Automating Vibrational Spectroscopy Data Preprocessing and Multivariate Analysis with MATLAB®

by Tanmoy Bhattacharjee

doi: http://dx.doi.org/10.1117/3.2543229
PDF ISBN: 9781510631250
epub ISBN: 9781510631267
mobi ISBN: 9781510631274

Published by

SPIE Press
P.O. Box 10
Bellingham, Washington 98227-0010 USA
Phone: +1 360.676.3290
Fax: +1 360.647.1445
Email: Books@spie.org
Web: http://spie.org

Spotlight vol. SL52
Last updated: 3 September 2019

SPIE.

Table of Contents

SPIE Spotlight Series

Welcome to the SPIE Spotlight series! This growing collection of concise eBooks serves as an entry point for particular topics in optics and photonics suitable for researchers, engineers, managers, executives, and educators. Spotlights fill the community need for timely and relevant references at a level of detail bridging the gap between in-depth journal articles and broad fundamental tutorials. Whatever your interest or need, we hope this series meets your expectations and encourage you to submit your own ideas for future Spotlights online.

Craig Olson, *Series Editor*
L3 Technologies

Associate Editors

Preface

Vibrational spectroscopy, with its sensitivity to biochemical changes and its potential for rapid noninvasive use, is a powerful tool for myriad clinical applications. A tremendous amount of research has been and continues to be reported, supporting existing applications and opening up exciting new avenues. As a result, the amount of data generated has exploded, demanding newer and faster analysis tools. It is no longer tenable to rely on programming experts for each and every problem since it restricts quick testing of ideas and exploring workflows. Knowledge of basic programming can help spectroscopy practitioners save enormous amounts of time spent on data analysis and channel that time toward experimentation. Learning general programming, however, can be time consuming and labor intensive. Therefore, this Spotlight aims to specifically teach only the commands necessary to analyze spectroscopic data (Raman/Fourier transform infrared (FTIR)) using MATLAB®. It explains how to build an analysis routine to apply a step-by-step combination of MATLAB commands and perform preprocessing and multivariate analysis directly from spectra-containing folders with a single click. As an example, an automated script that can import data from several folders, perform first derivatization, select a specific spectral range, perform area normalization and principal component analysis (PCA), plot PCA scores, save principal components, perform linear discriminant analysis (LDA) on PCA results, provide confusion matrix, cross-validate the LDA by the leave-one-out method, and perform predictions using the LDA model, all with a single click, is discussed in detail. A script for a support vector machine is also dealt with briefly. Using these scripts, the reader can build their own script dedicated to the routines used in their laboratory by making minor changes. As an example, modification of the code to automate mean and standard deviation calculations is included. The Spotlight is specifically meant for specialists from backgrounds other than mathematics and programming who wish to automate repetitive analysis and thus avoid technical jargon.

1 Background

Vibrational spectroscopy is a method of probing sample molecular vibrations by subjecting them to light irradiation. Most biomolecules present a unique set of vibrations, which consequently produce an identifiable spectroscopic signature. Thus, the technique can be used to detect and quantify changes in sample biomolecular composition. Apart from specificity, vibrational spectroscopy is also very sensitive and can detect minute changes. Additional attributes, such as the noninvasive and nondestructive nature of analysis and amenability to *in vivo* application designs, render it ideal for use in biology and medicine. Applications of this phenomenon are myriad and widespread. It is beyond the scope of this Spotlight to review all biological/medical applications, and readers can refer to excellent reviews on the subject by Hanlon et al.,[1] Tu and Chang,[2] Pence and Mahadevan-Jansen,[3] Petry et al.,[4] Singh et al.,[5] Cialla-May et al.,[6] Zhao et al.,[7] Ahn et al.,[8] Krafft et al.,[9] and many others. One of the major focus areas is the development of diagnostic and treatment tools for the biomedical industry, which has a market with a 6.4% growth rate. Within this industry, vibrational spectroscopy is especially suited for disease screening, early diagnosis, and disease prevention, and caters to a market with a 7.3% growth rate.[8]

What mainly sets this technique apart from conventional screening/diagnostic methodologies is its ability to provide a complete biochemical fingerprint of the sample. Instead of detecting one or a few disease-associated factors, it gives information on the whole metabolome—that is, the overall change in biomolecules such as proteins, lipids, nucleic acids, carbohydrates, and some other specific molecules. This attribute is of particular advantage in complex diseases such as cancer, where malignancy-specific changes vary greatly. The ability to profile an entire sample's biochemistry also makes this a powerful tool for detecting changes from the normal, potentially signaling disease onset. However, this very feature complicates data analysis to a great extent. Multiple spectral signatures need to be compared across samples and give a single definitive output regarding the sample. In light of extreme in-group variations encountered in biological systems, separating healthy tissue from diseased, especially for early onset, can become very challenging.

Over the years, researchers have developed mathematical tools to tackle this problem, giving rise to the field of chemometrics.[10] A themed collection called "Chemometrics: Tutorials" in advanced data analysis methods produced by the *Analytical Methods* journal can be referred to for more insights on preprocessing[11] and multivariate analysis.[12,13] Gautam et al. have reviewed data processing in detail, specifically for vibrational spectroscopy.[14] The primary aim of the analysis is to weigh the importance of each peak in the spectrum with respect to all other spectra, decide which peaks have maximum variation, and reduce the data to take only the chosen peaks into consideration for the final output. It is clear that the analysis relies heavily on spectral variations, and thus it is very important to

remove all confounding variations from the input spectra before subjecting them to analysis. This operation is referred to as *preprocessing*. Common confounding factors corrected for are sample background, optical errors, sample fluorescence, charge-coupled device (CCD) response, and intensity variations. Optical errors and sample background are corrected by acquiring a spectrum without the sample and subtracting it from the sample spectrum. CCD response variation is offset by dividing the sample spectrum by response from a standardized material (such as a National Institute of Standards and Technology material). This step is important for between-machines comparison and need not be applied if all data originate from the same instrument. Fluorescence background may be removed by baseline correction or by spectrum-first derivatization. Baseline correction involves subtracting a polynomial from the sample spectrum and can be subjective. First derivatization is an objective method and reduces the regions of spectra, where any change in the y-axis values is gradual with respect to the x-axis values to near zero. Since fluorescence signals are broad in nature, fluorescence signals are accordingly reduced to near zero. In the case of sharp vibrational peaks, changes in y-axis values are very rapid with respect to the corresponding x-axis values, leading to nonzero values, isolating signals from background. Finally, intensity-related variations can be removed by normalization. Normalization helps analysis based only on peak variations—presence/absence/shape changes, rather than peak intensity.

The preprocessed spectra are subjected to multivariate analysis. As mentioned earlier, this selects important spectral variations and provides output based only on the chosen signatures. The most common multivariate analysis tool is principal component analysis (PCA). It is an unsupervised tool that does not take group information into consideration. Simply put, it does not matter whether input spectra are supplied with labels "normal," "abnormal," etc., or are unlabeled. PCA calculates the mean of all input spectra, calculates the variation of each spectrum from the average, and then ranks the varying spectral signatures with respect to prevalence.

Consider that wavenumbers 1200, 1450, and 1680 cm^{-1} vary in every input spectra, 1340 and 1745 cm^{-1} vary in 50% input spectra, and 1300 cm^{-1} is variable in only 2% spectra; then PCA ranks the group 1200, 1450, and 1680 cm^{-1} as most important, 1340 and 1745 cm^{-1} as next, and 1300 cm^{-1} as having little significance. The data are transformed to contain only these important wavenumbers and used to separate spectra on the basis of these wavenumbers. The PCA ranks are called principal components (PCs), and the results are presented as graphs of PC scores, where scores are values assigned to each spectrum depending on the extent of variation with respect to the chosen PC. Thus, the plot of PC1 versus PC2 will group spectra with respect to presence/absence/change in wavenumbers 1200, 1450, and 1680 cm^{-1} and 1340 and 1745 cm^{-1}. PCA is usually considered a robust analytical tool, uninfluenced and unbiased. However, it is limited by the number of dimensions that can be plotted and visualized. For example, one cannot

visualize group separation in a plot of PC1 versus PC2 versus PC3 versus PC4, since only three plotting axes are available.

One way to circumvent this problem uses supervised analysis, which requires input in groups with labels. These methods work on principles similar to PCA, but there is a bias toward increasing intergroup separation and decreasing intragroup separation by selecting the best orientation in *n*-dimensions. The results are presented in the form of a confusion matrix; it displays the placement of each individual spectrum in a particular group. The spectrum may be correctly placed in the correct group (the same group as labeled) or in the wrong group. By the unique arrangement of the confusion matrix, all diagonal element spectra are correctly placed, whereas nondiagonal elements are incorrectly grouped. Such discriminant analyses can apply linear, quadratic, partial least square, or other equations to achieve group separations. They are prone to over-fitting, which can lead to erroneous conclusions. To reduce the chances of such errors, the spectra are further subjected to cross-validation. Cross-validation works by dividing the input spectra into training and test groups. The validity of the analysis is then checked based on the power of the subset training spectra group building a model that can correctly predict the test group spectra. The more spectra that are correctly predicted, the better the analysis. A commonly used cross-validation is the leave-one-out (LOO) method, whereby one spectrum is removed and the remaining input spectra are used to train a model, which is then used to predict the group of the one removed spectrum. This is repeated until all spectra have been left out once. The robustness of the model is determined by the number of spectra placed correctly in groups. Finally, for every type of analysis, one can use the built model to predict the group of the spectra whose group is unknown. This process is called *test prediction*.

There are several varied options available for preprocessing and analysis. Every method has its own advantage and disadvantage, and spectroscopists need to adapt the analysis routine that best suits the objectives of their studies. An analysis routine is the group of consecutive steps that are followed to analyze spectroscopy data. This understandably varies from lab to lab as well as experiment to experiment within a lab. There are several types of software available to preprocess spectra and perform multivariate analysis, such as LabSpec, Origin, OPUS, Minitab, Cytospec, and many others. They are excellent tools for the exploration and fixing of a routine. However, once a routine is established, it is time consuming to continue using multiple software for different steps. Programming platforms, such as MATLAB®, R, Python, SciLab, etc., allow scripting that can incorporate all steps and allow an analysis with a single click. Features such as turning on/off certain steps or multivariate analysis can be designed without forgoing the single-click option. Ultimately, the scripts may prove to be quickly exploratory as well as routine analysis tools, with the advantages of rapidity, accuracy, and ease of use. The Spotlight details the simple commands that can achieve this with instructions for using them to yield user-specific codes.

2 Overview

The tutorial is structured according to the steps of preprocessing and multivariate analysis. We will, therefore, start with a discussion on how to import a spectrum into MATLAB, followed by how to derivatize this spectrum, select a specific spectral range from the derivative spectrum, and normalize the derivative spectrum of the selected range in Section 4. Section 5 will demonstrate how to automate the preprocessing by scripting to import multiple spectra and applying preprocessing commands to them. Section 6 will extend this concept to import multiple spectra from multiple subfolders and preprocess them all. Section 7 will provide details on applying PCA, PC-LDA, and leave-one-out cross validation (LOOCV) on the preprocessed spectra with a note on support vector machines (SVMs). Section 8 teaches scripting of test prediction code. Sections 9 and 10 will discuss automating PCA plotting and script fine-tuning to build in options to turn features on and off. Common errors encountered while executing the code and an example of how to adapt the code to automate other processes such as mean and standard deviation calculations are included at the end.

3 MATLAB Desktop

Extensive documentation on the installation and use of MATLAB can be found using MATLAB help or by using any search engine. Figure 1 shows the editor,

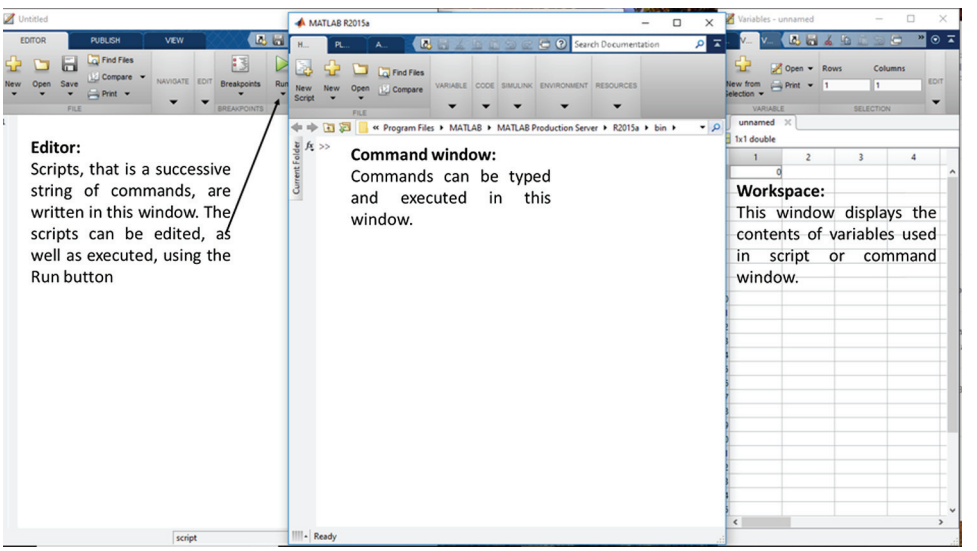

Figure 1 Desktop view of MATLAB. The command window is where commands are typed, while the variable window or workspace is where variables used can be seen. Editor is the place where a set of commands can be written, edited, and executed.

command window, and workspace. The initial sections will use the command window to type the commands and produce the output. Later on, as we master the commands, we will use them to make an automated script, wherein we will be using the editor. In all of the sections, a set of commands or scripts to be pasted in the editor and executed are enclosed in a box.

4 Note on Array Indexing and the "Loop" Function

Before we dive into the particulars of commands required for spectroscopy data analysis, it will be helpful to understand two recurring concepts. The first is *array indexing*. An array is a set of numbers arranged in a tabular form. Each number is called an element, and the position of the element is called the index. Several of the later commands will involve identifying the index, changing, isolating, or cutting rows and columns, or deleting certain parts of the array. The other is the "loop" function. As the name suggests, it is applied to repeat a set of commands for a particular number of times. This function is critical, as we aim to automate a repetitive process.

4.1 Array indexing

Let us begin with an example of a simple array:

1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19
5	10	15	20

We can assign them row and column numbers as follows:

	Column 1	Column 2	Column 3	Column 4
Row 1	1	6	11	16
Row 2	2	7	12	17
Row 3	3	8	13	18
Row 4	4	9	14	19
Row 5	5	10	15	20

Now, we will assign a position to each element in brackets beside them.

	Column 1	Column 2	Column 3	Column 4
Row 1	1 (1, 1)	6 (1, 2)	11 (1, 3)	16 (1, 4)
Row 2	2 (2, 1)	7 (2, 2)	12 (2, 3)	17 (2, 4)
Row 3	3 (3, 1)	8 (3, 2)	13 (3, 3)	18 (3, 4)
Row 4	4 (4, 1)	9 (4, 2)	14 (4, 3)	19 (4, 4)
Row 5	5 (5, 1)	10 (5, 2)	15 (5, 3)	20 (5, 4)

Thus, every element has a unique set of numbers to identify its position, called the index. The index of "1" is "(1, 1)," whereas the index of "20" is "(5, 4)."

To create an array in MATLAB, we can simply type it out, row by row, and press enter to start a new row within square brackets. Let us assign the array to a variable "*a*" (any alphabet or word can serve as a variable, it is up to the user).

```
>> a =

    [1      6      11      16
     2      7      12      17
     3      8      13      18
     4      9      14      19
     5      10     15      20]
```

a =

```
    1      6      11      16
    2      7      12      17
    3      8      13      18
    4      9      14      19
    5      10     15      20
```

Note that the input is shown in bold after the symbol ">>" in "Courier New" font, and the output by MATLAB is in italics.

To find out the number at a particular position in an array, we can simply type the name of the array (row, column). For example, if we want to find the number in the first column of the first row of array "*a*," we can type

```
>> a (1,1)
ans = 1
```

Similarly, to find the number in the second column of the fifth row, we can use

```
>> a (5,2)
ans = 10
```

To find out all elements in the first row, we use the name of the array (row, :). The colon symbol translates to "all" in this case. Thus, MATLAB understands the

same command as the name of the array (row, all). To find all of the elements in row 1, we type

```
>> a (1, : )
ans = 1     6     11     16
```

Similarly, to get all elements in a column, e.g., column 2, we type

```
>> a ( : , 2)
ans =
      6
      7
      8
      9
      10
```

To find all elements in more than one row, we can simply provide the range of rows—name of the array (from row *x* to row *y*, :), where *x* and *y* are any number. For example, to find all elements in the first three rows of the array "a," we type

```
>> a (1:3, : )
ans =
      1       6       11       16
      2       7       12       17
      3       8       13       18
```

Here, the colon translates to "to." For example, if we want MATLAB to list all numbers from one to five, we can type

```
>> 1:5
ans =
      1
      2
      3
      4
      5
```

If we want MATLAB to display the numbers at particular intervals, the command is—from number *x*: interval: to number *y*, where *x* and *y* can be any number. So, if we want MATLAB to show every second number between 1 and 5, we can type

```
>> 1:2:5
ans =
      1
      3
      5
```

Finally, we may need MATLAB to find numbers that are neither complete rows nor columns. In such cases, we can simply give the range for both rows

and columns for which we need the elements displayed —name of the array (row x to row y, column v to column u), where x, y, u, and v can be any number. For example, if we want all elements that belong to rows two to three and columns one to three, we can type

```
>> a (2:3, 1:3)
ans =
     6     11
     7     12
     8     13
```

4.2 "Loop" function

The structure of a loop command is

```
for (number of repeats)
            commands to be repeated
end
```

For example, if we want a variable, say, "*b*," to change from one to five, we can type

```
>> for i = 1:5;
       b = i
   end
b =  1
b =  2
b =  3
b =  4
b =  5
```

Note that the semicolon after "1.5" stops the result from being displayed. If we do not include the semicolon, we will get the outcome of typing $i = 1{:}5$ in MATLAB:

```
>> for i = 1:5
       a = i
   end
i =  1
i =  2
i =  3
i =  4
i =  5

b =  1
b =  2
b =  3
b =  4
b =  5
```

5 Basic Preprocessing Operations

5.1 Importing a spectrum

We will begin with the first step—importing a spectrum (Fig. 2). The command for importing a spectrum file in extensions "txt" or "asc" is dlmread (filename). To import a file with name "alpha spectra (1)," we need to type

```
>> dlmread('alpha spectra (1).txt')
ans =
```

4000	*0.055765*
3998	*0.055832*
3996	*0.056107*
...	...
900	*−0.06471*

Certain spectra files contain text along with wavenumber/intensity information. An example of such a file is shown in Fig. 3. In such cases, it is vital to remove these text lines while importing into MATLAB. To achieve this, we can extend the command—dlmread (filename," number of lines to skip, 0). For example, in the file in Fig. 3 named "alpha spectra (1)," there are a total of 56 lines of text/spaces before the wavenumber and intensity values start. Thus, 56 lines need to be skipped. For this, we can type

```
>>dlmread('alpha spectra (1).txt',",56,0)
ans =
```

4000	*0.055765*
3998	*0.055832*
3996	*0.056107*
...	...
900	*−0.06471*

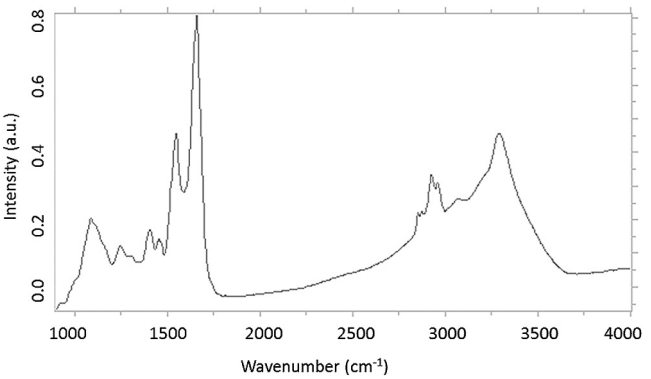

Figure 2 An example of FTIR spectrum.

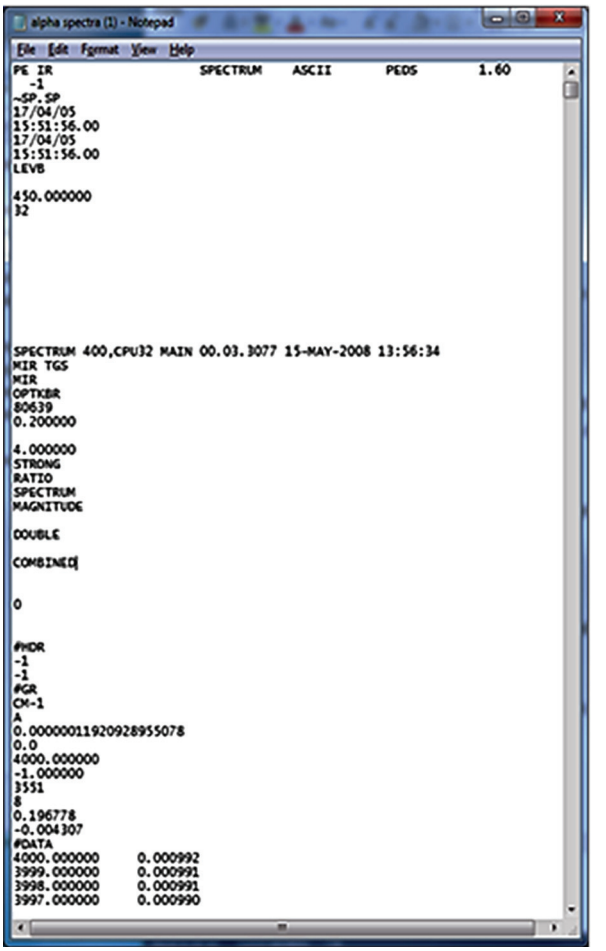

Figure 3 A spectral file containing 56 lines of text.

The spectral information is stored in the variable "ans." It is more convenient to define a variable and put the information in that variable by defining the name of the variable = dlmread (filename). For example, we can store the spectral information in "alpha spectra (1).txt" in the variable "`data`" by typing

```
>> data=dlmread('alpha spectra (1).txt')
```

data =

4000	*0.055765*
3998	*0.055832*
3996	*0.056107*
...	...
900	*−0.06471*

For files with extension "csv," the command will change to csvread (filename, number of lines to skip, 0). For example, for a file named "H2.csv," where we need not skip any lines, the import command will be `csvread ("H2.csv," 0, 0)`.

5.2 Separate wavenumbers from intensity

Every spectrum is a list of wavenumbers and their corresponding intensities. However, only intensity values are subjected to preprocessing and not the wavenumbers. Therefore, before proceeding to preprocessing, the "intensity" column (column 2) needs to be separated from the "wavenumber" column (column 1). We will store the wavenumbers in variable "`datax`," since wavenumbers form the *x* axis of the spectrum. The command for this is

```
>> datax=data(:,1)
datax =
        4000
        3999
        3998
        .....
        900
```

We will store the "intensity" column in variable "`datay`" with the command

```
>> datay=data(:,2)
datay =
        0.055765
        0.055832
        0.056107
          ...
        −0.06471
```

Script: Import single spectrum, separate wavenumber and intensity:

```
data=dlmread('alpha spectra (1).txt');
datax=data(:,1);
datay=data(:,2);
```

5.3 Perform the first derivation

We will obtain first-order derivatives of intensity values in "`datay`" and store the results in another variable called "`fd`." To perform this, we will type

```
>> fd=diff(datay)./diff(datax)
fd =
      −3.3E-05
      −0.00014
      8.65E-05
        ...
      0.00072
```

Since the operation involves the subtraction of every number from its consecutive number, the final array has one row less than the original array. Hence, the wavenumber containing variable "datax" also needs to be reduced by one row. This is done by typing

```
>> dataxd=datax(1:length(fd))
dataxd =
        4000
        3998
        3996
         ...
        902
```

The command—length, as the name suggests—finds the number of rows in a particular array. We can take the example of array "a." We type the following to first get the array:

```
>> a
a =
     1     6    11    16
     2     7    12    17
     3     8    13    18
     4     9    14    19
     5    10    15    20
```

Then, we type

```
>> length(a)
ans = 5
```

To find the length of the variable "datay," we type

```
>> length(datay)
ans = 1552
```

Then, we find length of fd

```
>> length(fd)
ans = 1551
```

Thus, when we type dataxd = datax (1:length (fd)), it means dataxd = datax (1: 1551). Thus, all elements from row 1 to row 1551 will be stored in "dataxd," which is one less than the original length.

Some spectroscopists may like to explore the use of the second-order derivative, instead of the first order. In such a case, the command will simply change to diff (datay, 2)./ diff (datax, 2).

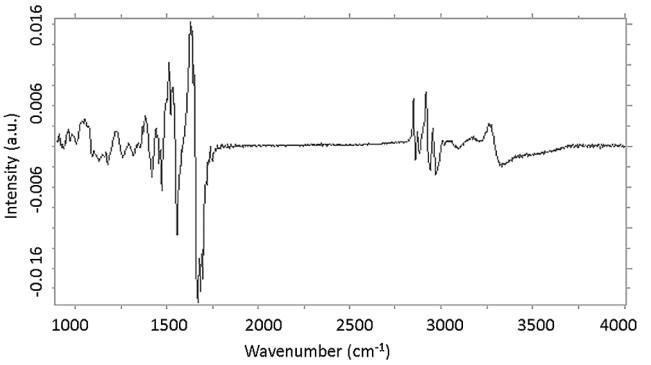

Figure 4 Spectrum after applying the first derivative.

Script: Import single spectrum, first derivative (Fig. 4):

```
data=dlmread('alpha spectra (1).txt');
datax=data(:,1);
datay=data(:,2);
fd=diff(datay)./diff(datax); dataxd=datax(1:length(fd));
```

5.4 Select a specific spectral range

Some regions of spectra are rich in peaks (400 to 1800 cm^{-1} and 2800 to 4000 cm^{-1}), while other regions lack spectral signatures (1800 to 2800 cm^{-1}). Spectroscopists tend to use the signature-rich regions for analysis. To choose these regions in MATLAB, we can use the operations to extract particular row and column elements from an array. We have already stored the wavenumbers of the spectrum from "data" after applying the first derivative in "`dataxd`":

```
>> dataxd
dataxd =
```

4000	→	*Row 1*
3998	→	*Row 2*
3996	→	*Row 3*
1800	→	*Row 1101*
1798	→	*Row 1102*
1202	→	*Row 1400*
1200	→	*Row 1401*
902	→	*Row 1551*

Let us say we wish to extract 1200 to 1800 cm^{-1} from this spectrum. In that case, we need to find the index for elements 1200 and 1800. To find the index of 1200 and store it in a variable "`p1`," we type

```
>> p1=find(dataxd==1200)
p1 = 1401
```

To store the index of 1800 in variable "`p2`," we type

```
>> p2=find(dataxd==1800)
p2 = 1101
```

We find out if p2 > p1 or p1 > p2 by

```
>> p3=p1-p2
p3 = 300
```

Since "`p3`" is positive, p1 > p2. Therefore, while extracting, we have to start from p2 and end with p1.

Then, we perform the element-selecting operation on the array "`fd`," which has the first derivative of the spectrum and store it in variable "`yInterpolate`." Note that the values are different from "`data`" because we are using the results after applying the first-order derivative on "`data`":

```
>> yInterpolate=fd(p2:p1)
yInterpolate =
              0.000202
              0.00013
             -7.6E-05
                 ...
             -7.9E-05
```

Since the operation decreases the number of rows in the "intensity" column, the number of rows need to correspondingly decrease in the "wavenumber" column. We can use the same operation on "`dataxd`" and store the result in variable "`xInterpolate`":

```
>> xInterpolate=dataxd(p2:p1)
xInterpolate =
              1800
              1798
              1796
               ...
              1200
```

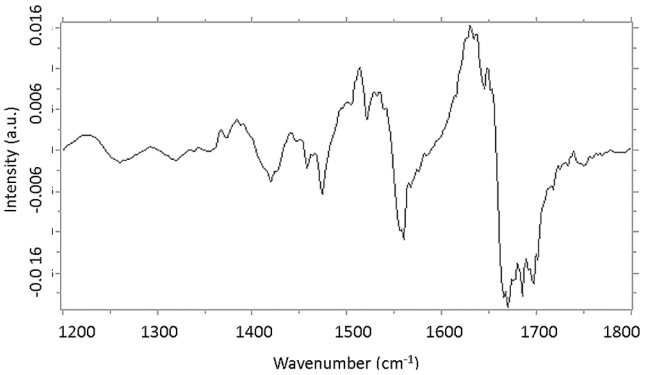

Figure 5 First derivatized spectrum after selecting the spectral range 1200 to 1800 cm^{-1}.

Script: Import single spectrum, first derivative, **selecting specific spectral range** (Fig. 5):

```
data=dlmread('alpha spectra (1).txt');
datax=data(:,1);
datay=data(:,2);
fd=diff(datay)./diff(datax);
dataxd=datax(1:length(fd));
p1=find(dataxd==1200);
p2=find(dataxd==1800);
p3=p1-p2;
yInterpolate=fd(p2:p1);
xInterpolate=dataxd(p2:p1);
```

5.5 Area normalization

As mentioned earlier, normalization is used to remove intensity variation. There are different types of normalization—specific peak normalization, highest peak normalization, area normalization, and so on. In this tutorial, we will use area normalization. We will use the output from the previous step. We have the wavenumber values stored in "`xInterpolate`" and derivatized values in "`yInterpolate`." First, we will calculate the area under the curve and store the area in variable "`Area`" using

```
>> Area=trapz(xInterpolate,yInterpolate)
```

Area = 0.0949

In this tutorial, the idea is to normalize the area under the curve of every spectrum to 100. Therefore, we will divide the area by 100:

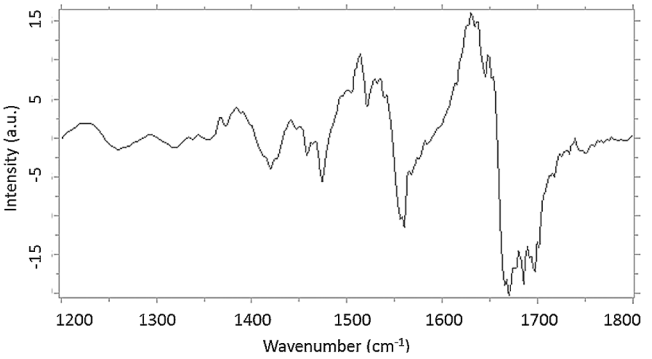

Figure 6 First derivatized spectrum in the selected spectral range after area normalization.

```
>> AreaS= Area/100
AreaS = 0.000949
```

We will then divide every intensity value by the "`AreaS`" and store it in variable "`norm`" by

```
>> norm=yInterpolate/AreaS
norm =
          0.212758
          0.136924
         −0.08005
             ...
          0.08268
```

Script: Import single spectrum, first derivative, **select spectral range**, and area normalization (Fig. 6):

```
data=dlmread('alpha spectra (1).txt');
datax=data(:,1);
datay=data(:,2);
fd=diff(datay)./diff(datax);
dataxd=datax(1:length(fd));
p1=find(dataxd==1200);
p2=find(dataxd==1800);
p3=p1-p2;
yInterpolate=fd(p2:p1);
xInterpolate=dataxd(p2:p1);
Area=trapz(xInterpolate,yInterpolate);
AreaS= Area/100;
norm=yInterpolate/AreaS;
```

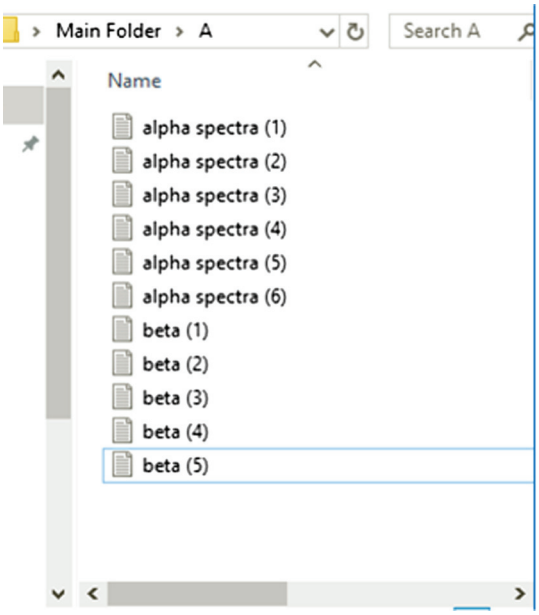

Figure 7 An example of a folder containing spectra.

6 Automating Preprocessing for Multiple Spectral Files

In this section, we will learn to automate the preprocessing for multiple spectral files in a folder. Let us consider a folder containing 11 spectra files (Fig. 7).

To begin, we must first make this folder the current directory in MATLAB. This can be done by typing (Fig. 8)

```
>> cd('C:\Users\Tanmoy\Desktop\Main Folder\A')
```

We can then store the files with extension "txt" in a variable "d" by

```
>> d = dir ('*.txt')
```

We can find out the number of files using MATLAB by

```
>> length (d)
ans = 11
```

To import data from these files and subsequently preprocess them, we need the filenames stored in "d." The first filename can be retrieved by (Fig. 9)

```
>> d(1).name
ans = alpha spectra (1).txt
```

The information in the file can then be imported using `dlmread`.

Similarly, the second filename can be retrieved by

```
>> d(2).name
ans = alpha spectra (2).txt
```

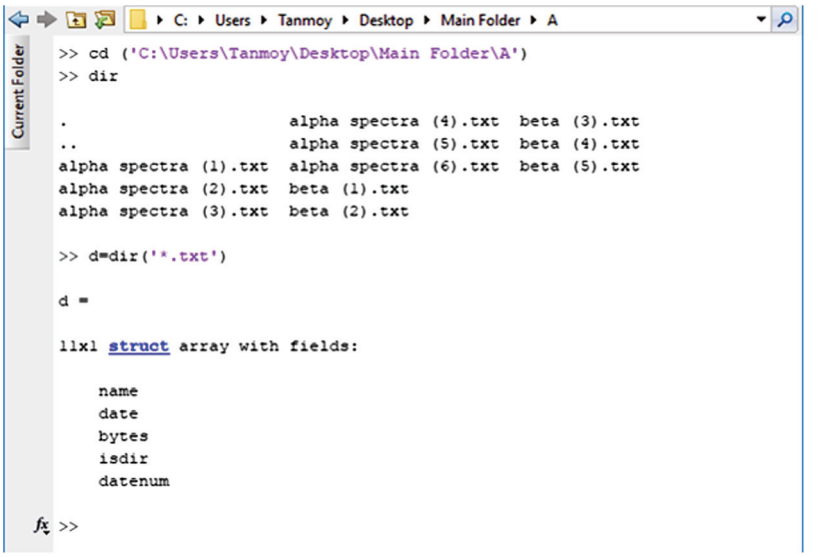

Figure 8 Output in MATLAB for commands `cd`, and `dir`. `cd` changes the directory to the one mentioned in brackets. dir displays all the files in the folders.

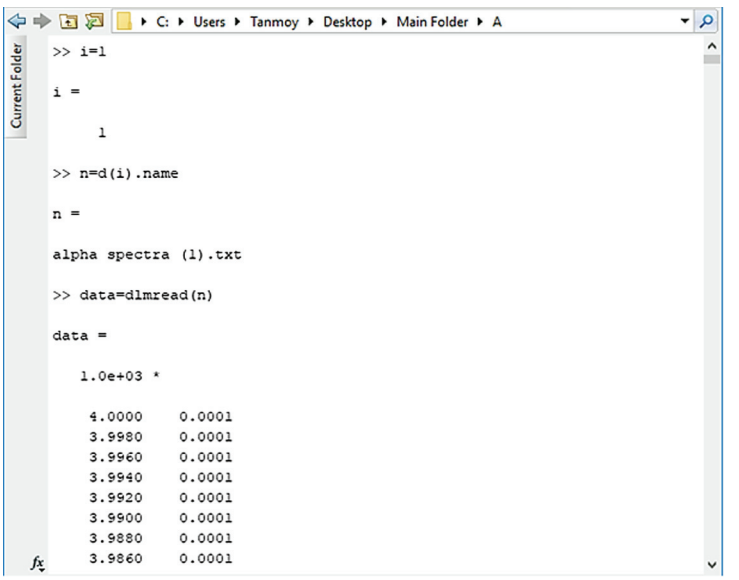

Figure 9 MATLAB commands to identify the filename and import the spectrum in the file.

We can automate the retrieval of names such that all names in the folder are taken up into a variable "n," one at a time, by using the "for" loop as follows (Fig. 9):

```
>> for i = 1:length(d);
       n = d(i).name
   end
```

```
n =    alpha spectra (1).txt
n =    alpha spectra (2).txt
n =    alpha spectra (3).txt
n =    alpha spectra (4).txt
n =    alpha spectra (5).txt
n =    alpha spectra (6).txt
n =    beta (1).txt
n =    beta (2).txt
n =    beta (3).txt
n =    beta (4).txt
n =    beta (5).txt
```

We can import the data by

```
>> for i=1:length(d)
       n=d(i).name
       data=dlmread(n);
   end
```

The preprocessing commands can be added after "data = dlmread (n)" and before "end," and all data will be imported and preprocessed.

Script: Preprocess all spectra in a folder (Fig. 10)

```
cd('C:\Users\Tanmoy\Desktop\Main Folder\A')
d=dir('*.txt')
for i=1:length(d)
n=d(i).name
data=dlmread(n); datax=data(:,1); datay=data(:,2);
fd=diff(datay)./diff(datax); dataxd=datax(1:length(fd));
p1=find(dataxd==1200); p2=find(dataxd==1800);
yInterpolate=fd(p2:p1,:);
xInterpolate=dataxd(p2:p1,:);
Area=trapz(xInterpolate,yInterpolate);
AreaS=Area/100;
norm=yInterpolate/AreaS;
end
```

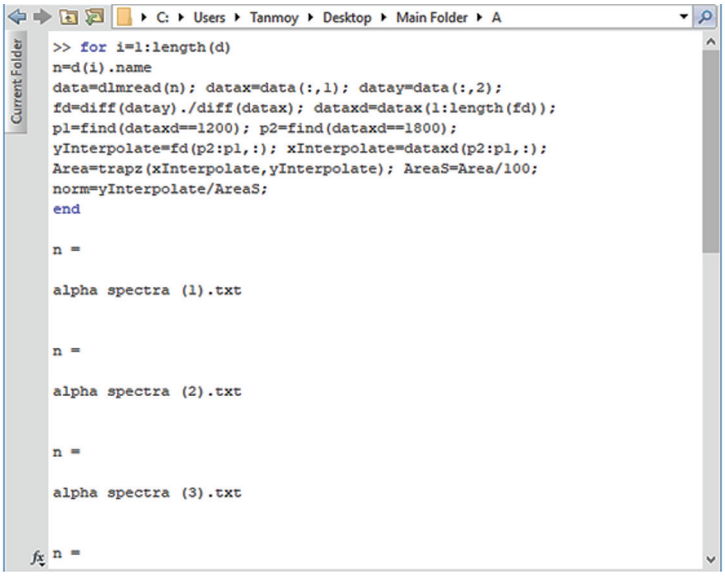

Figure 10 Demonstration of script that automates data import. From the output of "`n`," it can be seen that all filenames are identified one after the other.

One important aspect that remains is saving the preprocessed spectra. In the above script, all variables—"`i`," "`n`," "`data`," "`fd`," "`yInterpolate`," "`norm`," and others—change during every cycle of the "`for`" loop. This is critical, as it allows the same operations to be applied to data imported from each spectrum. However, nothing gets stored! We must make provisions within the code to save every final preprocessed spectrum. We can achieve this by declaring an empty variable, say, "`s`," and adding each preprocessed spectrum as a column in the same. To declare an empty variable, we can type

```
>> s = []
s = []
```

The variable has to be declared outside the "`for`" loop.

To add preprocessed spectrum intensity values as a column, we type

```
>> s = [s norm];
```

This needs to be added inside the "for" loop, as follows:

```
>> s = [];
      for i = 1:length (d);
          n = d(i).name; . . . ; norm=yInterpolate/AreaS;
          s=[s norm];
      end
```

In order to understand how it works, let us take an example.

After declaring s = [], it looks as follows

s =

[ ]

Now, we initiate the "for" loop. In the first round, "`i = 1,`" so "`n = d(1).name = 'alpha spectra (1).txt',`" which leads to "`data = dlmread('alpha spectra (1).txt')`." The "`data`" undergo preprocessing and result in "`norm.`" Let

norm =

0.212758
0.136924
−0.08005
...

Since `s = [s norm]`, and previously `s = []`, `s` will be

s =

$$[] + \begin{matrix} 0.212758 \\ 0.136924 \\ -0.08005 \\ \cdots \end{matrix} = \begin{matrix} 0.212758 \\ 0.136924 \\ -0.08005 \\ \cdots \end{matrix}$$

In the second round, "`i = 2,`" "`n = 'alpha spectra (2)'`" and they will be preprocessed to get a new "`norm`." Let it be

norm =

0.135933
0.358573
0.113558
...

So, s will become

s =

$$\begin{matrix} 0.212758 \\ 0.136924 \\ -0.08005 \\ \cdots \end{matrix} + \begin{matrix} 0.135933 \\ 0.358573 \\ 0.113558 \\ \cdots \end{matrix} = \begin{matrix} 0.212758 & 0.135933 \\ 0.136924 & 0.358573 \\ -0.08005 & 0.113558 \\ \cdots & \cdots \end{matrix}$$

After 11 rounds of the "`for`" loop, s will have 11 columns, each representing one preprocessed spectrum (Figs. 11–13).

This is not an efficient method of storing new spectra, because it may cause "out of memory" errors. Preallocating memory to s can prevent the problem and can be achieved by specifying s = zeros (total number of rows, total number of columns).

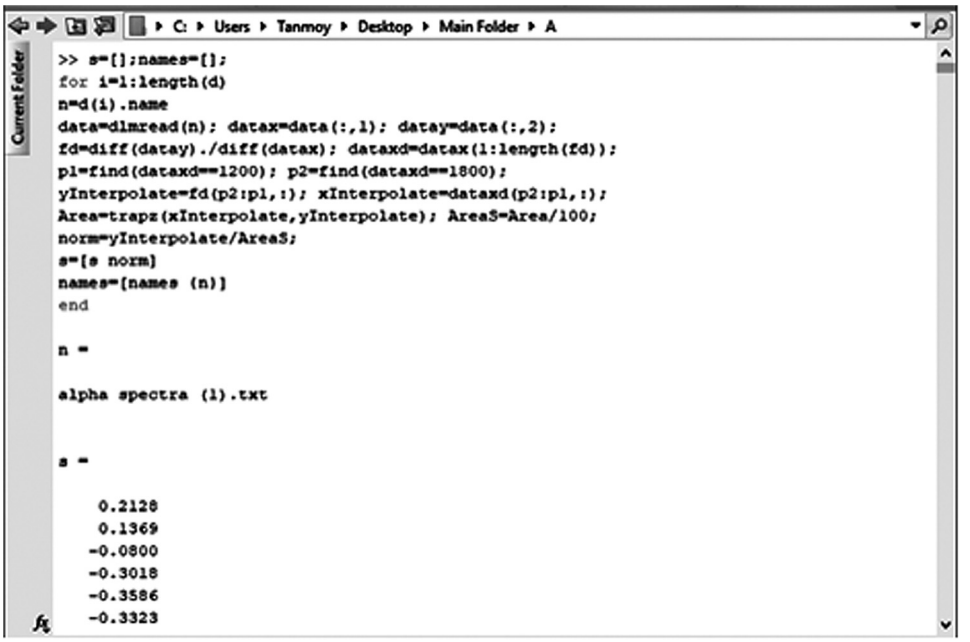

Figure 11 Demonstration of a script's ability to automatically save preprocessed spectra. In the first round, "alpha spectra (1).txt" is identified, and the preprocessed spectrum is stored in the first column of s.

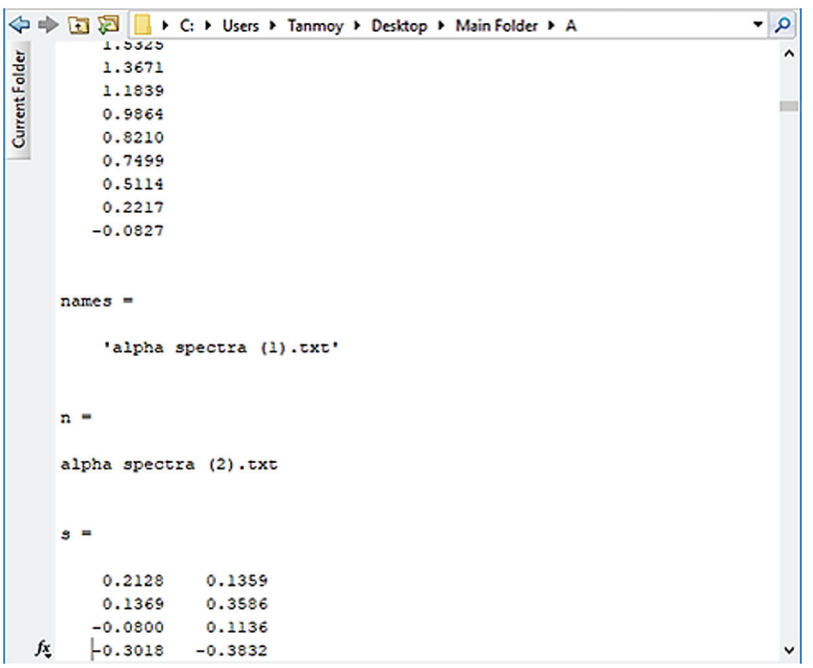

Figure 12 Demonstration of a script's ability to automatically save preprocessed spectra. The second spectrum filename is identified, and the preprocessed spectrum is stored in the second column of s.

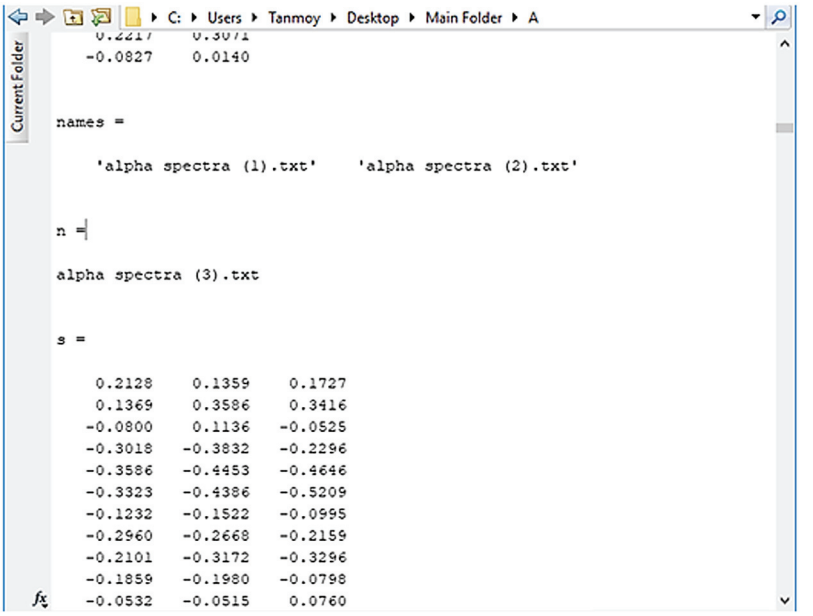

Figure 13 Demonstration of a script's ability to automatically save preprocessed spectra. The third spectrum filename is identified, and preprocessed spectrum is stored in the third column of s.

The same method can be applied to store filenames.

```
>> names = [];
>> for i =1:length (d); ..... names =[names {n}];
   ....... end
```

Script: Preprocess all spectra in a folder and save

```
s=[];names=[];
cd('C:\Users\Tanmoy\Desktop\Main Folder\A')
d=dir('*.txt')
for i=1:length(d)
n=d(i).name
data=dlmread(n); datax=data(:,1); datay=data(:,2);
fd=diff(datay)./diff(datax); dataxd=datax(1:length(fd));
p1=find(dataxd==1200); p2=find(dataxd==1800);
yInterpolate=fd(p2:p1,:); xInterpolate=dataxd(p2:p1,:);
Area=trapz(xInterpolate,yInterpolate);
AreaS=Area/100;
norm=yInterpolate/AreaS;
s=[s norm]
names=[names {n}]
end
```

7 Automating Preprocessing for Multiple Files Contained in Multiple Subfolders

Supervised multivariate analysis requires group information. Even unsupervised PCA is visualized by applying different colors to different groups, thus requiring group information. It is, therefore, common to make one subfolder for each group. It would make data analysis extremely convenient if MATLAB could open subfolders, import spectra, assign them a group name, go to the next subfolder, and so on, and provide the information on which spectrum belongs to which group automatically to multivariate analysis script. This section describes ways to achieve this.

Let us consider the following example. Two subfolders are created inside a main folder. The subfolder named "A" contains all spectra from group A, and subfolder "B" all spectra from group B. Group A has 11 spectra, whereas group B has 13 spectra (Fig. 14).

We will first make the main folder, the current MATLAB directory (Fig. 15):

```
>> cd('C:\Users\Tanmoy\Desktop\Main Folder')
```

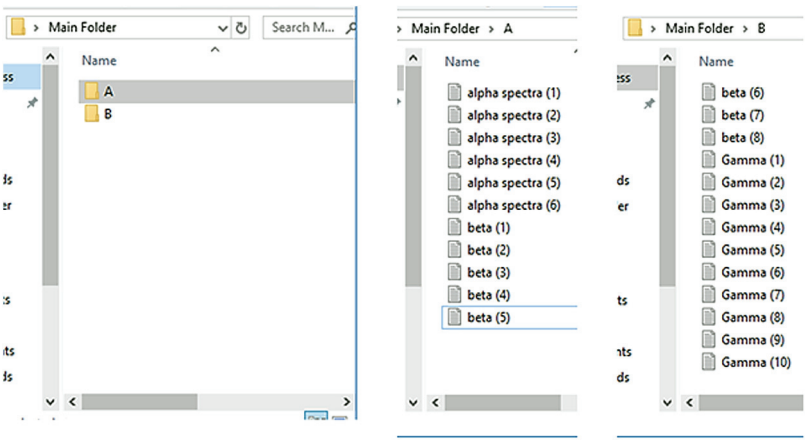

Figure 14 Example of folder and subfolder organization. The main folder contains two subfolders A and B, which contain 11 and 13 spectra, respectively.

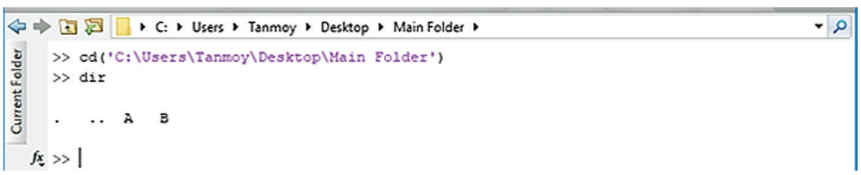

Figure 15 Output of MATLAB command `dir` showing that the main folder has four sub-folders: "." ".." "A," and "B."

Then, we will store the subfolder names in a variable "d1":

```
>> d1=dir
```

Using the "for" loop, we will retrieve the names of each subfolder:

```
>> for ii = 3:length(d1);
      n1 = d1(ii).name
   end
```

```
n1=          A
n1 =         B
```

Note that we are using `ii = 3: length (d1)`, and not `ii = 1:length (d1)`. This is due to two default subfolders "." and " ..".

Since we have retrieved the names of the subfolders, we can apply the commands to read spectra files from the folders and preprocess them as before. It is important to remember to put the command `cd ('C:\Users\Tanmoy\Documents\Folder')` in the loop to enable MATLAB to return to the main folder after processing each folder:

```
for ii = 3:length (d1);
     n1 = d1(ii).name;
     ....
     for i = 1:length(d);
          ......
          group_names=[group_names {n1}];
     end
      nome = [nome {n1}];
      cd ('C:\Users\Tanmoy\Documents\Main Folder');
end
```

We can use variables "nome" and "group_names" to save the names of subfolders in the same way we stored filenames. "nome" simply contains the names of folders A and B, while "group_names" contain the group name "A" or "B" for each file. In this case, there are 11 "A" group spectra and 13 "B" group spectra, so "group_names" will have 11 "A" and 13 "B." This becomes important during multivariate analysis LDA, when "group_names" is provided as the input to indicate which spectrum belongs to which group (Figs. 16 and 17).

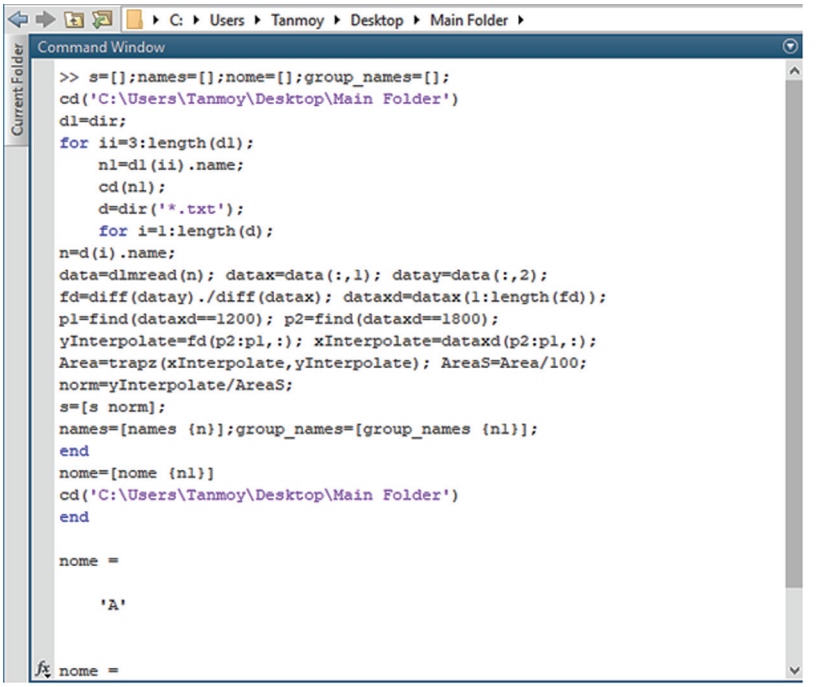

Figure 16 Demonstration of a script storing subfolder names in "nome."

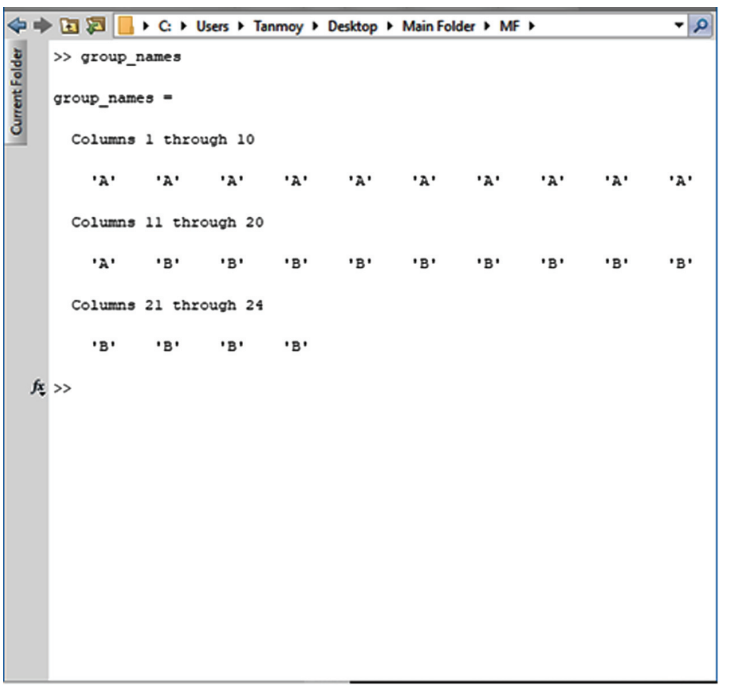

Figure 17 The group name for each spectrum stored individually in the variable "`group_names`."

Script: Preprocess all spectra in every subfolder in a folder

```
clear;clc;
s=[];names=[];nome=[];group_names=[];
cd('C:\Users\Tanmoy\Desktop\Main Folder')
d1=dir;
for ii=3:length(d1);
    n1=d1(ii).name;
    cd(n1);
    d=dir('*.txt');
    for i=1:length(d);
n=d(i).name;
data=dlmread(n); datax=data(:,1); datay=data(:,2);
fd=diff(datay)./diff(datax); dataxd=datax(1:length(fd));
p1=find(dataxd==1200); p2=find(dataxd==1800);
yInterpolate=fd(p2:p1,:);
xInterpolate=dataxd(p2:p1,:);
Area=trapz(xInterpolate,yInterpolate);
AreaS=Area/100;
norm=yInterpolate/AreaS;
s=[s norm];
names=[names {n}];group_names=[group_names {n1}];
end
nome=[nome {n1}]
cd('C:\Users\Tanmoy\Desktop\Main Folder')
end
```

8 Performing Multivariate Analysis

8.1 Principal component analysis

The script we have built until now preprocesses the "intensity" values of each spectrum one by one from all subfolders and gathers them in one variable "s":

```
>> s
```

s =

spectrum 1	spectrum 2	⟶	spectrum 24
0.212758	*0.135933*	…	−0.06265
0.136924	*0.358573*	…	0.257174
−0.08005	*0.113558*	…	0.16409
…	…	…	…

For PCA, we need to transpose this array, such that "observations," that is, spectra, are in the rows and "variables," which are intensity values corresponding to particular wavenumbers, are in the columns. The transpose operation can be performed simply by

```
>> s'
```

ans =

spectrum 1	0.212758	0.136924	−0.08005	...
spectrum 2	0.135933	0.358573	0.113558	...
↓	...	...	...	...
spectrum 24	−0.06265	0.257174	0.16409	...

Using this transpose function, PCA can be performed by the command

```
>> [pc,score,latent]=princomp(s')
```

pc =

−0.00385	*−0.00337*	...
−0.00045	*0.003497*	...
−0.00066	*−0.00035*	...
...	...	...

score =

26.89575	*4.828957*	...
29.06787	*5.086694*	...
27.42997	*5.222958*	...
...	...	...

latent =

19955.98
174.8768
13.66114
...

Note that "`princomp`" is an old expression, used in earlier MATLAB versions. The new expression is "`pca.`" So, "`[pc,score,latent] = princomp(s')`" can be replaced by "`[pc,score,latent]=pca(s')`".

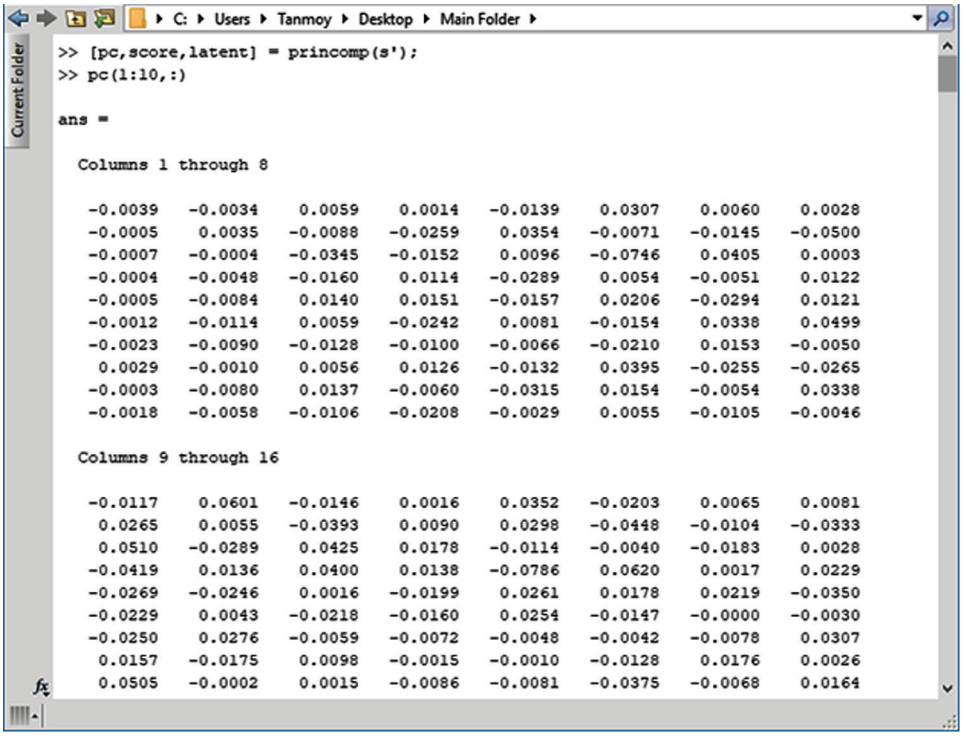

Figure 18 First 10 PCs after applying the `princomp` function in MATLAB.

The variable "`pc`" (Fig. 18) has 301 rows and 301 columns, "`score`" has 24 rows and 301 columns, and "`latent`" has 301 rows. The "`pc`" variable contains all of the PCs, with each column representing a PC. Thus, column 1 is PC1, column 2 is PC2, and so on. There are 301 PCs because the number of wavenumbers used as input is 301, from 1200 to 1800 cm^{-1}. "`score`" contains a value for each sample for every PC. This value reflects the position of that sample for that PC with respect to others. Samples with values close to each other suggest similarity in spectra. The 24 rows in "`score`" are due to 24 samples used as input, and 301 columns correspond to 301 PCs. The variable "`latent`" provides information on how much variation within the input samples are covered by that particular PC. So, if the first column in "latent" after calculating the percentage is 90%, it means that 90% of the spectral variation in the input sample has been included in PC1. This can be conveniently observed using a scree plot. To obtain this plot, we first divide every row of "`latent`" by the sum of all rows in "`latent`" and save it in "`contribution`." Since we want the plot to start at "0," we add a row containing "0" to the contribution array on the top in "`forplot`." We then plot "`forplot`" multiplied by 100 to get a percentage

(Fig. 19). Since most variation is generally covered in the first 10 PCs, we save the contribution of the first 10 PCs in "PC_sig":

```
>> contribution = cumsum (latent)./sum(latent);
    forplot=[0
    contribution];
    plot(0:length(contribution),forplot*100);
    PC_sig= contribution (1:10,:);
```

Script: Preprocess + PCA

```
clear;clc;
s=[];names=[];nome=[];group_names=[];
cd('C:\Users\Tanmoy\Desktop\Main Folder')
d1=dir;
for ii=3:length(d1);
    n1=d1(ii).name;
    cd(n1);
    d=dir('*.txt');
    for i=1:length(d);
n=d(i).name;
data=dlmread(n); datax=data(:,1); datay=data(:,2);
fd=diff(datay)./diff(datax); dataxd=datax(1:length(fd));
p1=find(dataxd==1200); p2=find(dataxd==1800);
yInterpolate=fd(p2:p1,:);
xInterpolate=dataxd(p2:p1,:);
Area=trapz(xInterpolate,yInterpolate);
AreaS=Area/100;
norm=yInterpolate/AreaS;
s=[s norm];
names=[names {n}];group_names=[group_names {n1}];
end
nome=[nome {n1}]
cd('C:\Users\Tanmoy\Desktop\Main Folder')
end
[pc,score,latent]=princomp(s');
contribution=cumsum (latent)./sum(latent);
PC_sig= contribution (1:10,:);
forplot=[0
    contribution];
plot(0:length(contribution),forplot*100);
save pc.mat pc; save score.mat score;
save contribution.mat contribution
```

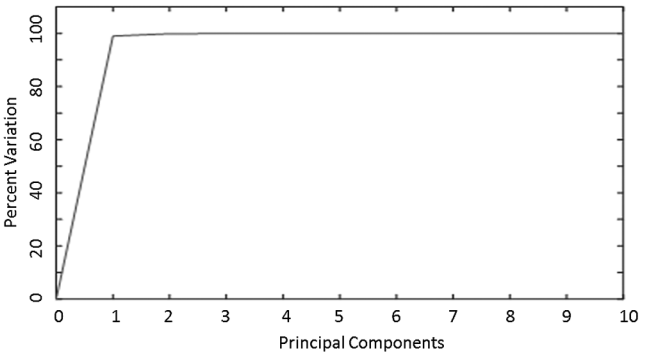

Figure 19 Scree plot generated by MATLAB. Note that *x*- and *y*-axis labels were added later, and the *x* axis was adjusted to show 0 to 10 PCs.

By plotting the columns in "`score`," the PCA plots can be obtained. For example, to plot PC1 versus PC2, one has to plot column 1 versus column 2 in "`score.`" Plotting PCA and the script to automate the same is discussed in the next section after multivariate analysis.

8.2 Principal component–linear discriminant analysis

As mentioned earlier, LDA is a supervised analysis method that can improve the classification obtained by unsupervised analysis methods, such as PCA. In "unsupervised" analysis, the spectra are input without mentioning which group they belong to. Thus, for example, if we are analyzing "healthy" and "malignant" spectra, in "unsupervised" analysis, all spectra are input together, without telling the computer which spectra are from the "healthy" group and which are from the "malignant" group. In supervised analysis, we tell the computer which group the spectra belong to. Thus, using the same example, we will tell the computer that spectra 1 to 10 are from "healthy" tissue and spectra 11 to 20 are from "malignant" tissue.

There are two ways to do LDA: directly using preprocessed spectra, or on the preprocessed spectra and use the results for LDA (PC-LDA). Using PCA before LDA filters out the noise and may give more accurate results. We have already discussed that PCA finds the most important variations in the spectra and ranks them as PC1, PC2, and so on. In most cases, the important variations are covered by the first 10 PCs; PCs beyond that are more likely to contain noise, and the probability increases with increasing PC numbers. By using only the first few PCs, the noise is eliminated, and only important variations are used for LDA, increasing the chances of getting good results. In this tutorial, we will use PC-LDA.

Supervised analysis has three components—training, validation, and prediction. Training, as the name suggests, involves teaching the computer program to recognize the spectrum as belonging to a particular group. For example, the

program will learn that spectra with a strong 1745-cm^{-1} band is "healthy" and those with a weak or no 1745-cm^{-1} peak are "malignant." The software goes through multiple "healthy" and "malignant" spectra, learning the difference between the two. The next step is validation, where we evaluate if the computer program has learned correctly. This is done by removing one or more spectra from the groups to see if the computer can still recognize the "healthy" or "malignant" spectra. The logic behind this is to test if any particular spectrum is unduly influencing the learning process. For example, let us say that one "malignant" spectrum has a very strong 1299-cm^{-1} band due to some recording error or background noise. The computer may wrongly learn that the 1299-cm^{-1} band means "malignant." In validation, when the computer is tested by removing this spectrum, it will give different results for the remaining spectra since none of them have the 1299-cm^{-1} band. The change in the result becomes an indication that the training may not have been correct. The final phase is prediction. In this, we further test how well the computer has learned. Without labeling, we input a "malignant" spectrum and ask the computer program to tell us if the spectrum is "healthy" or "malignant." We do this with several spectra. If the program identifies the group correctly, we can say that the "learning" is good. Another use of prediction is to determine the group of an unknown spectrum, which is of great benefit in clinical situations. For example, a device that can identify "malignant" from "healthy" is installed in a clinic. In such a situation, the clinician will record the spectra, and the algorithm can tell if it is "healthy" or not using prediction.

Note the concepts of sensitivity, specificity, and receiver operating characteristics (ROC) here. Sensitivity is the number of times the computer correctly identifies the group of the input spectra. For example, suppose that 9 out of 10 "malignant" spectra are identified as "malignant" and 8 out of 10 "healthy" spectra are classified as "healthy," as shown in the confusion matrix table below:

		Predicted	
		Malignant	Healthy
Actual	Malignant	9	1
	Healthy	2	8

which can be relabeled as

		Predicted	
		Disease	No disease
Actual	Disease	True positive = 9	False negative = 1
	No disease	False positive = 2	True negative = 8

As can be seen, when actual disease is also predicted correctly as disease, we call it a "true positive," and when a disease is wrongly predicted as "no disease," we call it a "false negative." When an actual "no disease" is wrongly predicted as "disease," it is a "false positive," and when "no disease" is correctly predicted as "no disease," it is a "true negative." Sensitivity is

$$\frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} * 100 = \frac{9}{9+1} * 100 = 90\%.$$

Specificity is

$$\frac{\text{True Negative}}{\text{True Negative} + \text{False Positive}} * 100 = \frac{8}{8+1} * 100 = 80\%.$$

The ROC curve helps evaluate the performance of the analysis. It is a plot of the true negative rate (1-specificity) versus the true positive rate (sensitivity). It tells us the chances that the true negative and true positive will be identified as separate. The ROC for the above example is shown in Fig. 20(a). When the area under the ROC curve is calculated, it will come to 0.7% or 70%, which tells us that there is a 70% chance that the true negative and true positive will be separate [Fig. 20(b)], as the false negative + false positive overlap region is 10 + 20 = 30%. The ideal condition where the area under the ROC curve is 1 is shown in Fig. 20(c), wherein there is no overlap region between true negative and true positive, as both are 100% [Fig. 20(d)].

We will now examine how to write the script for each step.

Training: As mentioned, we will have to tell the computer which spectra belong to which group in this step. Recall that we had stored the name of the group for each spectrum in "`group_names`." We will begin by storing the folder names in the variable "`group_names`" in a new variable "*y*" in transposed form:

```
>> y=group_names'
y =

    A
    A
    A
    ...
```

Since important variations can generally be found in the first 10 PCs, we will separate out the scores of the first 10 PCs into another variable. Since they are stored in the columns of the variable "scores," we can take out the first 10 columns, which will have scores for the first 10 PCs and store them in the variable "X":

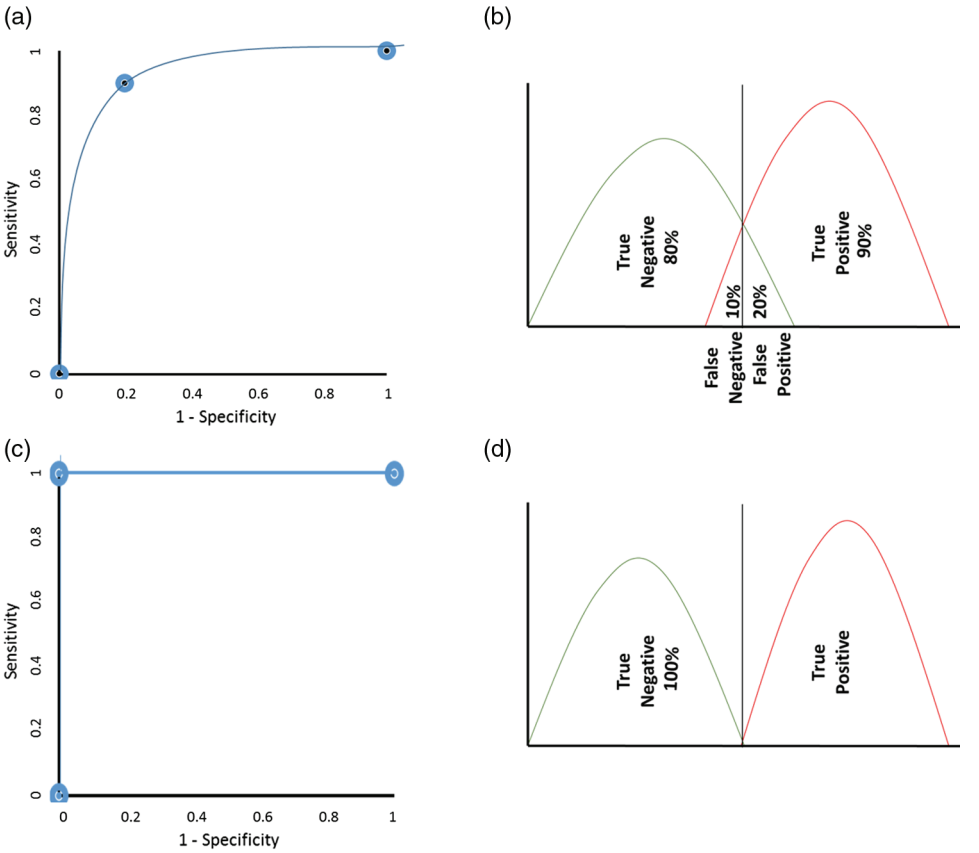

Figure 20 ROC curves.

```
>> X=score(:,1:10)
```

$X =$

26.89575	*4.828957*	...
29.06787	*5.086694*	...
27.42997	*5.222958*	...
...	...	...

Now, we have the group names in "y" and the PCA scores in "X." This is what we need as input for PC-LDA. We will build the training model by telling which rows in "X" belong to group A and which belong to group B using "y" as follows:

```
>> Model = fitcdiscr(X,y);
```

The "`fitcdiscr`" function uses the information to train the algorithm, and the outcome is stored in the variable "`Model`." We create a confusion matrix

using this "`Model.`" In order to see how well the training has worked, we ask the algorithm to display a confusion matrix. It shows how many spectra are correctly predicted for a group and how many are incorrectly classified. The commands for getting the confusion matrix are as follows:

```
>> ldaClass = resubPredict(Model);
   [LDA,grpOrder] = confusionmat(y,ldaClass)
```

As seen in Fig. 21, the output is a confusion matrix. It does not give out a labeled confusion matrix but gives the numbers and the labels separately. The above can easily be assembled into a labeled confusion matrix as shown in the following table:

	A	B
A	10	1
B	0	13

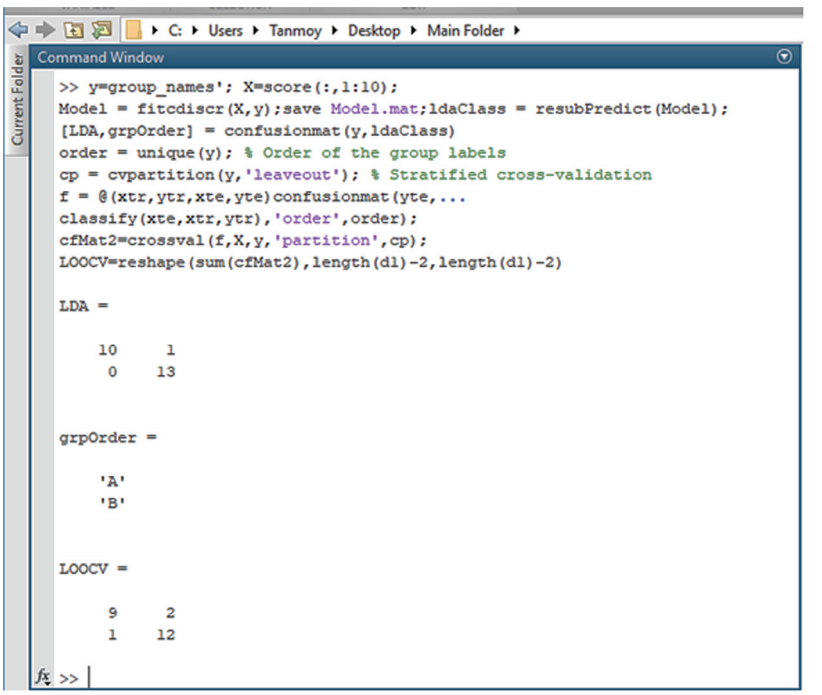

Figure 21 Output of LDA and cross-validation commands. The confusion matrices for LDA and LOOCV are seen along with the group order. The labels in group order are important to understand which columns and rows belong to which group.

Validation: The next step is validation. There are several methods of cross-validation, of which we will use LOOCV. The commands for performing LOOCV are as follows:

```
>> order = unique(y);
   cp = cvpartition(y,'leaveout');
   f = @(xtr,ytr,xte,yte)confusionmat(yte,...
   classify(xte,xtr,ytr),'order',order);
   cfMat2=crossval(f,X,y,'partition',cp);
```

Again, we will ask the algorithm to display the output in the form of a confusion matrix to evaluate the results. For this, we use the following command:

```
>> LOOCV=reshape(sum(cfMat2),length(d1)-2,length(d1)-2)
```

Note that we use `length(d1)-2` and `length(d1)-2` in the above command. This is because the dimensions of the matrix have to be specified. However, the dimensions will change depending on the number of groups. For example, in the above case, the dimensions of the confusion matrix will be 2×2. But if there are three groups, say, "A," "B," and "C," then the confusion matrix dimensions will be 3×3. In an automated algorithm, this should be identified automatically.

To program this, we use the following logic. The spectra are supplied to the algorithm in a subfolder with group names. The names of the subfolders are stored in the variable "d1." Thus, we can ask the algorithm to find out how many subfolders there are in "d1" and make the confusion matrix dimensions accordingly. So, if there are two subfolders in "d1," the confusion matrix should be 2×2, and if there are three subfolders, the confusion matrix should be 3×3.

The only catch is that "d1" always contains two additional files by default: "." and ".." (see Fig. 15). Therefore, if "d1" has two subfolders, the size of "d1" will be 4, and if there are three subfolders, the size of "d1" will be 5. To correct for this, we can subtract 2 from size "d1." We do this by using

```
length(d1)-2
```

Thus, when "d1" has two subfolders, `length(d1)=4, length(d1)-2 = 2`. Thus, `LOOCV=reshape(sum(cfMat2),length(d1)-2,length(d1)-2)` will become `LOOCV=reshape(sum(cfMat2),2,2)`, giving a 2×2 matrix as required.

Figure 21 shows the results of LOOCV, which, as before, we will arrange into a labeled confusion matrix.

	A	B
A	9	2
B	1	12

Compare this with the result before validation:

	A	B
A	10	1
B	0	13

We observe a slight change. Instead of 10/11 correctly classified as "A," after LOOCV, 9/11 are correctly classified. Similarly, the correct classification for "B" after LOOCV is 12/13 instead of 13/13. Despite this, the results of LOOCV are very close to the model, and the training can be considered robust.

LOO is just one of the several methods of cross-validation. One can opt to use *k*-fold cross validation, wherein the data are divided into "*k*" number of groups; "*k*" is any number specified by the user. So, if "*k*" is 10, the data will be divided into 10 groups of equal size. In this case, the training model will be built using nine groups, leaving one group out. The left-out group will be used to test the performance of the model built using the other nine groups. This will be repeated until all groups are left out once. LOO is a special case of *k*-fold cross-validation, where instead of groups, each spectrum is left out once. To apply *k*-fold, the command is

```
cp=cvpartition(y,'KFold', 'k')
```

Another option is the holdout method, where a particular number of data points "*p*," where "*p*" is a number specified by the user, is left out, and the model is built with the rest. The left out data points are then used to test the model built. The command for this is as follows:

```
cp=cvpartition(y,'HoldOut', 'p')
```

The user can replace "`cp =cvpartition(y,'leaveout');`" with either of the above commands and the rest of the code remains the same:

```
>> order=unique(y);
   cp=cvpartition(y,'leaveout');
   f=@(xtr,ytr,xte,yte)confusionmat(yte,...
   classify(xte,xtr,ytr),'order',order);
   cfMat2=crossval(f,X,y,'partition',cp);
```

Finally, we will save the model and the confusion matrix for later viewing/analysis using the "save" command:

```
>> save Model.mat; save LDA.mat; save LOOCV.mat
```

Script: Preprocess + PCA + PC-LDA

```
clear;clc;
s=[];names=[];nome=[];group_names=[];
cd('C:\Users\Tanmoy\Desktop\Main Folder')
d1=dir;
for ii=3:length(d1);
    n1=d1(ii).name;
    cd(n1);
    d=dir('*.txt');
    for i=1:length(d);
n=d(i).name;
data=dlmread(n); datax=data(:,1); datay=data(:,2);
fd=diff(datay)./diff(datax); dataxd=datax(1:length(fd));
p1=find(dataxd==1200); p2=find(dataxd==1800);
yInterpolate=fd(p2:p1,:); xInterpolate=dataxd(p2:p1,:);
Area=trapz(xInterpolate,yInterpolate); AreaS=Area/100;
norm=yInterpolate/AreaS;
s=[s norm];
names=[names {n}];group_names=[group_names {n1}];
end
nome=[nome {n1}]
cd('C:\Users\Tanmoy\Desktop\Main Folder')
end
[pc,score,latent]=princomp(s');
contribution=cumsum (latent)./sum(latent);
PC_sig= contribution (1:10,:);
forplot=[0
    contribution];
plot(0:length(contribution),forplot*100);
save pc.mat pc; save score.mat score;
save contribution.mat contribution
y=group_names'; X=score(:,1:10);
Model=fitcdiscr(X,y);save Model.mat;ldaClass=resubPredict(Model);
[LDA,grpOrder]=confusionmat(y,ldaClass)
order=unique(y); % Order of the group labels
cp=cvpartition(y,'leaveout'); % Stratified cross-validation
f=@(xtr,ytr,xte,yte)confusionmat(yte,...
classify(xte,xtr,ytr),'order',order);
cfMat2=crossval(f,X,y,'partition',cp);
LOOCV=reshape(sum(cfMat2),length(d1)-2,length(d1)-2)
    save LDA.mat
    save LOOCV.mat
```

Cross-validation can also be performed directly on the "Model" by using the following:

```
cfMat2=crossval(Model,'Leaveout','on');
[label,pcldascore]=kfoldPredict(cfMat2);
```

and a confusion matrix is created by

```
LOOCV=confusionmat(y,label)
```

In this case, the code will look as follows (the above code replaces the crossed-out lines):

```
clear;clc;
s=[];names=[];nome=[];group_names=[];
cd('C:\Users\Tanmoy\Desktop\Main Folder')
d1=dir;
for ii=3:length(d1);
    n1=d1(ii).name;
    cd(n1);
    d=dir('*.txt');
    for i=1:length(d);
n=d(i).name;
data=dlmread(n); datax=data(:,1); datay=data(:,2);
fd=diff(datay)./diff(datax); dataxd=datax(1:length(fd));
p1=find(dataxd==1200); p2=find(dataxd==1800);
yInterpolate=fd(p2:p1,:); xInterpolate=dataxd(p2:p1,:);
Area=trapz(xInterpolate,yInterpolate); AreaS=Area/100;
norm=yInterpolate/AreaS;
s=[s norm];
names=[names {n}];group_names=[group_names {n1}];
end
nome=[nome {n1}]
cd('C:\Users\Tanmoy\Desktop\Main Folder')
end
[pc,score,latent]=princomp(s');
contribution=cumsum (latent)./sum(latent);
PC_sig= contribution (1:10,:);
forplot=[0
     contribution];
plot(0:length(contribution),forplot*100);
save pc.mat pc; save score.mat score;
save contribution.mat contribution
y=group_names'; X=score(:,1:10);
Model=fitcdiscr(X,y);save Model.mat;ldaClass=resubPredict(Model);
[LDA,grpOrder]=confusionmat(y,ldaClass)
```

```
cfMat2=crossval(Model,'Leaveout','on');
[label,pcldascore]=kfoldPredict(cfMat2);
LOOCV=confusionmat(y,label)
order=unique(y); % Order of the group labels
cp=cvpartition(y,'leaveout'); % Stratified cross validation
f=@(xtr,ytr,xte,yte)confusionmat(yte,...
classify(xte,xtr,ytr),'order',order);
cfMat2=crossval(f,X,y,'partition',cp);
LOOCV=reshape(sum(cfMat2),length(d1)-2,length(d1)-2)
    save LDA.mat
    save LOOCV.mat
```

We discussed earlier that LDA can be performed directly without going through PCA. In this case, we can ask the algorithm to directly perform LDA on the preprocessed spectra stored in variable "s" using the following. In this case, all commands will remain the same, except "X = score(:,1:10)" will be replaced by "X = s."

Prediction: We will now discuss prediction. As mentioned previously, in this step, we ask the trained algorithm to determine the "group" of an unknown spectrum. We have already trained a "model" for groups "A" and "B," which we will use to predict the "group" of the unknown spectra. For this, we will first need to tell the algorithm that we will be using "model" for prediction. So, we will open the folder where the PC-LDA model is saved and load the same:

```
>> clear;clc;
    cd('C:\Users\Tanmoy\Desktop\Main Folder');
    load('Model.mat');
```

Next, for the sake of convenience, we will put all of the unknown spectra in this example, labeled as "test (1)," "test (2)," and so on, in a single folder called "T," as shown in Fig. 22.

Then, we will preprocess all the spectra and perform PCA in the same manner as before, but the path here will be different:

```
>> s=[];names=[];
    cd('C:\Users\Tanmoy\Desktop\T')
    d=dir('*.txt');
    for i=1:length(d);... y=names'; X=score(:,1:10); end
```

We will then use the model to predict the PCA scores of test spectra:

```
>> Pred=predict(Model,X);
    output=[y Pred]
```

Figure 23 shows the outcome of the test prediction. In the left column are the names of the spectra, and in the right are the predictions made by the algorithm. Thus, according to the trained software, "test (1)" belongs to group "A," "test (10)" belongs to group "B," and so on.

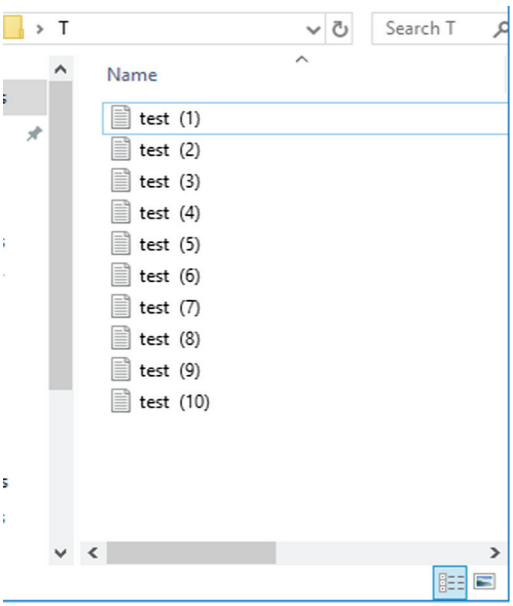

Figure 22 Example of a folder created to store test spectra.

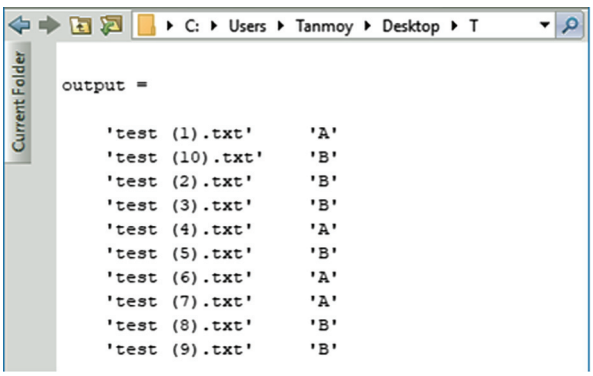

Figure 23 Output of test prediction. The left column lists the spectra names, and the right column displays the predictions.

Script: PC-LDA Test Prediction

```
clear;clc;
cd('C:\Users\Tanmoy\Desktop\Main Folder');
load('Model.mat');

s=[];names=[];
cd('C:\Users\Tanmoy\Desktop\T')
d=dir('*.txt');
for i=1:length(d);
n=d(i).name;
data=dlmread(n); datax=data(:,1); datay=data(:,2);
fd=diff(datay)./diff(datax); dataxd=datax(1:length(fd));
p1=find(dataxd==1200); p2=find(dataxd==1800);
yInterpolate=fd(p2:p1,:); xInterpolate=dataxd(p2:p1,:);
Area=trapz(xInterpolate,yInterpolate); AreaS=Area/100;
norm=yInterpolate/AreaS;
s=[s norm];
names=[names {n}];
end
[pc,score,latent]=princomp(s');
y=names'; X=score(:,1:10);

Pred=predict(Model,X);
output=[y Pred]
```

8.3 Support vector machine

An SVM is another supervised analysis method that can help classify groups. This method works with two groups at a time, so it cannot be used on multiple groups at once. Instead, it should be done on two groups at a time. The advantage of an SVM is that it can give better classification for groups that are very close to each other. For example, consider we analyze two groups "A" and "B" with 10 spectra each. Imagine the second spectrum in "A" (A2) is very similar to the third spectrum of "B" (B3). LDA may find it difficult to classify A2 and B3 and may erroneously put A2 in "B" or erroneously put B3 in "A." If there are more spectra close to each other, the result of LDA classification will be very poor. An SVM can help get better results as it can differentiate close points. The reader can refer to the books and papers mentioned in the introduction to get a detailed idea of how it works. Put in very simplified terms, LDA finds a single line to classify "A" and "B." We need to imagine a plot with the 10 "A" spectra and 10 "B" spectra as points on the plot. LDA draws a line that separates the "A" points from the "B." But because A2 and B3 are very close, they may lie on the

wrong side of the line. An SVM uses supporting lines, called support vectors, in addition to the main line to achieve classification. This increases the chance of all points from a group being on the correct side of the lines, giving better classification results.

An SVM is particularly useful in nonlinear problems, that is, in cases where there is no possibility of any line to classify two groups. Imagine all points from "A" are surrounded by "B" points on a plot. There is no way LDA can draw a line that can separate "A" and "B." An SVM can draw a circle around "A" points, separating them from "B" points.

The following will provide some simple commands that can help one perform SVMs and integrate them in the code we have built to automate the process. The user can build upon it to perform more complicated SVM operations.

The following is the script that we applied for LDA:

```
clear;clc;

s=[];....
.........
end
[pc,score,latent]=princomp(s');
contribution=cumsum (latent)./sum(latent);
PC_sig= contribution (1:10,:);
forplot=[0
     contribution];
plot(0:length(contribution),forplot*100);
save pc.mat pc; save score.mat score;
save contribution.mat contribution
y=group_names'; X=score(:,1:10);
Model=fitcdiscr(X,y);save Model.mat;
ldaClass=resubPredict(Model);
[LDA,grpOrder]=confusionmat(y,ldaClass)
order=unique(y); % Order of the group labels
cp =cvpartition(y,'leaveout'); % Stratified cross-
validation
f=@(xtr,ytr,xte,yte)confusionmat(yte,...
classify(xte,xtr,ytr),'order',order);
cfMat2=crossval(f,X,y,'partition',cp);
LOOCV=reshape(sum(cfMat2),length(d1)-2,length(d1)-2)
     save LDA.mat
     save LOOCV.mat
```

We have performed PCA and have the scores stored in the variable "X" and group names in "y." For LDA, we used the following command to build the model:

```
Model = fitcdiscr(X,y);
```

For building the SVM model, we will use

```
SVMModel = fitcsvm(X,y)
```

Then, we use the following to get the confusion matrix for LDA:

```
ldaClass = resubPredict(Model);
[LDA,grpOrder] = confusionmat(y,ldaClass)
```

We use the following for an SVM:

```
svmClass=resubPredict(SVMModel)
[SVM,grpOrder] = confusionmat(y,svmClass)
```

The commands for SVM cross-validation are similar to those of LDA:

```
CVSVMModel = crossval(SVMModel)
[label,svmscore] = kfoldPredict(CVSVMModel)
SVMCV = confusionmat(y,label)
```

Script: Preprocess + PCA + PC-SVM

```
clear;clc;
s=[];names=[];nome=[];group_names=[];
cd('C:\Users\Tanmoy\Desktop\Main Folder')
d1=dir;
for ii=3:length(d1);
    n1=d1(ii).name;
    cd(n1);
    d=dir('*.txt');
    for i=1:length(d);
n=d(i).name;
data=dlmread(n); datax=data(:,1); datay=data(:,2);
fd=diff(datay)./diff(datax);      dataxd=datax(1:
length(fd));
p1=find(dataxd==1200); p2=find(dataxd==1800);
yInterpolate=fd(p2:p1,:);
xInterpolate=dataxd(p2:p1,:);
Area=trapz(xInterpolate,yInterpolate);
AreaS=Area/100;
norm=yInterpolate/AreaS;
s=[s norm];
names=[names {n}];group_names=[group_names {n1}];
end
nome=[nome {n1}]
```

```
cd('C:\Users\Tanmoy\Desktop\Main Folder')
end
[pc,score,latent]=princomp(s');
y=group_names'; X=score(:,1:10);
SVMModel=fitcsvm(X,y)
svmClass=resubPredict(SVMModel)
[SVM,grpOrder]=confusionmat(y,svmClass)
CVSVMModel=crossval(SVMModel)
[label,svmscore]=kfoldPredict(CVSVMModel)
SVMCV=confusionmat(y,label)
save SVMModel.mat
save SVM.mat
save SVMCV.mat
```

As before, SVM can be performed without PCA by using the preprocessed spectra directly as input instead of PCA scores by using "`X=s'`."

Finally, we can perform prediction using the SVM model by applying the following commands:

```
Pred=predict(SVMModel,X);
output=[y Pred]
```

where "`X`" contains the preprocessed/PCA scores of spectra whose groups are to be predicted. As can be seen again, the command structures are very similar to ones used in LDA:

Script: PC-SVM Test Prediction

```
clear;clc;
cd('C:\Users\Tanmoy\Desktop\Main Folder');
load('Model.mat');

s=[];names=[];
cd('C:\Users\Tanmoy\Desktop\T')
d=dir('*.txt');
for i=1:length(d);
n=d(i).name;
data=dlmread(n); datax=data(:,1); datay=data(:,2);
fd=diff(datay)./diff(datax); dataxd=datax(1:length(fd));
p1=find(dataxd==1200); p2=find(dataxd==1800);
yInterpolate=fd(p2:p1,:);
xInterpolate=dataxd(p2:p1,:);
Area=trapz(xInterpolate,yInterpolate);
AreaS=Area/100;
norm=yInterpolate/AreaS;
```

```
s=[s norm];
names=[names {n}];
end
[pc,score,latent]=princomp(s');
y=names'; X=score(:,1:10);

Pred=predict(SVMModel,X);
output=[y Pred]
```

9 PCA Plotting

Plotting PCA is a routine operation performed for the majority of spectroscopic analysis pertaining to biological experiments, especially disease diagnosis experiments. Getting the plots manually can be very tedious, especially when several combinations of PCs need to be checked for the best representation of group classification. The process becomes even more time-consuming when the number of groups increases, and each group has to be selected and colored separately. The following section will show how to automate the entire process.

To understand the programming required, let us consider hypothetical data:

```
>> dat=
        [1      2      3
         3      4      5
        10     20     30
        30     40     50]
```

After pressing the Enter key, the following output will appear in the MATLAB window:

```
dat =
        1      2      3
        3      4      5
       10     20     30
       30     40     50
```

Let columns 1, 2, and 3 be the scores of PCs 1, 2, and 3, respectively. Let the first two rows be scores of "healthy" tissue spectra, and rows 3 and 4 be scores of "malignant" tissue spectra, as follows:

Group/PC	PC1 score	PC2 score	PC3 score
Healthy	1	2	3
Healthy	3	4	5
Malignant	10	20	30
Malignant	30	40	50

To plot PC1 versus PC2, the command will be ("filled" is for filled circle markers)

```
>> scatter (dat (:,1), dat (:,2), 'filled')
```

If we wish to color the "healthy" black and "malignant" red, we will have to type the commands separately for each group ("k" is for black, and "r" is for red; see Figs. 24 and 25):

```
>> scatter (dat (1:2,1), dat (1:2,2), 'k', 'filled')
   scatter (dat (3:4,1), dat (3:4,2), 'r', 'filled')
```

If we add the command "hold on" between the two lines, both will be displayed together:

```
>> scatter (dat (1:2,1), dat (1:2,2), 'k', 'filled');
   hold on
   scatter (dat (3:4,1), dat (3:4,2), 'r', 'filled')
```

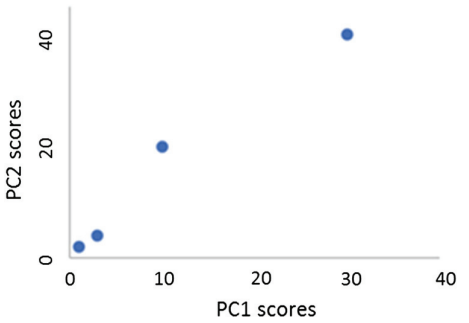

Figure 24 Plot of PC1 score versus PC2 score for "healthy" and "malignant" sample spectra.

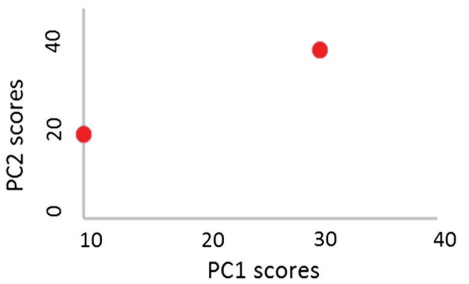

Figure 25 Plot of PC1 versus PC2 scores of "malignant" spectra, colored red. The scores of "healthy" spectra are not visible, as a new plot replaced the old plot.

If the start and end positions of the groups are in a matrix, it will be possible to automate the process (Figs. 26 and 27). Let us put the start numbers in an array "start" and end numbers in an array "endm":

```
>> start = [1 3]
```

start = 1 3

```
>> endm = [2 4]
```

endm = 2 4

We can write a "for" loop:

```
>> for sevi = 1:length (start)
     scatter (dat (start (sevi):end (sevi), 1), dat
     (start (sevi):end(sevi), 2), 'filled'); hold on
   end
```

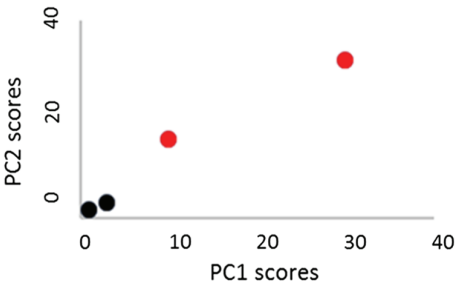

Figure 26 By applying `hold on` between commands, both "healthy" and "malignant" spectra PC1 versus PC2 scores are plotted. Using separate commands helps color groups separately.

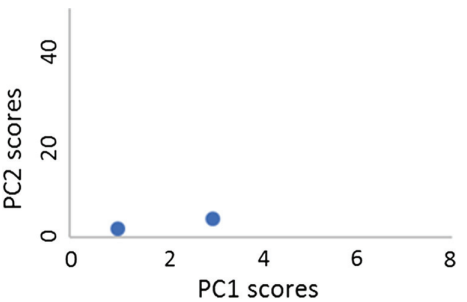

Figure 27 Automating PCA plotting—first round.

Since "`length(start) = 2,`" the "for" loop will have two rounds, "`sevi = 1`" and "`sevi = 2.`" When "`sevi = 1,`" "`start (sevi)`" = "`start (1)`" = "`1.`" Similarly, "`end (sevi)`" = "`end (1)`" = "2." Thus, the command in the first round of the "for" loop will be

```
>> scatter (dat (1:2, 1), dat (1:2, 2), 'filled'); hold on
```

In the second round of the "`for`" loop, "`sevi = 2.`" Therefore, "`start (sevi)`" = "`start (2)`" = "3"; and "`end (sevi)`" = "`end (2)`" = "`4.`" Thus, in the second round, the command will be automatically

```
>> scatter (dat (3:4, 1), dat (3:4, 2), 'filled');
   hold on
```

The markers for "healthy" and "malignant" are plotted together, because of the "**`hold on`**" command (Fig. 28).

The colors remain the same. To have different colors, we can create a separate matrix for colors:

C = {"k," "r," "b," "g," "y," [.5 .6 .7],[.8 .2 .6]}

Here, k, r, b, g, y, [.5 .6 .7], and [.8 .2 .6] stand for black, red, blue, green, yellow, gray, and purple colors, respectively. We can then ask MATLAB to use different colors during each round of plotting by (Fig. 29)

```
>> for sevi = 1:length (start)
       scatter (dat (start (sevi):end (sevi), 1), dat
       (start (sevi):end(sevi), 2), C {sevi} ,
       'filled'); hold on
   end
```

Since "`sevi = 1`" in the first round, "`C {sevi}`" = "`C {1}`" = "`k,`" that is black. In the second round, "`sevi = 2`"; "`C {sevi}`" = "`C {2}`" = "`r,`" that is red. With the "`hold on`," we get the plot in Fig. 28.

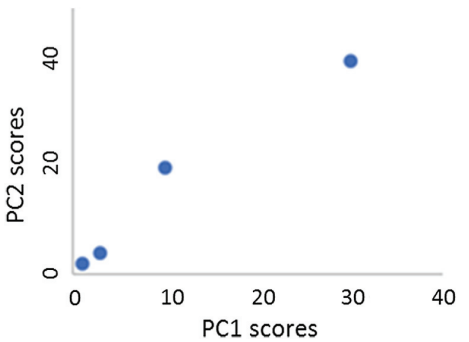

Figure 28 Automating PCA plotting—second round.

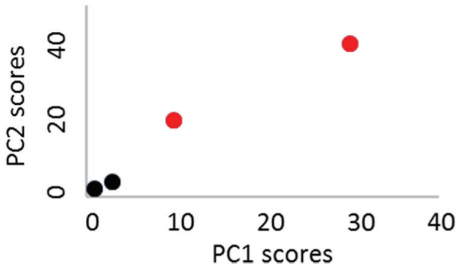

Figure 29 Including a separate matrix for color and calling them in a scatter function helps to color groups separately and automatically.

We can similarly change the markers by adding a marker matrix:

```
mkr={'o' 's' 'd' 'h'}
    >> for sevi=1:length (start)
           scatter (dat (start (sevi):end (sevi), 1), dat
           (start (sevi):end(sevi), 2), C {sevi}, mkr
           {sevi}, 'filled'); hold on
       end
```

It is not enough to automate the plotting of individual plots. We need the script to plot scatter graphs with each group colored separately for every combination of the first three PCs. We need the *x* axis to change from 1 to 3. So,

```
>> facsev=1:3
facsev=1
facsev=2
facsev=3
```

We need the *y* axis to start with a number after the *x* axis. For example, when the *x* axis is 1, the *y* axis should change from 2 to 3. This way, we will get plots for PC1 versus PC2, and PC1 versus PC3. Similarly, when the *x* axis = 2, the *y* axis should be 3, and we will get PC2 versus PC3. Thus,

```
>> facsev2=facsev+1:3
facsev2=2
facsev2=3
facsev2=4
```

We will achieve this by adding two additional "`for`" loops for the *x* and *y* axes, respectively. We will add the command "`figure ()`," so that every plot of PC*x* versus PC*y* appears in a different figure:

```
>>for facsev=1:3;
      for facsev2=facsev+1:3; figure ();
          for sevi=1:length (start)
```

```
                scatter (dat (start (sevi):end (sevi), 1),
                dat (start(sevi):end(sevi), 2), C {sevi},
                mkr {sevi}, 'filled'); hold on
            end
        end
    end
```

The next step will be to automatically get the "start" and "endm" matrices from the PC-LDA script to use them for PCA plotting. We can add some commands in the script to save the group start and end numbers, just as we saved the spectra and group names.

We will use the same example as before—importing spectra from two subfolders A and B, contained within the main folder. Subfolder A has 11 spectra, whereas subfolder B has 13 spectra. When we import and preprocess, we save all spectra in columns of variables called "s." Thus, "s" has 11 + 13 = 24 columns, and 301 rows corresponding to wavenumbers from 1200 to 1800 cm^{-1}. In "s," columns 1 to 11 belong to group A, and columns 12 to 24 belong to group B. Thus, group A starts from column 1, while group B starts from column 12. The end columns for groups A and B are 11 and 24, respectively. This means that we need to obtain two matrices, "start" = [1 12] and "endm" = [11 24].

The spectra are imported one by one until one subfolder is finished. Then the script goes to the main folder and opens the next subfolder. This means that before the script goes to the main folder, "s" contains only spectra from one subfolder. If we determine the number of columns in "s" at this point, we will know the end number for the subfolder. In our example, when subfolder A is read and processed, "s" will have 11 columns. This can be determined using the following command:

```
>> amt = size (s)
```

amt = 301 11

Here, 301 and 11 correspond to rows and columns, respectively, but we need only columns. So,

```
>> amt1 = amt (:,2)
```

amt1 = 11

We will save the "amt1" in every round of the "for" loop. First, we will declare two arrays outside the loop:

```
>> fstno = [1]; lastno = [0];
```

We will then store "amt1" in "lastno" by declaring it inside the "for" loop:

```
>> lastno = [lastno amt1];
```

The PC-LDA script will look as follows:

```
>> s=[];names=[];nome=[];group_names=[]; fstno = [1];
   lastno = [0];

    cd('C:\Users\Tanmoy\Desktop\Main Folder')
    d1=dir;
    for ii=3:length(d1);
        n1=d1(ii).name;
        cd(n1);
        d=dir('*.txt');
        for i=1:length(d);
            n=d(i).name;
            data=dlmread(n); datax=data(:,1); datay=
            data(:,2);
            fd=diff(datay)./diff(datax); dataxd=da-
            tax(1:length(fd));
            p1=find(dataxd==1200);p2=find(dataxd==1800);
            yInterpolate=fd(p2:p1,:); xInterpolate=
            dataxd(p2:p1,:);
            Area=trapz(xInterpolate,yInterpolate);
            AreaS=Area/100;
            norm=yInterpolate/AreaS;
            s=[s norm];
            names=[names   {n}];group_names=[group_-
            names {n1}];
        end
        nome=[nome {n1}];
        amt = size (s); amt1 = amt (:,2); lastno = [lastno
        amt1];

        fstno=[fstno lastno(:,i-1)+1];

        cd('C:\Users\Tanmoy\Desktop\Main Folder')
    end
```

Let us consider an addition to the script. In the first iteration of the "`for`" loop, "`ii = 3`," "`n1=d1(ii).name`," so "n1 = A," that is the subfolder "A." "`cd(n1)`" will become "cd(A)" and open the subfolder "A." "`d=dir('*.txt')`" will open all ".txt" files and save them in "d." In this case, as we have already seen, "`amt = [301 11]`," because subfolder "A" has 11 spectral files in it, which get stored in columns, and 301 wavenumbers, which are stored in rows. From this, we need the number of files in subfolder "A," and can easily get it using "`amt1 = amt (:,2)`," which gives "amt1 = 11." We store this in the variable "lastno" using "`lastno = [lastno amt1]`." So now, "lastno" = 11. This will tell the algorithm that it will have to plot all up to the

11th column in the same color in the PCA scatter plot. But this does not tell it from which column to start. We know that the 1st to 11th column have to be plotted in the same color as they all belong to subfolder "A." We can calculate this by using "`fstno = [1 lastno(:,3-1) +1]`." The calculations will occur as follows:

```
fstno = [1 lastno(:,3-1)+1] = [1 lstno (:,2)+1] = [1 11+1]
= [1 12].
```

We now have a matrix "fstno" = [1 12] and "lstno" = [0 11].

In the same manner, in the second iteration of the "`for`" loop:

```
ii =4
amt = [301 24], amt1 = 24,
'lastno = [0 11 amt1]' = '[0 11 24]',
```

and

```
'fstno = [1 12 lastno(:,4-1)+1]' = '[1 12 lastno(:,3)+1]'
= '[1 12 24+1]' = '[1 12 25]'.
```

We now have a matrix "fstno" = [1 12 25] and "lstno" = [0 11 24].

We need PCA to be plotted from 1 to 11 and then 12 to 24. Thus, the numbers "25" in "fstno" and "0" in "lstno" are not required. We will remove them as follows:

```
>> szfst=size(fstno)
szfst = 1 3

>> start=fstno(:,1:szfst(:,2)-1)
start = 1 12
```

In the above command, "`szfst(:,2)-1`" translates to "`1:(3-1)`," which is equal to "`1:2`." So, "`start=fstno(:,1:szfst(:,2)-1)`" becomes "`start=fstno(:,1:2)`," which means "`start = [1 12].`"

Similarly,

```
>> szlst=size(lastno)
szlst = 1 3

>> endm=lastno(:,2:szlst(:,2))
endm = 11 24
```

Now that "`start`" and "`endm`" are automatically obtained by the script, the entire process of preprocessing, PCA, and PCA plotting becomes automated.

Three things remain to be added to the automated PCA plotting script—graph legends, titles, and loading plots. Legends are extremely convenient to identify the groups instead of having to look up which color is mentioned first in the array.

Moreover, sometimes the order of importing in MATLAB is not the same as seen in Windows. In such cases, legends become vital.

We can easily give legends to the graphs by using the sub-folder names stored in variable "group_names." We ask MATLAB to save the first "group_name" in every round of the "for" loop:

```
>> Legend{sevi} = [{group_names(start(sevi))}]
```

Legend1 = A (in round 1)
Legend2 = B (in round 2)

We also store the number of rows and columns in the "Legend" by

```
>> sz23=size(Legend)
```

sz23 = 1 1 (in round 1)
sz23 = 1 2 (in round 2)

At the end of plotting each graph, we use the "for" loop to create the legends:

```
>> for ijk=1:sz23(1,2);
          L{ijk} = [Legend{1,ijk}{1,1}{1,1}];
    end
```

L1 = A
L2 = B

Finally, we command to place the legend in the graph by

```
>> legend(L)
```

We create the titles for each graph—PC1, 2 denoting the plot of PC1 versus PC2, and so on—by

```
>> T{facsev}=['PCA Plot factors', num2str(facsev),',',
   num2str(facsev2)]
```

T1 = PCA Plot factors 1, 2
T2 = PCA Plot factors 1, 3
T2 = PCA Plot factors 1, 4
T1 = PCA Plot factors 1, 5
T2 = PCA Plot factors 2, 3
T2 = PCA Plot factors 2, 4
T1 = PCA Plot factors 2, 5
T2 = PCA Plot factors 3, 4
T2 = PCA Plot factors 3, 5
T1 = PCA Plot factors 4, 5

and call to place the title in the graph by

```
>> title(T)
```

Finally, we create and save the PC loadings as follows:

```
>> FL1=[xInterpolate pc(:,1)];
   FL2=[xInterpolate pc(:,2)];
   FL3=[xInterpolate pc(:,3)];
   FL4=[xInterpolate pc(:,4)];
   FL5=[xInterpolate pc(:,5)];
   save FactorLoading1.txt FL1 -ascii
   save FactorLoading2.txt FL2 -ascii
   save FactorLoading3.txt FL3 -ascii
   save FactorLoading4.txt FL4 -ascii
   save FactorLoading5.txt FL5 -ascii
```

Script: Preprocess + PCA + PC-LDA + Plot PCA (Fig. 30)

```
s=[];names=[];nome=[];group_names=[]; fstno =[1];
lastno=[0];
cd('C:\Users\Tanmoy\Desktop\Main Folder')
d1=dir;
for ii=3:length(d1);
    n1=d1(ii).name;
    cd(n1);
    d=dir('*.txt');
    for i=1:length(d);
        n=d(i).name;
        data=dlmread(n);datax=data(:,1);datay=data(:,2);
        fd=diff(datay)./diff(datax); dataxd=datax(1:
        length(fd));
        p1=find(dataxd==1200); p2=find(dataxd==1800);
        yInterpolate=fd(p2:p1,:); xInterpolate=dataxd
        (p2:p1,:);
        Area=trapz(xInterpolate,yInterpolate);
        AreaS=Area/100;
        norm=yInterpolate/AreaS;
        s=[s norm];
        names=[names{n}];group_names=[group_names{n1}];
    end
    nome=[nome {n1}];
    amt=size(s);amt1=amt(:,2);lastno=[lastno amt1];

    fstno=[fstno lastno(:,ii-1)+1];

    cd('C:\Users\Tanmoy\Desktop\Main Folder')

end
```

```
[pc,score,latent]=princomp(s');
contribution=cumsum (latent)./sum(latent);
PC_sig= contribution (1:10,:);
forplot=[0
    contribution];
plot(0:length(contribution),forplot*100);
save pc.mat pc; save score.mat score;
save contribution.mat contribution

y=group_names'; X=score(:,1:10);
Model =fitcdiscr(X,y);save  Model.mat;ldaClass
=resubPredict(Model);
[LDA,grpOrder]=confusionmat(y,ldaClass)
order=unique(y); % Order of the group labels
cp =cvpartition(y,'leaveout'); % Stratified cross-
validation
f=@(xtr,ytr,xte,yte)confusionmat(yte,...
classify(xte,xtr,ytr),'order',order);
cfMat2=crossval(f,X,y,'partition',cp);
LOOCV=reshape(sum(cfMat2),length(d1)-2,
length(d1)-2)
    save LDA.mat
    save LOOCV.mat

szfst=size(fstno);
szlst=size(lastno);
start=fstno(:,1:szfst(:,2)-1);
endm=lastno(:,2:szlst(:,2));

C ={'k','b','r','g','y',[.5 .6 .7],[.8 .2 .6]};
mkr={'o' 's' 'd' 'h'};
aha=[1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2
3 4 5 6];
aha2=[1 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5 5 5 6
6 6 6 6 6];

for facsev=1:5;
   for facsev2=facsev+1:5
          figure();
         for sevi=1:length(start)
            div=sevi/6;
            Legend{sevi} =[{group_names(start(-
            sevi))}];sz23=size(Legend);
```

```
scatter(score(start(sevi):endm(sevi),facsev),
score((start(sevi):endm(sevi)),facsev2),100,
C{aha(1,sevi)},mkr{aha2(1,sevi)},'filled');
            hold on
            T{facsev}=['PCA Plot factors', num2str
            (facsev),',',num2str(facsev2)];
        end
            for ijk=1:sz23(1,2);
            L{ijk}=[Legend{1,ijk}{1,1}{1,1}];
            end
     legend(L)
     title(T);
   end
  T=[];
end
FL1=[xInterpolate pc(:,1)];
FL2=[xInterpolate pc(:,2)];
FL3=[xInterpolate pc(:,3)];
FL4=[xInterpolate pc(:,4)];
FL5=[xInterpolate pc(:,5)];
save FactorLoading1.txt FL1 -ascii
save FactorLoading2.txt FL2 -ascii
save FactorLoading3.txt FL3 -ascii
save FactorLoading4.txt FL4 -ascii
save FactorLoading5.txt FL5 -ascii
```

Note that the matrices "aha" and "aha2" are used to change the color and marker according to a pattern. They allow all colors to be used up with the first marker before changing to the second marker. This way, $7 \times 4 = 28$ combinations are possible before the color and marker are repeated. Thus, 28 groups can be distinctly plotted using this script.

10 Turning Features On and Off

There may be situations, especially in the exploratory phase of study, where the user might be interested in turning off a particular preprocessing step or have some steps on while others are off. There may be times when one is interested in only looking at the confusion matrix and not PCA plots. A simple way to achieve this is to copy/paste only the required parts of the code, but that can be tedious and error prone. It is much more convenient to have all of the on/off features at the beginning, where one can quickly make changes and refer back to it after the analysis. In this section, we take this fine-tuning into consideration.

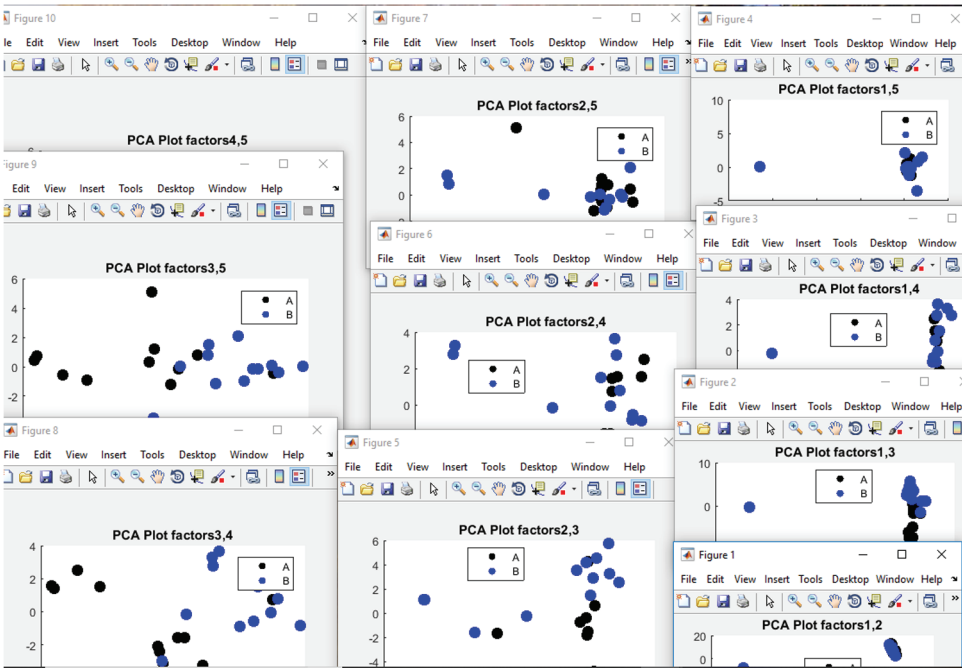

Figure 30 Output of automatic PCA plotting. Different combinations of PCs are automatically plotted with titles PCA plot factors PC*x* axis, PC*y* axis. Note that groups are automatically colored differently.

For this part, we will be using the condition function "if." The function offers the option to perform an operation only when a condition is met. For example,

```
>> a=7;
>> if a > 5
       b=1
     else
       b=2
   end
b=1
```

The output is thus based on the condition. Since $a = 7$, $a > 5$; hence, MATLAB displays $b = 1$, as instructed.

Let us use this for the "`import`" function. We have already seen that if the extension is "txt" or "asc," we use "`dlmread`," whereas if the extension is "csv," we use "`csvread`." We can make the code more user-friendly by asking for user input regarding the extension beforehand and automatically deciding the correct command. We ask the user to make changes in the code at a particular location:

```
%%% Enter 1 for 'csv', 2 for 'txt', and 3 for 'asc' %%%%%%%%%
Extension =
```

The user provides input:

```
Extension =3
```

The code will now have to be changed to react as per user input. Originally, the code was as follows:

```
>> clear;clc;

s=[];names=[];nome=[];group_names=[]; fstno =[1];
lastno=[0];

cd('C:\Users\Tanmoy\Desktop\Main Folder')
d1=dir;
for ii=3:length(d1);
      n1=d1(ii).name;
      cd(n1);
       d=dir('*.txt');
      for i=1:length(d);
            n=d(i).name;
            data=dlmread(n); datax=data(:,1); datay=
            data(:,2);
            fd=diff(datay)./diff(datax); dataxd=datax
            (1:length(fd));
            p1=find(dataxd==1200); p2=find(dataxd==
            1800);
            yInterpolate=fd(p2:p1,:); xInterpolate=
            dataxd(p2:p1,:);
            Area=trapz(xInterpolate,yInterpolate);
            AreaS=Area/100;
            norm=yInterpolate/AreaS;
            s=[s norm];
            names=[names {n}];group_names=[group_names
            {n1}];
    end
    nome=[nome {n1}];
    amt=size (s); amt1=amt (:,2); lastno=[lastno amt1];

    fstno=[fstno lastno(:,ii-1)+1];

    cd('C:\Users\Tanmoy\Desktop\Main Folder')

end

[pc,score,latent]=princomp(s');...
```

This will change to

```
>> clear;clc;

%%% Enter 1 for 'csv', 2 for 'asc', and 3 for 'txt' %%%%%%%%%%

Extension =

%%% Automated script %%%%%%%%%%

s=[];names=[];nome=[];group_names=[]; fstno =[1];
lastno=[0];

if Extension==1;
    Ext='csv';
else if Extension==2;
        Ext='txt';
    else if Extension==3;
            Ext='asc';
        end
    end
end

cd('C:\Users\Tanmoy\Desktop\Main Folder')
d1=dir;
for ii=3:length(d1);
      n1=d1(ii).name;
      cd(n1);
       d=dir(strcat('*.',Ext));
      for i=1:length(d);
            n=d(i).name;

            if Extension==1;
                  norm=csvread(n,skip_lines,0);
            else
                  norm=dlmread(n,'',skip_lines,0);
            end

            datax=norm(:,1); datay=norm(:,2);
            fd=diff(datay)./diff(datax); dataxd=datax
            (1:length(fd));
            p1=find(dataxd==1200); p2=find(dataxd==1800);
            yInterpolate=fd(p2:p1,:); xInterpolate=
            dataxd(p2:p1,:);
            Area=trapz(xInterpolate,yInterpolate);
            AreaS=Area/100;
            norm=yInterpolate/AreaS;
            s=[s norm];
```

```
            names=[names {n}];group_names=[group_names
            {n1}];
        end
        nome=[nome {n1}];
        amt =size (s); amt1 =amt (:,2); lastno =[lastno
        amt1];

        fstno=[fstno lastno(:,ii-1)+1];

        cd('C:\Users\Tanmoy\Desktop\Main Folder')
    end

    [pc,score,latent]=princomp(s');...
```

All data will be stored in "norm" from the beginning to avoid confusion.

We can turn "on" or "off" the derivatization step and also ask the user to choose a particular order of derivative—first or second—by making the following changes:

```
>> clear;clc;

%%% Change parameters here %%%%%%%%%%

Extension =3;   %%% Enter 1 for 'csv', 2 for 'txt', and 3
for 'asc' %%%%%

Derivative=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%
Which_Derivative=1;   %%% Enter 1 for 'First derivative'
and 2 for 'Second derivative' %%%%%

%%% Automated script %%%%%%%%%%

s=[];names=[];nome=[];group_names=[]; fstno =[1];
lastno=[0];

if Extension==1;
    Ext='csv';
else if Extension==2;
        Ext='txt';
    else if Extension==3;
            Ext='asc';
        end
    end
end

cd('C:\Users\Tanmoy\Desktop\Main Folder')
d1=dir;
for ii=3:length(d1);
      n1=d1(ii).name;
      cd(n1);
```

```
        d=dir(strcat('*.',Ext));
      for i=1:length(d);
            n=d(i).name;

            if Extension==1;
                  norm=csvread(n,skip_lines,0);
            else
                  norm=dlmread(n,'',skip_lines,0);
            end

            xaxis=norm(:,1);

            if Derivative~=0;
                        if Which_Derivative==1;
                        norm=diff(norm(:,2))/
                        diff(xaxis);
                        xaxis=xaxis(1:length(norm),:);
                        else if Which_Derivative==2;
                        norm=diff(norm(:,2),2)/
                        diff(xaxis,2);

                        xaxis=xaxis (1:length(norm),:);
                              end
                        end
            else
                  norm=norm(:,2);xaxis=xaxis;
            end

            p1=find(dataxd==1200); p2=find(dataxd==1800);
            yInterpolate=fd(p2:p1,:); xInterpolate=
            dataxd(p2:p1,:);
            Area=trapz(xInterpolate,yInterpolate);
            AreaS=Area/100;
            norm=yInterpolate/AreaS;
            s=[s norm];
            names=[names {n}];group_names=[group_names
            {n1}];
      end
      nome=[nome {n1}];
      amt=size (s); amt1=amt (:,2); lastno=[lastno amt1];

      fstno=[fstno lastno(:,ii-1)+1];

      cd('C:\Users\Tanmoy\Desktop\Main Folder')

end

[pc,score,latent]=princomp(s');...
```

Next, we add the same "on/off" option to select a specific spectral range as follows:

```
>> clear;clc;

%%% Change parameters here %%%%%%%%%%

Extension =3; %%% Enter 1 for 'csv', 2 for 'txt', and 3 for
'asc' %%%%%

Derivative=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%

Which_Derivative=1; %%% Enter 1 for 'First derivative'
and 2 for 'Second derivative' %%%%%

Interpolation=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%
 Interpolate_from=1200; Interpolate_to=1800; %%% Enter
range %%

%%% Automated script %%%%%%%%%%

s=[];names=[];nome=[];group_names=[]; fstno =[1];
lastno=[0];

if Extension==1;
    Ext='csv';
else if Extension==2;
        Ext='txt';
    else if Extension==3;
            Ext='asc';
        end
    end
end

cd('C:\Users\Tanmoy\Desktop\Main Folder')
d1=dir;
for ii=3:length(d1);
      n1=d1(ii).name;
      cd(n1);
       d=dir(strcat('*.',Ext));
      for i=1:length(d);
            n=d(i).name;

            if Extension==1;
                  norm=csvread(n,skip_lines,0);
            else
                  norm=dlmread(n,'',skip_lines,0);
            end

            xaxis=norm(:,1);

            if Derivative~=0;
```

```
                        if Which_Derivative==1;
                        norm=diff(norm(:,2))/diff(xaxis);
                        xaxis=xaxis(1:length(norm),:);
                        else if Which_Derivative==2;
                        norm=diff(norm(:,2),2)/
                        diff(xaxis,2);

                        xaxis=xaxis (1:length(norm),:);
                        end
                        end
                else
                     norm=norm(:,2);xaxis=xaxis;
                end

                if Interpolation~=0;

                    p1=find(xaxis==Interpolate_from);
                    p2=find(xaxis==Interpolate_to);
                    p3=p1-p2;
                          if p3<0;
                              norm=norm(p1:p2,:);xaxis=
                              xaxis(p1:p2);
                          else if p3>0;
                              norm=norm(p2:p1,:);xaxis=
                              xaxis(p2:p1);
                        end
                        end
                else
                    norm=norm; xaxis=xaxis;

                end

                Area=trapz(xInterpolate,yInterpolate);
                AreaS=Area/100;
                norm=yInterpolate/AreaS;
                s=[s norm];
                names=[names {n}];group_names=[group_names
                {n1}];
          end
          nome=[nome {n1}];
          amt =size (s); amt1 =amt (:,2); lastno =[lastno
          amt1];

          fstno=[fstno lastno(:,ii-1)+1];

          cd('C:\Users\Tanmoy\Desktop\Main Folder')
    end

    [pc,score,latent]=princomp(s');...
```

To achieve the same for area normalization:

```
>> clear;clc; %%% Change parameters here %%%%%%%%%%

Extension =3; %%% Enter 1 for 'csv', 2 for 'txt', and 3 for
'asc' %%%%%

Derivative=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%

Which_Derivative=1; %%% Enter 1 for 'First derivative'
and 2 for 'Second derivative' %%%%%

Interpolation=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%

Interpolate_from=1200; Interpolate_to=1800; %%% Enter
range %%

Area_Normalization=1; %%% Enter 0 for 'NO' and 1 for
'Yes' %%%%%

%%% Automated script %%%%%%%%%%

s=[];names=[];nome=[];group_names=[]; fstno =[1];
lastno=[0];
if Extension==1;
    Ext='csv';
else if Extension==2;
        Ext='txt';
    else if Extension==3;
            Ext='asc';
        end
    end
end

cd('C:\Users\Tanmoy\Desktop\Main Folder')
d1=dir;
for ii=3:length(d1);
     n1=d1(ii).name;
     cd(n1);
      d=dir(strcat('*.',Ext));
     for i=1:length(d);
          n=d(i).name;

          if Extension==1;
               norm=csvread(n,skip_lines,0);
          else
               norm=dlmread(n,'',skip_lines,0);
          end

          xaxis=norm(:,1);
```

```
        if Derivative~=0;
                    if Which_Derivative==1;
                    norm=diff(norm(:,2))/diff(xaxis);
                    xaxis=xaxis(1:length(norm),:);
                    else if Which_Derivative==2;
                    norm=diff(norm(:,2),2)/
                    diff(xaxis,2);
                    xaxis=xaxis (1:length(norm),:);
                    end
                    end
        else
             norm=norm(:,2);xaxis=xaxis;
        end
        if Interpolation~=0;
           p1=find(xaxis==Interpolate_from);
           p2=find(xaxis==Interpolate_to);
           p3=p1-p2;
                     if p3<0;
                        norm=norm(p1:p2,:);xaxis=
                        xaxis(p1:p2);
                     else if p3>0;
                        norm=norm(p2:p1,:);xaxis=
                        xaxis(p2:p1);
                    end
                    end
        else
             norm=norm; xaxis=xaxis;
        end
        if Area_Normalization~=0;
           Area=trapz(xaxis ,norm); AreaS=Area/100;
           norm=norm/AreaS;
     else;
          norm=norm;
     end
     s=[s norm];
       names=[names    {n}];group_names=[group_
       names {n1}];
end
nome=[nome {n1}];
amt=size (s); amt1=amt (:,2); lastno=[lastno amt1];
fstno=[fstno lastno(:,ii-1)+1];
```

```
        cd('C:\Users\Tanmoy\Desktop\Main Folder')
    end
    [pc,score,latent]=princomp(s');...
```

As mentioned earlier, "`norm`" is used throughout instead of different variable names as was used before. This is to avoid extra coding lines. For example, we start with wavenumbers and intensity values imported in variable "`data`." Then, we use the first derivative on it and call it "`fd.`" The command will be

```
fd=diff (data)
```

To select the spectral range, we have to command

```
interpolate=fd (p1:p2)
```

But consider if the user had skipped the derivatization step. There is no "`fd`;" thus, the command `fd (p1:p2)` will be unrecognized, and MATLAB will show an error. To account for this, we will have to have two separate commands:

`interpolate=fd (p1:p2)` when user applies derivatization

and

`interpolate=data (p1:p2)` when user skips derivatization.

The problem becomes more complicated with each step and condition, requiring more and more coding lines. By having the same variable name for all, the continuity is not broken.

The final option we will discuss is the ability to turn on/off the PCA plotting function. Sometimes, the user is just interested in LDA/LOOCV results and does not want PCA plots popping up at every run. We use the same logic as before to enable the user to choose whether or not PC plotting is desired:

```
>> clear;clc;%%% Change parameters here %%%%%%%%%%

nnnn='C:\Users\Tanmoy\Desktop\Main Folder';%%Enter
folder path%%

Extension =2; %% Enter 1 for 'csv', 2 for 'txt', and 3 for
'asc' %%%%%

skip_lines=0; %%% Enter number of lines to be skipped %%%%%

Derivative=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%

Which_Derivative=1; %%% Enter 1 for 'First derivative'
and 2 for 'Second derivative' %%%%%

Interpolation=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%
```

```
Interpolate_from=1200; Interpolate_to=1800; %%% Enter
range %%

Area_Normalization=1; %%% Enter 0 for 'NO' and 1 for
'Yes' %%%%%

Plot_PCA=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%

%%% Automated script %%%%%%%%%%

s=[];names=[];nome=[];group_names=[]; fstno =[1];
lastno=[0];

if Extension==1;
    Ext='csv';
else if Extension==2;
        Ext='txt';
  else if Extension==3;
          Ext='asc';
        end
  end
end

cd(nnnn)
d1=dir;
for ii=3:length(d1);
      n1=d1(ii).name;
      cd(n1);
       d=dir(strcat('*.',Ext));
      for i=1:length(d);
            n=d(i).name;

            if Extension==1;
                  norm=csvread(n,skip_lines,0);
            else
                  norm=dlmread(n,'',skip_lines,0);
            end

            xaxis=norm(:,1);

            if Derivative~=0;
                        if Which_Derivative==1;
                        norm=diff(norm(:,2))/
                        diff(xaxis);
                        xaxis=xaxis(1:length(norm),:);
                        else if Which_Derivative==2;
                        norm=diff(norm(:,2),2)/
                        diff(xaxis,2);
```

```
                    xaxis=xaxis (1:length(norm),:);
                    end
                    end
            else
                 norm=norm(:,2);xaxis=xaxis;
            end

            if Interpolation~=0;

               p1=find(xaxis==Interpolate_from);
               p2=find(xaxis==Interpolate_to);
               p3=p1-p2;
                         if p3<0;
                              norm=norm(p1:p2,:);
                              xaxis=
                              xaxis(p1:p2);
                         else if p3>0;
                              norm=norm(p2:p1,:);
                              xaxis=
                              xaxis(p2:p1);
                    end
                    end
            else
                 norm=norm; xaxis=xaxis;

            end

            if Area_Normalization~=0;
                Area=trapz(xaxis ,norm);
                AreaS=Area/100; norm=norm/AreaS;

           else;
                 norm=norm;
           end
           s=[s norm];

            names=[names    {n}];group_names=[group_
            names {n1}];
      end
      nome=[nome {n1}];
      amt=size (s); amt1=amt (:,2); lastno=[lastno amt1];

      fstno=[fstno lastno(:,ii-1)+1];

      cd(nnnn)
end
```

```
[pc,score,latent]=princomp(s');
contribution=cumsum (latent)./sum(latent);
PC_sig= contribution (1:10,:);
forplot=[0
    contribution];
plot(0:length(contribution),forplot*100);
save pc.mat pc; save score.mat score;
save contribution.mat contribution

y=group_names'; X=score(:,1:10);
Model =fitcdiscr(X,y);save   Model.mat;ldaClass
=resubPredict(Model);
[LDA,grpOrder]=confusionmat(y,ldaClass)
order=unique(y);
cp=cvpartition(y,'leaveout');
f=@(xtr,ytr,xte,yte)confusionmat(yte,...
classify(xte,xtr,ytr),'order',order);
cfMat2=crossval(f,X,y,'partition',cp);
LOOCV=reshape(sum(cfMat2),length(d1)-2,length(d1)-2)
    save LDA.mat
    save LOOCV.mat

if Plot_PCA~=0;

  szfst=size(fstno);
  szlst=size(lastno);
  start=fstno(:,1:szfst(:,2)-1);
  endm=lastno(:,2:szlst(:,2));

  C ={'k','b','r','g','y',[.5 .6 .7],[.8 .2 .6]};
  mkr={'o' 's' 'd' 'h'};
  aha=[1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6
  1 2 3 4 5 6];
  aha2=[1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 4 5 5 5 5 5
  6 6 6 6 6 6];

  for facsev=1:5;
     for facsev2=facsev+1:5
              figure();
            for sevi=1:length(start)
                div=sevi/6;
                Legend{sevi}=[{group_names(start
                (sevi))}];sz23=size(Legend);
```

```
scatter(score(start(sevi):endm(sevi),facsev),
score((start(sevi):endm(sevi)),facsev2),100,
C{aha(1,sevi)},mkr{aha2(1,sevi)},'filled');
                hold on
                T{facsev}=['PCA  Plot  factors',
                num2str(facsev),',',
                num2str(facsev2)];
            end
                for ijk=1:sz23(1,2);
                L{ijk}=[Legend{1,ijk}{1,1}{1,1}];
                end
        legend(L)
        title(T);
    end
   T=[];
  end
  FL1=[xaxis pc(:,1)];
  FL2=[xaxis pc(:,2)];
  FL3=[xaxis pc(:,3)];
  FL4=[xaxis pc(:,4)];
  FL5=[xaxis pc(:,5)];
  save FactorLoading1.txt FL1 -ascii
  save FactorLoading2.txt FL2 -ascii
  save FactorLoading3.txt FL3 -ascii
  save FactorLoading4.txt FL4 -ascii
  save FactorLoading5.txt FL5 -ascii
  else
      FL1=[xaxis pc(:,1)];
      FL2=[xaxis pc(:,2)];
      FL3=[xaxis pc(:,3)];
      FL4=[xaxis pc(:,4)];
      FL5=[xaxis pc(:,5)];
      save FactorLoading1.txt FL1 -ascii
      save FactorLoading2.txt FL2 -ascii
      save FactorLoading3.txt FL3 -ascii
      save FactorLoading4.txt FL4 -ascii
      save FactorLoading5.txt FL5 -ascii
  end
```

Note the script line with the gray background. This is to enable the user to enter the path right at the beginning. The line in the light blue background allows skipping lines. Also note that even if PCA is not plotted, the PCA loadings are saved.

Script I: Preprocess, PC-LDA, Plot PCA with ON/OFF features

```
clear;clc;%%% Change parameters here %%%%%%%%%%

nnnn='C:\Users\Tanmoy\Desktop\Main  Folder';%%
Enter folder path%%

Extension =2; %% Enter 1 for 'csv', 2 for 'txt', and 3
for 'asc' %%%%%

skip_lines=0; %%% Enter number of lines to be skipped
%%%%%

Derivative=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%

Which_Derivative=1; %%% Enter 1 for 'First deriva-
tive' and 2 for 'Second derivative' %%%%%

Interpolation=1; %%% Enter 0 for 'NO' and 1 for 'Yes'
%%%%%

Interpolate_from=1200; Interpolate_to=1800; %%%
Enter range %%

Area_Normalization=1; %%% Enter 0 for 'NO' and 1 for
'Yes' %%%%%

Plot_PCA=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%

%%% Automated script %%%%%%%%%%

s=[];names=[];nome=[];group_names=[]; fstno = [1];
lastno = [0];

if Extension==1;
    Ext='csv';
else if Extension==2;
        Ext='txt';
    else if Extension==3;
            Ext='asc';
        end
    end
end

cd(nnnn)
d1=dir;
for ii=3:length(d1);
      n1=d1(ii).name;
      cd(n1);
       d=dir(strcat('*.',Ext));
```

```
for i=1:length(d);
      n=d(i).name;
      if Extension==1;
            norm=csvread(n,skip_lines,0);
      else
            norm=dlmread(n,'',skip_lines,0);
      end
      xaxis=norm(:,1);
      if Derivative~=0;
            if Which_Derivative==1;
            norm=diff(norm(:,2))/diff(xaxis);
             xaxis=xaxis(1:length(norm),:);
      else if Which_Derivative==2;
                  norm=diff(norm(:,2),2)/
                  diff(xaxis,2);
                  xaxis=xaxis       (1:length
                  (norm),:);
                  end
                  end
      else
            norm=norm(:,2);xaxis=xaxis;
      end
      if Interpolation~=0;
         p1=find(xaxis==Interpolate_from);
         p2=find(xaxis==Interpolate_to);
         p3=p1-p2;
                    if p3<0;
        norm=norm(p1:p2,:);xaxis=xaxis(p1:p2);
                    else if p3>0;
        norm=norm(p2:p1,:);xaxis=xaxis(p2:p1);
                  end
                  end
      else
           norm=norm; xaxis=xaxis;
      end
      if Area_Normalization~=0;
         Area=trapz(xaxis ,norm); AreaS=Area/
         100;
         norm=norm/AreaS;
```

```
            else;
                    norm=norm;
            end
            s=[s norm];

             names=[names {n}];group_names=[group_
             names {n1}];
        end
        nome=[nome {n1}];
        amt =size (s); amt1 =amt (:,2); lastno =
        [lastno amt1];

        fstno=[fstno lastno(:,ii-1)+1];

        cd(nnnn)
end
[pc,score,latent]=princomp(s');
contribution=cumsum (latent)./sum(latent);
PC_sig= contribution (1:10,:);
forplot=[0
    contribution];
plot(0:length(contribution),forplot*100);
save pc.mat pc; save score.mat score;
save contribution.mat contribution

y=group_names'; X=score(:,1:10);
Model =fitcdiscr(X,y);save Model.mat;ldaClass
=resubPredict(Model);
[LDA,grpOrder]=confusionmat(y,ldaClass)
order=unique(y);
cp=cvpartition(y,'leaveout');
f=@(xtr,ytr,xte,yte)confusionmat(yte,...
classify(xte,xtr,ytr),'order',order);
cfMat2=crossval(f,X,y,'partition',cp);
LOOCV=reshape(sum(cfMat2),length(d1)-2,
length(d1)-2)
     save LDA.mat
     save LOOCV.mat

if Plot_PCA~=0;

    szfst=size(fstno);
    szlst=size(lastno);
    start=fstno(:,1:szfst(:,2)-1);
    endm=lastno(:,2:szlst(:,2));
```

```
C = {'k','b','r','g','y',[.5 .6 .7],[.8 .2 .6]};
mkr={'o' 's' 'd' 'h'};
aha=[1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4
5 6 1 2 3 4 5 6];
aha2=[1 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 4 4 4 4 4 5 5 5
5 5 5 6 6 6 6 6 6];

for facsev=1:5;
    for facsev2=facsev+1:5
            figure();
          for sevi=1:length(start)
              div=sevi/6;
              Legend{sevi}=[{group_names(start
              (sevi))}];sz23=size(Legend);

scatter(score(start(sevi):endm(sevi),facsev),
score((start(sevi):endm(sevi)),facsev2),100,
C{aha(1,sevi)},mkr{aha2(1,sevi)},'filled');
              hold on
              T{facsev}=['PCA Plot factors',
              num2str(facsev),',',num2str
              (facsev2)];
          end
              for ijk=1:sz23(1,2);
              L{ijk}=[Legend{1,ijk}{1,1}{1,1}];
              end
       legend(L)
       title(T);
    end
  T=[];
end
FL1=[xaxis pc(:,1)];
FL2=[xaxis pc(:,2)];
FL3=[xaxis pc(:,3)];
FL4=[xaxis pc(:,4)];
FL5=[xaxis pc(:,5)];
save FactorLoading1.txt FL1 -ascii
save FactorLoading2.txt FL2 -ascii
save FactorLoading3.txt FL3 -ascii
save FactorLoading4.txt FL4 -ascii
save FactorLoading5.txt FL5 -ascii
else
     FL1=[xaxis pc(:,1)];
     FL2=[xaxis pc(:,2)];
```

```
            FL3=[xaxis pc(:,3)];
            FL4=[xaxis pc(:,4)];
            FL5=[xaxis pc(:,5)];
            save FactorLoading1.txt FL1 -ascii
            save FactorLoading2.txt FL2 -ascii
            save FactorLoading3.txt FL3 -ascii
            save FactorLoading4.txt FL4 -ascii
            save FactorLoading5.txt FL5 -ascii
end
```

Since options have been incorporated, the same should be done in the script for the test prediction. Note that the information for turning on/off the features is stored in the "Model" obtained using the PC-LDA script and need not be repeated. The modified script is shown below:

Script II: Test Prediction with model built using Script I

```
clear; clc;
%%%%%% Importing model information%Enter path for
Model%%%%%%%%
nnnn1='C:\Users\Tanmoy\Desktop\Main  Folder';
cd(nnnn1);load('Model.mat');
%%%%%% Enter path for test spectra %%%%%%%%%%
nnnn='C:\Users\Tanmoy\Desktop\T';
%%%%%%% SCRIPT AUTOMATED HEREON %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%
cd(nnnn); d=dir(strcat('*.',Ext));s=[];nome=[];s1=[];
for i=1:length(d);
    n=d(i).name;
    nome=[nome {n}];
    if Ext=='csv';
        norm=csvread(n,skip_lines,0);
    else
        norm=dlmread(n,'',skip_lines,0);
    end
    xaxis=norm(:,1);
if Derivative~=0;
            if Which_Derivative==1;
            norm=diff(norm(:,2))/diff(xaxis);
            xaxis=xaxis(1:length(norm),:);
            else if Which_Derivative==2;
            norm=diff(norm(:,2),2)/diff(xaxis,2);
xaxis=xaxis (1:length(norm),:);
     end
```

```
    end
    else
              norm=norm(:,2);xaxis=xaxis;
          end
    if Interpolation~=0;
    p1=find(xaxis==Interpolate_from);
    p2=find(xaxis==Interpolate_to);p3=p1-p2;
       if p3<0;
              norm=norm(p1:p2,:);xaxis=xaxis(p1:p2);
          else if p3>0;
              norm=norm(p2:p1,:);xaxis=xaxis(p2:p1);
            end
        end
    else
       norm=norm; xaxis=xaxis;
    end
    if Area_Normalization~=0;
    Area=trapz(xaxis ,norm); AreaS=Area/100; norm=-
    norm/AreaS;
    else;
       norm=norm;
    end
    s=[s norm];
    end
    [pc, score]=princomp(s');
    %lda prediction
    y=nome';X=score(:,1:10);Pred=predict(Model,X);
    xx=[y Pred];save pred.mat
    xx
```

11 Note on MATLAB Functions

We have seen how to program on/off features. There is an easier way to do this, and many other operations, using MATLAB functions. MATLAB functions are separate scripts for performing specific tasks, which can be put together as a single line in the final script. Let us consider the following code:

```
>> clear;clc;

   %%% Enter 1 for 'csv', 2 for 'asc', and 3 for 'txt' %%%%%%%%%

   Extension =
```

```
%%% Automated script %%%%%%%%%%

s=[];names=[];nome=[];group_names=[]; fstno =[1];
lastno=[0];

if Extension==1;
    Ext='csv';
else if Extension==2;
        Ext='txt';
      else if Extension==3;
              Ext='asc';
        end
      end
end

cd('C:\Users\Tanmoy\Desktop\Main Folder')....
....
....
....
....
end

[pc,score,latent]=princomp(s');...
```

We can create a function for the yellow highlighted section in a new MATLAB script as follows:

```
function Ext=DecideExtension(Extension)

if Extension==1;
    Ext='csv';
else if Extension==2;
        Ext='txt';
  else if Extension==3;
          Ext='asc';
        end
  end
end
```

We then save it as DecideExtension.m in Desktop.

In the final script, the function can be called in a single line:

```
>> clear;clc;

%%% Enter 1 for 'csv', 2 for 'asc', and 3 for 'txt' %%%%%%%%%%

Extension =

%%% Automated script %%%%%%%%%%
```

```
s=[];names=[];nome=[];group_names=[]; fstno =[1];
lastno=[0];
cd('C:\Users\Tanmoy\Desktop')
[Ext]=DecideExtension(Extension)
cd('C:\Users\Tanmoy\Desktop\Main Folder').....
.....
.....
.....
.....
end
[pc,score,latent]=princomp(s');...
```

The advantage is that the final script is cleaner, less cluttered, and less confusing. Another advantage is that the same function can be used in different scripts without having to write it again and again.

It is important to note here that we have to first open the folder where the function file is saved. In this example, we have stored it on the Desktop. Hence, we first open the Desktop using "`cd('C:\Users\Tanmoy\Desktop')`," and then call the function "`[Ext] =DecideExtension(Extension)`." The next line in the script opens the folder where the spectral files are stored "`cd('C:\Users\Tanmoy\Desktop\Main Folder')`." In case there is no subsequent command line returning the script to the operational folder (in our case, the one containing the spectra), then it should be added after calling the function. This will ensure that the script is redirected for rest of the functions to the operational folder and also stores the results in the same. The function file should not be stored in the operational folder in the case of this script, as the script is designed to recognize folders and open them, and if the function file is in there, the script will try to open it, thinking it is a folder, will be unable to do so, and will return an error, stopping the script.

The same can be done for all preprocessing steps, other on/off features, PCA, PC-LDA, and plotting PCA, as well as prediction.

12 Final Note on How to Best Use the Script

It is clear that the script is folder dependent. One of the ways to carry out analysis is to change the folder path in

```
cd('C:\Users\Tanmoy\Desktop\Main Folder')
```

All user choices, PCA scores, loadings, PC significance, LDA, LOOCV, and preprocessing step information, as well as the LDA model, are stored in this folder. It is, therefore, imperative to use the same folder while predicting an unknown spectrum. Thus, the test prediction script needs two paths: one to

specify the location of the "Model," and the other to specify the location of test spectra:

```
%%%%% Importing model information% Enter path for Model
%%%%%%%%
nnnn1='C:\Users\Tanmoy\Desktop\Main Folder'; cd(nnnn1);
load('Model.mat');
%%%%% Enter path for test spectra %%%%%%%%%%
nnnn='C:\Users\Tanmoy\Desktop\T';
```

The specification of folder paths is critical for analysis, but it may be tedious to change the folder path each time. An alternative is to create folders for LDA and testing in a suitable location and copy the subfolders into them before analysis. Once analysis is done, all of the files along with results can be transferred to another folder labeled appropriately. Say one is testing "before treatment A" and "after treatment A" spectra. Subfolders with the same name can be made in the LDA folder and spectra can be transferred in them. The PC-LDA script can then be run. After the analysis, the subfolders, along with PCs, LDA model, LDA result, and LOOCV result, can be transferred from LDA to a folder labeled "treatment A – before after." The empty LDA folder can be used for another analysis. When spectra need to be predicted against a treatment A before-and-after PC-LDA model, all of the contents of folder "treatment A – before after" need to be copied into the LDA folder, the test spectra copied into the test folder, and then run the test prediction script. After prediction, both the LDA and test folders should be emptied before the next analysis. This way, any number of analyses can be performed without making changes in the script folder paths.

13 Common Errors

Errors are very common when running the script due to several reasons. One must check the following before running the script to avoid errors:

1. The LDA and test folders should contain no other file other than the spectra and they should be within proper subfolders. Any spectra or file out of a subfolder or files that do not match the other spectral files within the subfolder will cause error. For example, if one has a few files with spectrum information starting from line 1, and a few others starting from line 2, MATLAB will show an error.
2. It is important to specify lines to be skipped, as MATLAB will not read text lines and will show error.
3. Before giving a spectral range, check whether the spectrum has those wavenumbers. If MATLAB is asked to select a spectra range of 1200 to 1800 cm^{-1}, but the spectrum has wavenumbers 1199, 1201, 1203 cm^{-1}..., MATLAB will show an error. Also, it is important to remember that the first

derivatization reduces the spectral data by one. So, if there are spectra from 400 to 1800 cm^{-1} and the first derivatization is performed, the result will be an array with wavenumbers 400 to 1799 cm^{-1} or 1798 cm^{-1}, depending on the interval. In such a case, selecting 1200 to 1800 cm^{-1} will not work, and MATLAB will show an error.

4. If the extension of the file is other than asc, csv, or txt, MATLAB will show an error.
5. It is important to make sure that the folder/subfolder has spectra. If empty, MATLAB will show an error.
6. If the number of spectra per group are too few, MATLAB will show an error. It is recommended that each group has at least 10 spectra.
7. There must be a minimum of two subfolders for analysis. If one is exploring trends in a dataset with group information not known, it is best to divide the spectra randomly into two groups and visualize the data.

In case of error despite all precautions, contact the author. All datasets are different, and minor elements may give error unless script is fine-tuned according to the dataset.

14 Automating Mean and Standard Deviations Calculations: An Example

As mentioned earlier, the same script can be adapted for other purposes. In this section, we will briefly examine how to use the script for calculating mean and standard deviations.

Calculating mean and standard deviation is not a challenge and can be easily performed using Microsoft Excel or Origin. Then, why do we need MATLAB? The reason is to reduce tedium and manual errors. Imagine we have three groups "A," "B," and "C," each containing 30 spectra, and we need to find the mean spectrum and its standard deviation for each group. In this case, we have to copy/paste each spectrum to the Excel sheet, delete the wavenumbers column, and then put in the formula. It is possible that we may forget to copy all spectra or forget to delete a wavenumbers column, in which case the calculations will be erroneous. Origin allows one to import files at one go, but we still have to delete the wavenumber columns. Further, if the number of groups is large, the work becomes time consuming.

We can use MATLAB to automate the process. We have already learned to design a script that can import spectra from multiple subfolders and perform PCA and PC-LDA on them. Instead of PCA/PC-LDA analysis, we can ask MATLAB to calculate mean and standard deviations of the spectra imported from each subfolder.

Let us examine the first part of the script, excluding the parts that deal with PCA and PC-LDA:

```
clear;clc;%%% Change parameters here %%%%%%%%%%

nnnn='C:\Users\Tanmoy\Desktop\Main Folder';%%Enter
folder path%%

Extension =2; %% Enter 1 for 'csv', 2 for 'txt', and 3 for
'asc' %%%%%

skip_lines=0; %%% Enter number of lines to be skipped %%%%%

Derivative=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%

Which_Derivative=1; %%% Enter 1 for 'First derivative'
and 2 for 'Second derivative' %%%%%

Interpolation=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%

Interpolate_from=1200; Interpolate_to=1800; %%% Enter
range %%

Area_Normalization=1; %%% Enter 0 for 'NO' and 1 for
'Yes' %%%%%

Plot_PCA=1; %%% Enter 0 for 'NO' and 1 for 'Yes' %%%%%

%%% Automated script %%%%%%%%%%

s=[];names=[];nome=[];group_names=[]; fstno =[1];
lastno=[0];

if Extension==1;
    Ext='csv';
else if Extension==2;
        Ext='txt';
  else if Extension==3;
          Ext='asc';
        end
  end
end

cd(nnnn)
d1=dir;
for ii=3:length(d1);
      n1=d1(ii).name;
      cd(n1);
       d=dir(strcat('*.',Ext));
      for i=1:length(d);
            n=d(i).name;

            if Extension==1;
                  norm=csvread(n,skip_lines,0);
            else
```

```
        norm=dlmread(n,'',skip_lines,0);
    end

    xaxis=norm(:,1);

    if Derivative~=0;
            if Which_Derivative==1;
            norm=diff(norm(:,2))/diff(xaxis);
            xaxis=xaxis(1:length(norm),:);
            else if Which_Derivative==2;
            norm=diff(norm(:,2),2)/
            diff(xaxis,2);
            xaxis=xaxis (1:length(norm),:);
            end
            end
    else
        norm=norm(:,2);xaxis=xaxis;
    end

    if Interpolation~=0;

       p1=find(xaxis==Interpolate_from);
       p2=find(xaxis==Interpolate_to);
       p3=p1-p2;
            if p3<0;
                norm=norm(p1:p2,:);
                xaxis=
                xaxis(p1:p2);
            else if p3>0;
                norm=norm(p2:p1,:);
                xaxis=
                xaxis(p2:p1);
            end
            end
    else
        norm=norm; xaxis=xaxis;
    end

    if Area_Normalization~=0;
        Area=trapz(xaxis ,norm);
        AreaS=Area/100; norm=norm/AreaS;

else;
        norm=norm;

end
```

```
            s=[s norm];
                names=[names {n}];group_names=[group_
                names {n1}];
        end
        nome=[nome {n1}];
        amt=size(s);amt1=amt(:,2);lastno=[lastno amt1];
        fstno=[fstno lastno(:,ii-1)+1];
        cd(nnnn)
end
```

Note the parts of the script marked in yellow deal with preprocessing the spectrum or with PCA plotting, which we do not need to perform. So, we can simply delete those parts. Here is how it looks after that, with the places that need change highlighted in yellow:

```
clear;clc;%%% Change parameters here %%%%%%%%%%
nnnn='C:\Users\Tanmoy\Desktop\Main Folder';%%Enter
folder path%%
Extension =2; %% Enter 1 for 'csv', 2 for 'txt', and 3 for
'asc' %%%%%
skip_lines=0; %%% Enter number of lines to be skipped %%%%%
%%% Automated script %%%%%%%%%%
s=[];names=[];nome=[];group_names=[];
if Extension==1;
    Ext='csv';
else if Extension==2;
        Ext='txt';
    else if Extension==3;
            Ext='asc';
        end
    end
end
cd(nnnn)
d1=dir;
for ii=3:length(d1);
      n1=d1(ii).name; s=[];
      cd(n1);
       d=dir(strcat('*.',Ext));
```

```
        for i=1:length(d);
              n=d(i).name;
              if Extension==1;
                    norm=csvread(n,skip_lines,0);
              else
                    norm=dlmread(n,'',skip_lines,0);
              end
              xaxis=norm(:,1);
             s=[s norm(:,2)];
              names=[names   {n}];group_names=[group_
              names {n1}];
        end
        HERE
        nome=[nome {n1}];
        cd(nnnn)
end
```

We see there are three places that need changing—line 21 ("`s = []`"), line 35 ["`norm(:,2)`"], and after line 37 marked with "here" written in large font with yellow highlight. We will discuss the line 21 change last. The minor change in line 35 from "`norm`" to "`norm(:,2)`" makes sure that only intensity values are stored in the variable instead of both wavenumbers and intensity values. This is because we calculate the mean of intensity values, while the wavenumber values remain unchanged.

The major changes are made after line 37 marked by "here." Continuing with the example given before, we need to calculate the mean and standard deviations for spectra in each group, "A," "B," and "C." For this reason, we have saved all spectra in each group in subfolders "A," "B," and "C," inside the "main folder" as before. The script until line 37 opens the "main folder," then opens the first subfolder "A," and gathers all the intensity values in the variable "`s`." The next step is to calculate the mean.

We can, therefore, insert the MATLAB command for calculating the average:

```
y=mean((s'));
```

We use the transpose function (`'`), because MATLAB calculates mean for rows, and we have spectra arranged in columns, where one column = 1 spectrum. By transposing, we make 1 row = 1 spectrum. Then, the "mean" function can be used. We then store the calculated mean in variable "`y`" and transpose it again for convenience and store it in variable "`y1`":

```
y1=y';
```

We calculate the standard deviation again after transposing the variable "s," then transpose the result in "z1," and store it in "z":

```
z=std((s'));z1=z';
```

Next, we calculate "mean + standard deviation" and "mean − standard deviation" as follows:

```
s1=y1 + z1; s2=y1-z1;
```

We then plot the results as follows (recall that the wavenumbers are stored in the variable "xaxis"):

```
figure();
plot(xaxis, y1, '-r'); hold on
plot(xaxis, s1, '-b'); hold on
plot(xaxis, s2, '-g'); hold on
```

and save them as

```
save(strcat(n1,'mean'),'y1','-ascii');
save(strcat('stp',n1),'s1','-ascii');
save(strcat('ntp',n1),'s2','-ascii');
```

Recall that the variable "n1" contains the name of the subfolder, that is the group name, and that the command "strcat" will join the text within the brackets. So, if "n1" is "A," "`strcat(n1,'mean')`" will result in "Amean," which will help identify the saved file.

We keep

```
cd(nnnn)
```

While these calculations are going on, the script is in the first subfolder "A." The above command will return the script to the "main folder" then open the next sub-folder "B," and so on.

The final and very important step is to add "`s=[]`" after the beginning of the first "`for`" loop. This is the change that is made in line 21. This ensures that the variable "s" is empty to receive spectra from the next subfolder, say, "B." If this is not done, spectra from the previous subfolder, say, "A," will remain in "s," and when spectra from "B" subfolder get added, the calculated mean will be the mean of "A" and "B," and not just "B." We need the mean of each folder separately, the mean of "A," mean of "B," and so on. So, this step is critical.

With this script, the mean spectrum with the spectra mean + standard deviation and mean − standard deviations will be calculated for each subfolder and saved within the subfolder. Plots of the same will pop up as figures for each subfolder, too, which can help a quick visualization of the mean and standard deviations of each group. The script does not do any preprocessing, but most spectroscopists prefer to preprocess all the spectra before calculating the mean.

The user can choose to use preprocessed spectra as input or write a script for automated preprocessing and append it before the above script.

All precautions outlined in Section 13 apply for this script, too.

References

1. E. Hanlon et al., "Prospects for *in vivo* Raman spectroscopy," *Phys. Med. Biol.* **45**(2), R1 (2000).
2. Q. Tu and C. Chang, "Diagnostic applications of Raman spectroscopy," *Nanomed.: Nanotechnol. Biol. Med.* **8**(5), 545–558 (2012).
3. I. Pence and A. Mahadevan-Jansen, "Clinical instrumentation and applications of Raman spectroscopy," *Chem. Soc. Rev.* **45**(7), 1958–1979 (2016).
4. R. Petry, M. Schmitt, and J. Popp, "Raman spectroscopy—a prospective tool in the life sciences," *ChemPhysChem* **4**(1), 14–30 (2003).
5. B. Singh et al., "Application of vibrational microspectroscopy to biology and medicine," *Curr. Sci.* **102**(2), 232–244 (2012), http://eprints.iisc.ac.in/id/eprint/43712 and https://pdfs.semanticscholar.org/5e51/be4f6fadf16c8a211e3dafe761cdcd245f99.pdf.
6. D. Cialla-May et al., "Recent progress in surface-enhanced Raman spectroscopy for biological and biomedical applications: from cells to clinics," *Chem. Soc. Rev.* **46**(13), 3945–3961 (2017).
7. Z. Zhao et al., "Applications of vibrational tags in biological imaging by Raman microscopy," *Analyst* **142**(21), 4018–4029 (2017).
8. H. Ahn et al., "Emerging optical spectroscopy techniques for biomedical applications—a brief review of recent progress," *Appl. Spectrosc. Rev.* **53**(2–4), 264–278 (2018).
9. C. Krafft et al., "Disease recognition by infrared and Raman spectroscopy," *J. Biophotonics* **2**(1–2), 13–28 (2009).
10. P. K. Hopke, "The evolution of chemometrics," *Anal. Chim. Acta* **500**(1–2), 365–377 (2003).
11. Å. Rinnan, "Pre-processing in vibrational spectroscopy-when, why and how," *Anal. Methods* **6**(18), 7124–7129 (2014).
12. R. Bro and A. K. Smilde, "Principal component analysis," *Anal. Methods* **6**(9), 2812–2831 (2014).
13. D. Ballabio and V. Consonni, "Classification tools in chemistry. Part 1: linear models. PLS-DA," *Anal. Methods* **5**(16), 3790–3798 (2013).
14. R. Gautam et al., "Review of multidimensional data processing approaches for Raman and infrared spectroscopy," *EPJ Tech. Instrum.* **2**(1), 1–38 (2015).

Tanmoy Bhattacharjee received his M.S. (Biotechnology) from the University of Mumbai (2007) and Ph.D. (Life Science) from the Homi Bhabha National Institute, Mumbai (2015). He was a post-doctoral fellow at the University of Paraiba Valley, Brazil (2016-2017), and currently works as a post-doctoral researcher at the University of Otago, New Zealand. He has 22 publications and has co-authored a book on Raman spectroscopy. His research interests include the development of diagnostic devices, animal models of diseases, biological applications of spectroscopy and ultrasound techniques, and writing MATLAB- and R-based scripts for data analysis.