GSTools version 0.4.2

A set of matlab functions to read, write and deal with .spc spectra using MATLAB®...

By Kris De Gussem
Research assistant of the Research Foundation Flanders
Ghent University
Department of analytical chemistry
Proeftuinstraat 86
B-9000 Ghent

web: http://www.AnalChem.UGent.be/Raman/
e-mail: Kris.DeGussem @ gmail.com

Table of contents

1) Introduction

GSTools is a set of free matlab functions, released under the *GPL 3.0* license. The GPL license permits you to install and use this toolbox on all of your computers and to view and modify its source code.

This toolbox is a first implementation to open and save *SPC-spectra* in matlab. GSDendrogram is supplied as well. It is to be used as a supplement to the matlab's toolbox function Dendrogram. GSDendrogram therefore allows the user to put coloured labels with the samples' names under the dendrogram. In addition some other functions which are needed for these functions to work properly are added. Last but not least, the BTree class is supplied as well, which offers the user to easily inspect the different items in a list, as well as it returns e.g. the positions of these different items. This combined approach makes it easy to work with numeric arrays and stringlists, such as spectral descriptions.

In this document you will find information on how to install this toolbox. Thereafter you will find the information on how to use the main functions. Finally the BTree object is explained in detail.

2) Installation

Unzip the original distribution file into a subdir in your matlab work directory. In matlab, choose "Set Path" > "Add with Subfolders..." > Select your directory of the toolbox > Choose "Yes" followed by "Save".

3) GSSpcRead

GSSpcRead will import a spectrum in the spc-file format into the matlab-environment. The universal design of the spc-file format permits the use of spectra from different spectroscopic sources containing a wide range of spectra and information. Currently, GSSpcRead is able to import (most if not all) single-spectra spc-files. Multi-spectra spc-files, such as chromatograms, aren't supported.

GSSpcRead aims to implement all properties of the SPC-file format. As such it returns all spectral information in the SPC-file. A parameter is supplied to provide compatibility with PLSToolbox's SPC-file import function.

Syntax

Specdata = GSSpcRead (filename, compatibility, plotspec, YScaling)

Input parameters

- filename: the filename of the spectrum in the spc-format, path included
- compatibility: (optional) a numerical parameter used for compatibility with other functions
    - if not given or -1: no compatibility, leave output as it is
    - if 0: maintains compatibility with the PLS toolbox' spcreadr function. The data are returned in an array: using the outputparameters: [data,xaxis,auditlog] with:
        - data: one matrix containing all spectra
        - xaxis: the axis to which the spectra in the spc-file correspond
        - auditlog: log of the file (ordinary text in an array)
- plotspec: (optional, default = true): boolean: indicates whether the spectrum has to be plot when it is loaded
- YScaling: (optional) headers in some SPC-files are not consistent, which results in improper Y-scaling. If scaling is incorrect, you can tell GSSpcRead which exponent to use: if YScaling is:
    - 'fexp': File exponent (exponent in SPC-header) is used
    - 'subexp': sub-file exponent (exponent in sub-header) is used

    Only use this parameter in case of problems. This parameter may be removed in a future version, so if used, use it with care.
- VerboseOutput: (optional: default false): boolean: indicates whether detailed information needs to be printed in the command window. If this parameter is set to false/0, then only error messages are given

Output parameters

- Specdata: structure containing the spectrum/spectra in the spc-file, with full spectral info, including spectrometer info, instrumental technique, date-time, units of axis...

Sample code

The following example will load the spectrum “C:\test.spc” and plot in on the screen including the axis labels. Finally the information logged by the spectrometer is printed in the command window.

```
SData = GSSpcRead ('c:\test.spc'); %Read the spectrum
plot (SData.xaxis, data); %plot the spectrum
h = title (Sdata.Name);
set (h, 'FontWeight', 'bold');
set (h, 'FontSize', 16);
xlabel (SData.xtype);
ylabel (SData.ytype);
SData.log.txt   %show spectral info
```

4) GSImportSpec

GSImportspec uses GSSpcRead to import all spc-files in a folder. All files are collected in a single structure array containing all spectra, including all spectral information.

Syntax

spectra = GSImportSpec (dirname, compatibility);
spectra = GSimportspec (dirname);

Input parameters

- dirname: optional string: name of map which contains the spectra
- compatibility: optional (default -1): numerical value: if:
    - 0: function is compatible with PLS-toolbox output
    -1 (default value) or anything else: standard GSSpcRead output

Output parameters

- spectra: structure array: spectra with all spectral information: if input parameter compatibility was 0:
    - structure array contains fields:
        - name: file name
        - spectrum: vector of spectral values
        - xaxis: vector of xaxis-values
        - auditlog: log of file
    - each structure has standard GSSpcRead format

Sample Code

```
spectra = GSimportspec ('c:\testset\', 0);
spectra2 = cat (1, spectra.data); %convert to data matrix
plot (spectra(1).xaxis, spectra2);
```

5) GSSpcWrite

GSSpcWrite will export a spectrum from the matlab-environment into an spc-file using a vector. GSSpcWrite is currently able to export single-spectra spc-files.

Syntax

GSSpcWrite (filename, spectra, axis, logtxt, logbin, options)

Input parameters

- filename: the name of the file in which the spectrum will be written, path included
- spectra: (double array) the spectral data itself. Spectral data are in a matrix, where one row represents one spectrum.
- axis: (cell array), the x-axis of the spectra.
- logtxt: (char array) logtext to save with the spectra
- logbin: (uint8 data) binary data to save with the data
- options: optional parameter: structure: possible fields:
    - fcmnt: string: file comment
    - fexper: integer: experimental technique: see SPC file format description (0 = general)
    - fsource: string: instrument source
    - fmethod: string: used methods, instrumental techniques, ...
    - xunit: integer: type of x-units: see SPC file format description (0 = general) or call the function GetSPCAxisTypes
    - yunit: integer: type of y-units: see SPC file format description (0 = general) or call the function GetSPCAxisTypes

6) GSDendrogram

GSDendrogram builds a dendrogram using the statistical toolbox's dendrogram function. However, in case of spectral numbers, labels are used to indicate different groups of samples. There is a new labelling method, in which coloured bars below the dendrogram are plot, indicating different groups of spectra, as well as the amount of spectra in the different groups. Information on the linkage of leafs and samples is printed in the command window.

Syntax

Info = GSDendrogram (Z, Labels, Colors)

Input parameters

- Z: The Z-matrix as normal input for dendrogram (see also *help dendrogram*)
- Labels: Labels of the different spectra
- Colors: a optional colour matrix, indicating the individual colours for the different sample groups

Output parameters

- Info: a matrix containing dendrogram info about the spectra

Sample Code

```
%calculate euclidean distances and use Ward's clustering method, with X = datamatrix
Y = pdist(X,'euclidean');
Z = linkage(Y,'ward');

%show the cophenet value
cophenet(Z,Y)

%Show the dendrogram
Info = GSDendrogram (Z, SampleNames);
h = title ('Spectra: Ward''s clustering method using Euclidean distance');
set (h, 'FontWeight', 'bold')
set (h, 'FontSize', 14)
```

7) BTree: a binary tree implementation

A binary tree is an ordered data structure. In such a tree data is stored alphabetically. In other cases, this data structure can be used to store information (such as tasks to perform) keeping priority in mind. In a binary tree each item, which is called a leaf, can have only two subitems, which are referred to as left and right branch. In the left branch items are stored which are less important, or have a lower (alphabetical) order than the current item; while in the right branch items are stored which are more important (have a higher (alphabetical) order). Often, in a binary tree, extra information can be stored in a leaf, such as the amount an item is found elsewhere.

The binary tree is implemented as a Matlab class or object. This implementation of a binary tree stores strings as well as numbers. Both data can be used together in the same BTree object. Because the data are ordered, items are mostly found in no-time. When converting a list of strings or numbers, the amount an item is found in the list as well as the positions in the original list are stored together with the item. In this way a Btree object permits one to easily obtain the indices of an item in a list. As such a BTree object is an other representation of the original list.

Primary uses

Primary uses of the BTree object are:

1) obtaining the different values in a list
2) checking the presence of an item
3) obtaining the positions of an item in a list
4) additionally a list could be sorted

Results obtained by using the BTree class can also be obtained by using other matlab functions. The BTree object is to be preferred when using a lot of data, when the positions of multiple items in a list are needed and when multiple things in the previous list need to be

done. Innovative use of the BTree class is found in the GSDendrogram function to obtain the number of different groups of samples and colour the labels accordingly. The performance of a BTree class ideally (and most of the time it performs ideally) is independent of the order of the original data which have been put in the binary tree. The performance of most search algorithms, however, is influenced by this order. Because the way in which the data are structured, less comparisons and less search time is needed, compared to a simple a line per line search. The advantages of a binary tree are much more pronounced when more items are added to the tree.

Creation of a BTree class

To create a class you can simply call the BTree command:

tree = BTree (X, description)
tree = BTree

Where:

- X is an original stringlist (a 2D char array, or 1D cellstring array): the stringlist is converted to a BTree object
- description: an optional description of the data in the BTree object
- tree: the created BTree object

Functions in the BTree class

In what follows a short list is presented of the functions used to manipulate a BTree object. You can obtain extensive help by calling het command "*help BTree/FunctionName*" where *FunctionName* is the function name found in the list below:

- Add: add an item (string or number) to the binary tree
- BTree: initiate a BTree
- Count: returns the number of different items in the binary tree
- GetItem: returns the information on an item, including in which order items are added to the list or the positions of the item in the list which was used to create the BTree using the BTree command
- GetItems: returns a list of all items in the binary tree
- GetItemWithMostValues: searches for the item that has the most values linked to the item (in most cases these values are the positions in a list, thus this function seeks for the item that appears the most in the list).
- GetSortedItemsList: returns a sorted list of the items in the binary tree; you'll find an item replicated x times, when it was added x times to the binary tree. As such GetSortedItemsList can be effectively used to sort a stringlist
- IsItem: simply checks whether an item exists in a tree
- Plot: gives an ASCII representation of the binary tree.
- WriteContentsToFile: saves the content of the tree to a file.

Sample code

```
%Create some data
list1 = [{'first'} {'first'} {'first'} {'second'} {'second'} {'second'} {'second'} {'third'}
{'third'}]'
list2 = shuffle (list1)
tree = BTree (list2) %create the binary tree

%check the presence of a string in the binary tree
isitem (tree, 'fourth')
isitem (tree, 'second')
Items = GetItems (tree);
Items{1} %obtain information on first item (which is 'first')
pos = cat(1, Items{1}.itemvalues{:}) %positions of first item in the list

SortedList = GetSortedItemsList (tree) %sort the original list2, and list1 is obtained
```