tab delimited): ");
                    label1.setFont(new Font("Arial", Font.PLAIN, 12));
                    label1.setBounds(26, 87, 177, 16);
                    panel2.add(label1);

                    inputFileName = new JTextField();
                    inputFileName.setFont(new Font("Arial", Font.PLAIN, 10));
                    // inputFileName.setText("(without a \".txt\" extension)");
                    inputFileName.setBounds(26, 145, 177, 22);
                    panel2.add(inputFileName);
                    inputFileName.setColumns(10);

                    run = new JButton("RUN");
                    run.setFont(new Font("Arial", Font.BOLD, 10));
                    run.setBounds(210, 115, 61, 52);
                    panel2.add(run);

                    JButton open1 = new JButton("OPEN");
                    open1.addActionListener(new ActionListener() {
                          public void actionPerformed(ActionEvent arg0) {
                          }
                    });
                    open1.setFont(new Font("Arial", Font.BOLD, 10));
                    open1.setBounds(207, 84, 64, 22);
                    panel2.add(open1);

                    JLabel lblNberOfColumnsinjection = new JLabel("Nber of
columns/sample:");
                    lblNberOfColumnsinjection.setBounds(26, 25, 193, 16);
                    panel2.add(lblNberOfColumnsinjection);
                    lblNberOfColumnsinjection.setFont(new Font("Arial", Font.PLAIN,
12));

                    numberOfColumns = new JTextField();
                    numberOfColumns.setFont(new Font("Arial", Font.PLAIN, 12));
                    numberOfColumns.setBounds(171, 21, 48, 24);
                    panel2.add(numberOfColumns);
                    numberOfColumns.setHorizontalAlignment(SwingConstants.CENTER);
                    numberOfColumns.setText("3");
                    numberOfColumns.setColumns(10);

                    JButton interrogationNberOfColumns = new JButton("?");
                    interrogationNberOfColumns.setBounds(227, 22, 44, 22);
                    panel2.add(interrogationNberOfColumns);
                    interrogationNberOfColumns.setFont(new Font("Arial", Font.PLAIN,
11));

                    final JPopupMenu popupMenu1 = new JPopupMenu();
                    popupMenu1.setBorderPainted(false);
                    popupMenu1.setFont(new Font("Arial", Font.PLAIN, 11));
                    popupMenu1.setPopupSize(new Dimension(400, 35));
                    addPopup(interrogationNberOfColumns, popupMenu1);

                    JTextPane text2 = new JTextPane();
                    text2.setFont(new Font("Arial", Font.PLAIN, 11));
                    text2.setEditable(false);
                    text2.setText("This the number of columns per injection/sample
in the input file (including the compulsory first columns with the retention
times).");
                    popupMenu1.add(text2);

                    JButton interrogationInputFile = new JButton("?");
                    interrogationInputFile.setFont(new Font("Arial", Font.PLAIN,
11));
                    interrogationInputFile.setBounds(162, 84, 44, 22);
                    panel2.add(interrogationInputFile);
```

```
                    final JPopupMenu popupMenu3 = new JPopupMenu();
                    popupMenu3.setBorderPainted(false);
                    popupMenu3.setFont(new Font("Arial", Font.PLAIN, 11));
                    popupMenu3.setPopupSize(new Dimension(400, 110));
                    addPopup(interrogationInputFile, popupMenu3);

                    JTextPane text3 = new JTextPane();
                    text3.setFont(new Font("Arial", Font.PLAIN, 11));
                    text3.setText("The second input file GCKovats needs is the same
as for GCAligner. In this tab delimited file, the first column of each
sample/injection must always be the RT column. The program does not read the other
columns, which will only be copied in the output file. The number of additional
columns to the RT column is not limited but has to be specified before running the
program (cfr. software manual). Empty data are not allowed. The two first lines are
not read. They can contain any useful information for users but input files need to
present two lines before the different columns per sample.\r");
                    text3.setEditable(false);
                    popupMenu3.add(text3);

                    JLabel label2 = new JLabel("Calibration (alkanes):");
                    label2.setFont(new Font("Arial", Font.PLAIN, 12));
                    label2.setBounds(26, 56, 177, 16);
                    panel2.add(label2);

                    JButton interrogationCalibrationFile = new JButton("?");
                    interrogationCalibrationFile.setFont(new Font("Arial",
Font.PLAIN, 11));
                    interrogationCalibrationFile.setBounds(162, 53, 44, 22);
                    panel2.add(interrogationCalibrationFile);

                    final JPopupMenu popupMenu2 = new JPopupMenu();
                    popupMenu2.setPopupSize(new Dimension(400, 110));
                    popupMenu2.setFont(new Font("Arial", Font.PLAIN, 11));
                    popupMenu2.setBorderPainted(false);
                    addPopup(interrogationCalibrationFile, popupMenu2);

                    JTextPane txtpnGckovatsFirstlyNeeds = new JTextPane();
                    txtpnGckovatsFirstlyNeeds.setText("GCKovats firstly needs a
\"calibration file\": a \u00AB.txt\u00BB tab delimited file displaying the RT of
several or all the n-alkanes from the calibration solution (first column) and the
number of carbon atoms of each alkane (second column). Please note that the two
first lines are not read. They can contain any useful information for users but, as
for GCAligner, this input file needs to present two lines before the n-alkanes RT
and number of carbon atoms. It is also important to underline that the range of
samples RT have to be included into the range of n-alkanes RT. See the software
manual for further details.\r");
                    txtpnGckovatsFirstlyNeeds.setFont(new Font("Arial", Font.PLAIN,
11));
                    txtpnGckovatsFirstlyNeeds.setEditable(false);
                    popupMenu2.add(txtpnGckovatsFirstlyNeeds);

                    JButton open2 = new JButton("OPEN");
                    open2.setFont(new Font("Arial", Font.BOLD, 10));
                    open2.setBounds(207, 53, 64, 22);
                    panel2.add(open2);

                    calibrationFileName = new JTextField();
                    calibrationFileName.setFont(new Font("Arial", Font.PLAIN, 10));
                    calibrationFileName.setColumns(10);
                    calibrationFileName.setBounds(26, 115, 177, 22);
                    panel2.add(calibrationFileName);

                    interrogationNberOfColumns.addMouseListener(new MouseAdapter() {
                          @Override
                          public void mouseClicked(MouseEvent arg0) {
                                popupMenu1.show(arg0.getComponent(),
                                            arg0.getX(), arg0.getY());
                          }
                    });
```

```
                    interrogationCalibrationFile.addMouseListener(new MouseAdapter()
{
                          @Override
                          public void mouseClicked(MouseEvent arg0) {
                                popupMenu2.show(arg0.getComponent(),
                                            arg0.getX(), arg0.getY());
                          }
                    });

                    interrogationInputFile.addMouseListener(new MouseAdapter() {
                          @Override
                          public void mouseClicked(MouseEvent arg0) {
                                popupMenu3.show(arg0.getComponent(),
                                            arg0.getX(), arg0.getY());
                          }
                    });

                    final JFileChooser fileChooser1 = new JFileChooser();
                    open1.addMouseListener(new MouseAdapter() {
                          public void mouseClicked(MouseEvent arg0) {
                          int returnVal = fileChooser1.showOpenDialog(null);
                          if (returnVal == JFileChooser.APPROVE_OPTION)
                                {
                                      File selection =
fileChooser1.getSelectedFile();

      inputFileName.setText(selection.getAbsolutePath());
                                }
                          }
                    });

                    final JFileChooser fileChooser2 = new JFileChooser();
                    open2.addMouseListener(new MouseAdapter() {
                          public void mouseClicked(MouseEvent arg0) {
                          int returnVal = fileChooser2.showOpenDialog(null);
                          if (returnVal == JFileChooser.APPROVE_OPTION)
                                {
                                      File selection =
fileChooser2.getSelectedFile();

      calibrationFileName.setText(selection.getAbsolutePath());
                                }
                          }
                    });
            }

      JButton getJButton()
            {
                  return run;
            }

      public String getInputNameText()
            {
                  String inputNameText = inputFileName.getText();
                  return inputNameText;
            }

      public String getCalibrationNameText()
            {
                  String inputNameText = calibrationFileName.getText();
                  return inputNameText;
            }

      public String getNumberOfColumnsPerInjectionText()
            {
                  String getNumberOfColumnsPerInjectionText =
numberOfColumns.getText();
                  return getNumberOfColumnsPerInjectionText;
```

```
		}

	private static void addPopup(Component component, final JPopupMenu popup)
		{

		}
}
```

MainClass.java

```
package gckovats;

// Class created by Simon Dellicour (November 2012).

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.util.ArrayList;
import javax.swing.JButton;
import javax.swing.JFrame;

public class MainClass
{
	public static ArrayList<String> finalArray;
	// data coming from the interface :
	public static Interface window;
	public static JButton start;
	public static String inputName;
	public static String calibrationName;
	public static String numberOfColumnsPerInjectionText;
	public static double numberOfColumnsPerInjection;

	// data coming from the inputFileReader :
	public static ArrayList<ArrayList <Double>> originalArray;

	public static void main(String[] args)
		{
			window = new Interface(); // creation of the interface.
			window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
			window.setVisible(true); // necessary line to add to have a nice
window.
			// window.setIconImage(new ImageIcon("Icon.PNG").getImage());
			start = window.getJButton();

			start.addActionListener(new ActionListener()
				{
					public void actionPerformed(ActionEvent e)
						{
							// inputName =
(window.getInputNameText() + ".txt");
							// calibrationName =
(window.getCalibrationNameText() + ".txt");
							inputName = window.getInputNameText();
							calibrationName =
window.getCalibrationNameText();
							numberOfColumnsPerInjectionText =
window.getNumberOfColumnsPerInjectionText();
							numberOfColumnsPerInjection =
Double.valueOf(numberOfColumnsPerInjectionText);

							InputFileReader input = new
InputFileReader();
								try {

	input.readInput(inputName);
							} catch (IOException e3) {
```

```
                                                        // TODO Auto-generated
catch block
                                                        e3.printStackTrace();
                                                }

                                        finalArray = new ArrayList<String>();
                                        finalArray.add(input.individualsList);
                                        finalArray.add(input.origins);
                                                try {

        input.readCalibration(calibrationName);
                                                } catch (IOException e3) {
                                                        // TODO Auto-generated
catch block
                                                        e3.printStackTrace();
                                                }

                                        ArrayList<Double>
calibrationRetentionTimes = new ArrayList<Double>();
                                        for (int i = 0; i <
input.calibrationRetentionTimes.size(); i++)
                                                {

        calibrationRetentionTimes.add(Double.valueOf(input.calibrationRetentionTimes.
get(i)));
                                                }
                                        ArrayList<Double> numberOfCarbons =
new ArrayList<Double>();
                                        for (int i = 0; i <
input.calibrationRetentionTimes.size(); i++)
                                                {

        numberOfCarbons.add(Double.valueOf(input.numberOfCarbons.get(i)));
                                                }

                                        // identify the highest and smallest
retention time :
                                        double maxValue0 = 0; int b0 = -1;
                                        for (int i = 0; i <
input.calibrationRetentionTimes.size(); i++)
                                                {
                                                        if
(calibrationRetentionTimes.get(i).doubleValue() > maxValue0)
                                                                {
                                                                        maxValue0 =
calibrationRetentionTimes.get(i).doubleValue();
                                                                        b0 = i;
                                                                }
                                                }
                                        double minValue0 = maxValue0; int a0 =
-1;
                                        for (int i = 0; i <
input.calibrationRetentionTimes.size(); i++)
                                                {
                                                        if
(calibrationRetentionTimes.get(i).doubleValue() < minValue0)
                                                                {
                                                                        minValue0 =
calibrationRetentionTimes.get(i).doubleValue();
                                                                        a0 = i;
                                                                }
                                                }

                                        for (int i = 0; i <
input.individuals.size(); i++)
                                                {
                                                        ArrayList<Double>
sampleRetentionTimes = new ArrayList<Double>();
```

```
                                                                for (int j = 0; j <
input.individuals.get(i).retensionTimesString.size(); j++)
                                                                {
                                                                        if
(input.individuals.get(i).retensionTimesString.get(j).isEmpty())
                                                                        {
      }
                                                                        else
                                                                        {

      sampleRetentionTimes.add(Double.valueOf(input.individuals.get(i).retensionTim
esString.get(j)));
                                                                        }
                                                                }
                                                        for (int j = 0; j <
sampleRetentionTimes.size(); j++)
                                                                {
                                                                        double
minValue = minValue0;
                                                                        double
maxValue = maxValue0;
                                                                        int a = a0;
// indice of alkane n.
                                                                        int b = b0;
// indice of alkane n.
                                                                        for (int k
= 0; k < calibrationRetentionTimes.size(); k++)
                                                                        {

      if ((sampleRetentionTimes.get(j) > calibrationRetentionTimes.get(k)) &&
(sampleRetentionTimes.get(j)-calibrationRetentionTimes.get(k)) <
(sampleRetentionTimes.get(j)-minValue))

            {

                  minValue = calibrationRetentionTimes.get(k);

                  a = k;

            }

      if ((calibrationRetentionTimes.get(k) > sampleRetentionTimes.get(j)) &&
(calibrationRetentionTimes.get(k)-sampleRetentionTimes.get(j)) < (maxValue-
sampleRetentionTimes.get(j)))

            {

                  maxValue = calibrationRetentionTimes.get(k);

                  b = k;

            }
                                                                        }
                                                                        double I =
-1;

                                                                        //
System.out.println();
                                                                        //
System.out.println("numberOfCarbons.get(a) = " + numberOfCarbons.get(a));
                                                                        //
System.out.println("numberOfCarbons.get(b) = " + numberOfCarbons.get(b));
                                                                        //
System.out.println("sampleRetentionTimes.get(j) = " + sampleRetentionTimes.get(j));
                                                                        //
System.out.println(minValue + " - " + maxValue);
                                                                        I =
100*(numberOfCarbons.get(a)+((numberOfCarbons.get(b)-
```

```
numberOfCarbons.get(a))*((sampleRetentionTimes.get(j)-minValue)/(maxValue-
minValue))));
                                                                        //
System.out.println("I = " + I);
                                                                        // String
IString = new String(); Double IDouble = new Double(I); IString =
IDouble.toString();
                                                                        float
Ifloat = (float) I; String IString = new String(); Float IFloat = new
Float(Ifloat); IString = IFloat.toString();

      input.individuals.get(i).kovatsI.add(IString);
                                                                  }
                                                            }

                                                      String row = new String();
                                                      for (int i = 0; i <
input.maximumNumberOfDataRows; i++)
                                                            {
                                                                  row = new String();
                                                                  for (int j = 0; j <
input.numberOfIndividuals; j++)
                                                                        {
                                                                        if (i <
input.individuals.get(j).kovatsI.size())
                                                                              {

      row = row + input.individuals.get(j).kovatsI.get(i) + "    " +
input.individuals.get(j).columnsWithoutRetentionTimes.get(i) + "  ";
                                                                              }
                                                                        else
                                                                              {

      for (int l = 0; l < (int) numberOfColumnsPerInjection; l++)

            {

                  row = row + "";

            }
                                                                              }
                                                                        }
                                                                  finalArray.add(row);
                                                            }

                                                      // construction of the .txt or .xls
output file :
                                                      OutputFileWriter output1 = new
OutputFileWriter();
                                                      try {

      output1.writeOutput1(finalArray, "GCKovats_output_file.txt");
                                                            } catch (IOException e1) {
                                                                  // TODO Auto-generated
catch block
                                                                  e1.printStackTrace();
                                                            }
                                                }
                                          } // end of "addActionListener".
                              });
      } // end of "main".
}
```

OutputFileWriter.java

```
package gckovats;

//Class created by Simon Dellicour (March 2010).

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;

public class OutputFileWriter
{
	public void writeOutput1(ArrayList<String> string0, String name0) throws
IOException
		{
			ArrayList<String> string = string0;
			String name = name0;

			FileWriter file = new FileWriter(name);
			BufferedWriter buffer = new BufferedWriter(file);
			PrintWriter out = new PrintWriter(buffer);

			for (int i = 0; i < string.size(); i++)
				{
					out.println(string.get(i));
				}

			out.close();
			buffer.close();
			file.close();
		}

	public static void writeOutput2(ArrayList<String> string0, String name0)
throws IOException
		{
			ArrayList<String> string = string0;
			String name = name0;

			FileWriter file = new FileWriter(name);
			BufferedWriter buffer = new BufferedWriter(file);
			PrintWriter out = new PrintWriter(buffer);

			for (int i = 0; i < string.size(); i++)
				{
					out.print(string.get(i) + " ");
				}

			out.close();
			buffer.close();
			file.close();
		}
}
```

References

Bertsch A, Schweer H, Titze A, Tanaka H (2005) Male labial gland secretions and mitochondrial DNA markers support, species status of Bombus cryptarum and B. magnus (Hymenoptera, Apidae). *Insectes Sociaux*, **52**, 45–54.

Bloemberg TG, Gerretzen J, Lunshof A, Wehrens R, Buydens LMC (2013) Warping methods for spectroscopic and chromatographic signal alignment: A tutorial. *Analytica Chimica Acta*, **781**, 14–32.

Dellicour S, Lecocq T (2013) GCALIGNER 1.0: an aligment program to compute a multiple sample comparison data matrix from large eco-chemical datasets obtained by gas chromatography. *Journal of Separation Science*, **36**, 3206–3209.

Hartmann T (2008) The lost origin of chemical ecology in the late 19th century. *Proceedings of the National Academy of Sciences of the United States of America*, **105**, 4541–4546.

Kovats E (1958) Gas-chromatographische Charakterisierung organischer Verbindungen. Teil 1: Retentionsindices aliphatischer Halogenide, Alkohole, Aldehyde und Ketone. *Helv. Chim. Acta*, **41**, 1915–1932.

Lecocq T, Lhomme P, Michez D *et al.* (2011) Molecular and chemical characters to evaluate species status of two cuckoo bumblebees: Bombus barbutellus and Bombus maxillosus (Hymenoptera, Apidae, Bombini). *Systematic Entomology*, **36**, 453–469.

Lecocq T, Vereecken NJ, Michez D *et al.* (2013) Patterns of Genetic and Reproductive Traits Differentiation in Mainland vs. Corsican Populations of Bumblebees. *PLoS ONE*, **8**, e65642.

Lhomme P, Ayasse M, Valterová I, Lecocq T, Rasmont P (2012) Born in an alien nest: how do social parasite male offspring escape from host aggression? *PLoS ONE*, **7**, e43053.

Martin SJ, Helantera H, Drijfhout FP (2008) Evolution of species-specific cuticular hydrocarbon patterns in Formica ants. *Biological Journal of the Linnean Society*, **95**, 131–140.

Medeiros PM, Simoneit BRT (2007) Gas chromatography coupled to mass spectrometry for analyses of organic compounds and biomarkers as tracers for

geological, environmental, and forensic research. *Journal of Separation Science*, **30**, 1516–1536.

Meinwald J, Eisner T (2008) Chemical ecology in retrospect and prospect. *Proceedings of the National Academy of Sciences of the United States of America*, **105**, 4539–4540.

Vereecken NJ, Mant J, Schiestl FP (2007) Population differentiation in female sex pheromone and male preferences in a solitary bee. *Behavioral Ecology and Sociobiology*, **61**, 811–821.

Youngsteadt E, Nojima S, Häberlein C, Schulz S, Schal C (2008) Seed odor mediates an obligate ant-plant mutualism in Amazonian rainforests. *Proceedings of the National Academy of Sciences of the United States of America*, **105**, 4571–4575.

GCALIGNER
GCKOVATS

Simon Dellicour

Thomas Lecocq

GCALIGNER 1.0 and GCKOVATS 1.0 is a software suite to compute a multiple sample comparison data matrix from eco-chemical datasets obtained by gas chromatography without mass spectrometry information.

This manual is a step by step manual for the software suite. The source-code of GCALIGNER and GCKOVATS are presented. A disk with GCALIGNER 1.0 and GCKOVATS 1.0 is included with this manual.

Publisher:
Université de Mons
Faculté des Sciences
Place du Parc, 23
7000 Mons
Belgium

UMONS

Price 10€

ISBN : 978-2-87325-079-9
Dépôt légal : D/2013/970/5