Import and Export of Spectra Files Vignette for the R package hyperSpec

Claudia Beleites <cbeleites@units.it>
CENMAT, DMRN, University of Trieste

March 10, 2010

Supported File Formats

hyperSpec supports a number of file formats relevant for different types of spectroscopy. This is naturally only a subset of the file formats produced by different spectroscopic equipment.
If you use *hyperSpec* with data formats not mentioned in this document, please send an email to ClaudiaBeleites<cbeleites@units.it>, so that this document can be updated.
The information should include

- The type of spectroscopy
- Spectrometer model, manufacturer, and software
- The "native" file format (including a sample file)
- Description of relevant procedures to convert the file
- R code to import the data together with an example file that can actually be read by R.
- Documentation, particularly the description of the data format

If you need help finding out how to import your data, *hyperSpec* has a mailing list hyperspec-help@lists.r-forge.r-project.org, subscription and archives are available at http://r-forge.r-project.org/mail/?group_id=366.

Reproducing the Examples in this Vignette

The source code of this vignette including the spectra files are available as .zip file at *hyperSpec*'s home page: http://hyperspec.r-forge.r-project.org/FileIO.zip
Note that some definitions are in file `vignettes.defs`.

Contents

1 Introduction

This document describes how spectra can be imported into *hyperSpec* objects. Some possibilities to export *hyperSpec* objects as files are mentioned, too.

The most basic funtion to create *hyperSpec* objects is `new ("hyperSpec")` (section 2). It makes a *hyperSpec* object from data already in R's workspace. Thus, once the spectra are imported into R, conversion to *hyperSpec* objects is straightforward.

In addition, *hyperSpec* comes with predifined import functions for different data formats. This document divides the discussion into dealing with ASCII files (section 3, p. 3) and binary file formats (section 4, p. 4). If data export for the respective format is possible, it is discussed in the same sections. As sometimes the actual data written by the spectrometer software exhibits peculiarities, *hyperSpec* offers several specialized import functions. These are in general named after the data format followed by the manufacturer (e. g. read.ENVI.Nicolet).

Overview lists of the directly supported file formats are in the appendix: sorted by file format (appendix **??**, p. **??**), manufacturer (appendix **??**, p. **??**), and by spectroscopy (appendix **??**, p. **??**).

2 Creating a hyperSpec object with `new`

To create a *hyperSpec* object from data in R's workspace, use:

```
spc <- new ("hyperSpec", spc, wavelength, data, label)}
```

With the arguments:

`spc`	the spectra matrix (may also be given as matrix inside column `$spc` of `data`)
`wavelength`	the wavelength axis vector
`data`	the extra data (possibly already including the spectra matrix in column `spc`)
`label`	a list with the proper labels. Do not forget the wavelength axis label in `$.wavelength` and the spectral intensity axis label in `$spc`.

Thus, once your data is in R's workspace, creating a *hyperSpec* object is easy. I suggest wrapping the code to import your data and the line joining it into a *hyperSpec* object by your own import function. You are more than welcome to contribute such import code to *hyperSpec*. Secion **??**, (p. **??**) discusses examples of custom import functions.

3 ASCII files

Currently, *hyperSpec* provides two functions for general ASCII data import:

`read.txt.long`	imports long format ASCII files, i. e. one intensity value per row
`read.txt.wide`	imports wide format ASCII files, i. e. one spectrum per row

The import functions immediately return a *hyperSpec* object.

Internally, they use `read.table`, a very powerful ASCII import function. R supplies another ASCII import function, `scan`. `scan` imports numeric data matrices and is faster than `read.table`, but cannot import column names. If your data does not contain a header or it is not important and can safely be skipped, you may want to import your data using `scan`.

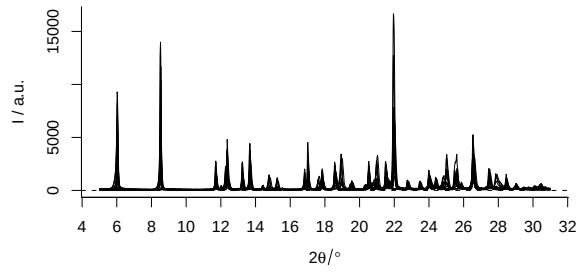
3.1 ASCII files with samples in columns

Richard Pena asked about importing another ASCII file type:

> *Triazine5_31.txt file corresponds to X ray powder diffraction data (Bruker AXS). The native files data ".raw" are read with EVA software then they are converted into .uxd file with the File Exchange software (Bruker AXS). The .uxd file are opened with Excel software and saved as .txt file, csv file (ChemoSpec) or xls.*
> *The first and following columns corresponds to the angle diffraction and the intensity values of samples respectively.*

This file thus differs from the ASCII formats discussed above in that the samples are actually in columns whereas *hyperSpec* expects them to be in rows. The header line gives the name of the sample. Import is straightforward, just the spectra matrix needs to be transposed to make a *hyperSpec* object:

```
> file <- read.table ("txt.t/Triazine 5_31.txt", header = TRUE, dec = ",", sep = "\t")
> triazine <- new ("hyperSpec", wavelength = file [,1], spc = t (file [, -1]),
+                  data = data.frame (sample = colnames (file [, -1])),
+                  label = list (.wavelength = expression (2 * theta / degree),
+                                spc = "I / a.u."))
> triazine

hyperSpec object
   25 spectra
   2 data columns
   1759 data points / spectrum
wavelength: 2 * theta/degree [numeric] 5.0025 5.0173 ... 31.0042
data:  (25 rows x 2 columns)
   1. sample:  [factor] rng  DIV1208200 DIV1208300 ... VCA0106703
   2. spc: I / a.u. [AsIs matrix x 1759] rng  16 18 ... 16664


> plot (triazine)
```



3.2 JCAMP-DX

JCAMP-DX files[**?**] are not supported as there has not yet been the need to import them.

3.3 ASCII Export

ASCII export can be done in wide and long format using `write.txt.long` and `write.txt.wide`. If you need a specific header or footer, use R's functions for writing files: `write.table`, `write`, `cat` and so on offer fine-grained control of writing ASCII files.

4 Binary file formats

4.1 Matlab Files

Matlab files can be read and written using the package *R.matlab*[**?**], which is available at CRAN and can be installed by `install.packages ("R.matlab")`.

```
spc.mat <- readMat ("spectra.mat")
```

If the .mat file was saved with compression, the additional package *Rcompression* is needed. It can be installed from omegahat:

```
install.packages("Rcompression", repos = "http://www.omegahat.org/R")
```

See the documentation of *R.matlab* for more details and possibly needed further packages.

`readMat` imports the .mat file's contents as a list. The variables in the .mat file are properly named elements of the list. The *hyperSpec* object can be created using `new`, see 2 (p. 3).

Again, you probably want to wrap the import of your matlab files into a function.

4.1.1 Matlab Export

R.matlab's function `writeMat` can be used to write R objects into .mat files. To save an *hyperSpec* object `x` for use in Matlab, you most likely want to save:

- the wavelength axis as obtained by `wl (x)`,
- the spectra matrix as obtained by `x [[]]`, and
- possibly also the extra data as obtained by `x$..`
- as well as the axis labels `labels (x)`.
- Alternatively, `x$.` yields the extra data together with the spectra matrix.

However, it may be convenient to transform the saved data according to how it is needed in Matlab. The functions `as.long.df` and `as.wide.df` may prove useful for reshaping the data.

4.2 ENVI Files

ENVI files are binary data accompanied by an ASCII header file. *hyperSpec*'s function `read.ENVI` can be used to import them.

As we experienced missing header files (Bruker's Opus software frequently produced header files without any content), the data that would usually be read from the header file can also be handed to `read.ENVI` as a list. The help page gives details on what elements the list should contain, see also the discussion of ENVI files written by Bruker's OPUS software (section **??**, p. **??**).

Here is a demonstration of the use of `read.ENVI`:

```
> spc <- read.ENVI ("ENVI/example2.img")

.read.ENVI.header: Guessing header file name (ENVI/example2.hdr)
.read.ENVI.bin: 'byte order' not given or incorrect. Guessing 'little'
```

```
> spc

hyperSpec object
   420 spectra
   3 data columns
   1738 data points / spectrum
wavelength:  [numeric] 649.90 651.83 ... 3999.7
data:  (420 rows x 3 columns)
   1. x:  [integer] rng  0 1 ... 13
   2. y:  [integer] rng  0 1 ... 29
   3. spc:  [AsIs matrix x 1738] rng  0
```

4.2.1 ENVI Export

Use package *caTools* or *rgdal* with GDAL for writing ENVI files.

4.3 spc Files

Thermo Galactic's .spc file format[**?**] can be imported by `read.spc`.

A variety of sub-formats exists. *hyperSpec*'s import`read.spc` function does *not* support the old file format that was used before 1996. In addition, no test data with *w planes* was available — thus the import of such files could not be tested. If you come across such files, please contact the package maintainer (ClaudiaBeleites<cbeleites@units.it>).

Here are some tests using Thermo Galactic's example files:

```
> ## old format files stop with an error:
> old <- paste ("spc", c ('CONTOUR.SPC', 'DEMO 3D.SPC', 'LC DIODE ARRAY.SPC'), sep = "/")
> for (f in old)
+   try (read.spc (f))
> ## all other files should be good for import
> other <- setdiff (Sys.glob ("spc/*.[sS][pP][cC]"), old)
> for (f in other){
+   spc <- read.spc (f)
+
+   if (is (spc, "hyperSpec"))
+     cat (f, ": ", nrow (spc), " spectrum(a), ", nwl (spc), " data pts / spc.\n", sep = "")
+   else
+     cat (f, ": list of ", length (spc), " spectra, ",
+          paste (range (sapply (spc, nwl)), collapse = " - "),
+          " data pts / spc\n", sep = "")
+ }

spc/BARBITUATES.SPC: list of 286 spectra, 4 - 101 data pts / spc
spc/barbsvd.spc: list of 286 spectra, 4 - 101 data pts / spc
spc/BENZENE.SPC: 1 spectrum(a), 1842 data pts / spc.
spc/DRUG SAMPLE_PEAKS.SPC: list of 6 spectra, 80 - 253 data pts / spc
spc/DRUG SAMPLE.SPC: list of 400 spectra, 2 - 254 data pts / spc
spc/FID.SPC: 1 spectrum(a), 8192 data pts / spc.
spc/HCL.SPC: 1 spectrum(a), 8361 data pts / spc.
spc/HOLMIUM.SPC: 1 spectrum(a), 901 data pts / spc.
spc/IG_BKGND.SPC: 1 spectrum(a), 4096 data pts / spc.
spc/IG_MULTI.SPC: 10 spectrum(a), 4096 data pts / spc.
```

```
spc/IG_SAMP.SPC: 1 spectrum(a), 4645 data pts / spc.
spc/KKSAM.SPC: 1 spectrum(a), 751 data pts / spc.
spc/POLYR.SPC: 1 spectrum(a), 1844 data pts / spc.
spc/POLYS.SPC: 1 spectrum(a), 1844 data pts / spc.
spc/SINGLE POLYMER FILM.SPC: 1 spectrum(a), 1844 data pts / spc.
spc/SPECTRUM WITH BAD BASELINE.SPC: 1 spectrum(a), 1400 data pts / spc.
spc/TOLUENE.SPC: 1 spectrum(a), 801 data pts / spc.
spc/TUMIX.SPC: 1 spectrum(a), 1775 data pts / spc.
spc/TWO POLYMER FILMS.SPC: 1 spectrum(a), 1844 data pts / spc.
spc/XYTRACE.SPC: 1 spectrum(a), 3469 data pts / spc.
```

The header and subheader blocks of spc files store additional information of pre-defined types (see the file format specification[**?**]). Further information can be stored in the so-called log block at the end of the file, and should be in a key-value format (although even the official example files do not always). This information is often useful (Kaiser's Hologram software e. g. stores the stage position in the log block).

`read.spc` has four arguments that allow fine-grained control of storing such information in the *hyperSpec* object:

`keys.hdr2data` parameters from the spc file and subfile headers that should become extra data columns

`keys.hdr2log` parameters from the spc file and subfile headers that should be stored as list entries in the *long.description* of the log entry

`keys.log2data` parameters from the spc file log block that should become extra data columns

`keys.log2log` parameters from the spc file log block that should be stored as list entries in the *long.description* of the log entry

The value of these arguments can either be logical (amounting to either use all or none of the information in the file) or a character vector giving the names of the parameters that should be used. Note that the header file field names are always lowercase, while the log entries are treated case sensitive.

.spc files may contain multiple spectra that do *not* share a common wavelength axis. In this case, `read.spc` returns a list of *hyperSpec* objects with one spectrum each. `rbind.fill` may be used to combine this list into one *hyperSpec* object:

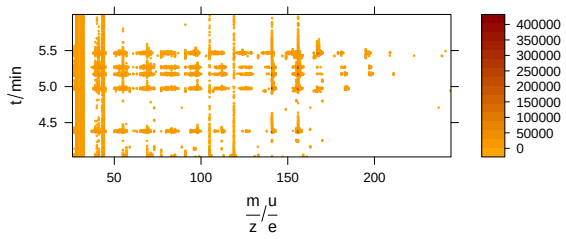
```
> spc <- read.spc ("spc/BARBITUATES.SPC")
> class (spc)

[1] "list"

> length (spc)

[1] 286

> spc <- do.call (rbind.fill.hyperSpec, spc)
> spc

hyperSpec object
   286 spectra
   4 data columns
   375 data points / spectrum
wavelength: frac(m, z)/frac(u, e) [numeric] 25.95 26.05 ... 244.05
data:  (286 rows x 4 columns)
   1. z: t/min [numeric] rng  4.027184 4.034117 ... 5.997766
```

```
   2. z.end: t/min [numeric] rng  4.027184 4.034117 ... 5.997766
   3. .object:  [integer] rng  1 2 ... 286
   4. spc: I/"a. u." [AsIs matrix x 375] rng  510 512 ... 403968 + NA
```

```
> spc [[1:10, , 25 ~ 30]]
```

```
      25.95 26.05 26.15 26.95 27.05 27.15 28.05 28.15 29.05 29.15 29.95
 [1,]    NA    NA    NA    NA   562    NA    NA 11511  6146    NA    NA
 [2,]    NA    NA    NA    NA    NA   618 10151    NA  5040    NA    NA
 [3,]    NA    NA    NA    NA   638    NA    NA 10722  5253    NA    NA
 [4,]    NA    NA    NA    NA    NA    NA 10548    NA  5865    NA    NA
 [5,]    NA    NA    NA    NA    NA    NA    NA 10519  4664    NA    NA
 [6,]    NA    NA    NA   796    NA    NA 10519    NA  5110    NA    NA
 [7,]    NA    NA    NA    NA    NA    NA 10096    NA  4769    NA   907
 [8,]    NA    NA    NA    NA    NA    NA    NA 10929  5400    NA    NA
 [9,]    NA    NA    NA    NA    NA    NA 10235    NA  4930    NA    NA
[10,]    NA    NA    NA    NA    NA    NA    NA 10663  4690    NA   799
```

```
>  levelplot (spc ~ .wavelength * z, spc, panel = panel.leveldotplot, cex = .3,
+             col.regions = colorRampPalette (c ("orange", "darkred")))
```



Deriving manufacturer specific import filters. Please note that future changes inside the read.spc function are likely to occur. However, if you just post-process the *hyperSpec* object returned by `read.spc`, you should be fine.

5 Manufacturer-Specific Discussion of File Import

5.1 Manufacturer Specific Import Functions

Many spectrometer manufacturers provide a function to export their spectra into ASCII files. The functions discussed above are written in a very general way, and are highly customizable. I recommend wrapping these calls with the appropriate settings for your spectra format in an import function. You may also consider contributing such import filters to *hyperSpec*: send me ClaudiaBeleites<cbeleites@units.it>the documented code (either .R + .Rd file or Roxygen commented .R).

If you are able to import data of any format not mentioned in this document (even without the need of new converters), please let me know (see the box at the beginning of this document).

5.2 Bruker FT-IR Imaging

We use `read.ENVI` to import IR-Images collected with a Bruker Hyperion spectrometer with OPUS software. As mentioned above, the header files are frequently missing. We found the necessary information to be:

```
> header <- list (samples = 64 * no.images.in.row,
+                 lines = 64 * no.images.in.column,
+                 bands = no.data.points.per.spectrum,
+                 `data type` = 4,
+                 interleave = "bip")
```

No spatial information is given in the ENVI header (if written). The lateral coordinates can be setup by specifying origin and pixel size for x and y directions. For details please see the help page.

The proprietary file format of the Opus software is not (yet) supported.

5.3 Nicolet FT-IR Imaging

Also Nicolet saves imaging data in ENVI files. These files use some non-standard keywords in the header file that should allow to reconstruct the lateral coordinates as well as the wavelength axes and units for wavelength and intensity axis. *hyperSpec* has a specialized function `read.ENVI.Nicolet` that uses these header entries.

It seems that the position of the first spectrum is recorded in µm, while the pixel size is in mm. Thus a flag *nicolet.correction* is provided that divides the pixel size by 1000. Also here, giving the correct offset and pixel size values as function arguments is possible.

```
> spc <- read.ENVI.Nicolet ("ENVI/example2.img", nicolet.correction = TRUE)

.read.ENVI.header: Guessing header file name (ENVI/example2.hdr)
.read.ENVI.bin: 'byte order' not given or incorrect. Guessing 'little'

> spc ## dummy sample with all intensities zero

hyperSpec object
   420 spectra
   3 data columns
   1738 data points / spectrum
wavelength:  [numeric] 649.90 651.83 ... 3999.7
data:  (420 rows x 3 columns)
   1. x:  [numeric] rng  -102377.1 -102372.1 ... -102312.1
   2. y:  [numeric] rng  -8936 -8931 ... -8791
   3. spc:  [AsIs matrix x 1738] rng  0
```

5.4 Kaiser Optical Systems Raman

Spectra obtained using Kaiser's Hologram software can be saved either in their own .hol format and imported into Matlab (from where the data may be written to a .mat file readable by *R.matlab*'s `readMat`. Alternatively, Hologram can write ASCII files and .spc files. We found working with .spc files the best option.

The spectra are usually interpolated to an evenly spaced wavelength (or $\Delta\tilde{\nu}$) axis unless the spectra are saved in a by-pixel manner.

5.4.1 Kaiser Optical Systems ASCII Files

The ASCII files are long format that can be imported by `read.txt.long` (see section 3, p. 3).

We experienced two different problems with these files:

1. If the instrument computer's locale is set so that also the decimal separator is a comma, commas are used both as decimal and as column separator.
2. Values with a decimal fraction of 0 are written as e.g. `2,`. This may be a problem for certain conversion functions.

Problems may arise

Still the files may be imported, though care must be taken:

```
> ## 1. import as character
> tmp <- scan ("txt.Kaiser/test-lo-4.txt", what = rep ("character",4), sep = ",")
> tmp <- matrix (tmp, nrow = 4)
> ## 2. concatenate every two columns by a dot
> wl <- apply (tmp [1:2, ], 2, paste, collapse = '.')
> spc <- apply (tmp [3:4, ], 2, paste, collapse = '.')
> ## 3. convert to numeric and create hyperSpec object
> spc <- new ("hyperSpec", spc = as.numeric (spc), wavelength = as.numeric (wl))
```

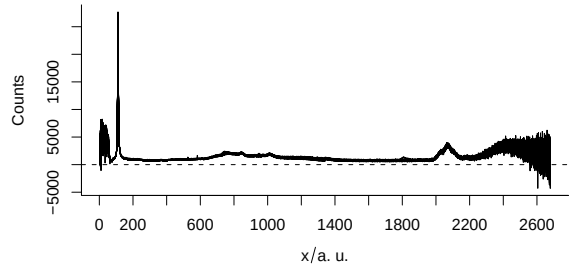
5.4.2 Kaiser Optical Systems Raman Maps

Spectra obtained using Kaiser's Hologram software can be saved either in their own .hol format and imported into Matlab (from where the data may be written to a .mat file readable by *R.matlab*'s `readMat`. Alternatively, Hologram can write ASCII files and .spc files. We found working with .spc files the best option.

hyperSpec provides the function `read.spc.KaiserMap` to easily import spatial collections of .spc files written by Kaiser's Hologram software. The filenames of all .spc files to be read into one *hyperSpec* object can be provided either in a character vector or as a wildcard expression (e. g. "path/to/files/*.spc").

The data for the following example was saved with wavelength axis camera pixels rather than Raman shift. Thus two files for each spectrum (one low wavenumber region, one high wavenumber region) were saved by Hologram. Thus, a file name pattern is difficult to give and consequently a vector of file names is used instead:

```
> files <- Sys.glob ("spc.KaiserMap/*.spc")
> spc.low <- read.spc.KaiserMap (files [seq (1, length (files), by = 2)])
> spc.high <- read.spc.KaiserMap (files [seq (2, length (files), by = 2)])
> wl (spc.high) <- wl (spc.high) + 1340
> plot (cbind (spc.low, spc.high))
```



5.5 Renishaw Raman

Renishaw's Wire software comes with an file format converter. This program can produce a long ASCII format, .spc, or .jdx files.

We experienced that the conversion to .spc is *not* fully reliable: maps were saved as depth profile, loosingall spatial information. Also, an evenly spaced wavelength axis was produced, although this was de-selected in the converter. We therefore recommend using the ASCII format. Otherwise the import using `read.spc` worked.

5.5.1 Renishaw ASCII data

An optimized import function for the ASCII files is available: `scan.txt.Renishaw`. The ASCII files can easily become very large, particularly with linefocus- or streamline imaging. `scan.txt.Renishaw` provides two mechanisms to avoid running out of memory during data import. First of all, the file may be imported in chunks (of a given number of lines). Secondly, the processing of the long ASCII format into the spectra matrix is done by reshaping the vector of intensities into a matrix. This process does not allow any missing values in the data. *Therefore it is not possible to import multi-spectra files with individually "zapped" spectra.*

The second argument to `scan.txt.Renishaw` decides what type of experiment is imported. Supported types are:

`"xyspc"`	maps, images, multiple spectra with x and y coordinates (default)
`"spc"`	single spectrum
`"depth"`, `"zspc"`	depth series
`"ts"`	time series

```
> scan.txt.Renishaw ("txt.Renishaw/paracetamol.txt", "spc")

hyperSpec object
   1 spectra
   1 data columns
   4064 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 96.7865 98.1432 ... 3200.07
data:  (1 rows x 1 columns)
   1. spc: I / a.u. [AsIs matrix x 4064] rng  299.229 317.041 ... 49052.2

> scan.txt.Renishaw ("txt.Renishaw/laser.txt", "ts")

hyperSpec object
   84 spectra
   2 data columns
   574 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] -633.346 -631.072 ... 595.505
data:  (84 rows x 2 columns)
   1. t: t / s [numeric] rng  0 2 ... 5722
   2. spc: I / a.u. [AsIs matrix x 574] rng  -29.7177 -20.9125 ... 73934.4

> scan.txt.Renishaw ("txt.Renishaw/chondro.txt", nlines = 1e5, nspc = 875)

............
hyperSpec object
   875 spectra
   3 data columns
```

```
   1272 data points / spectrum
wavelength: Delta * tilde(nu)/cm^-1 [numeric] 601.622 602.664 ... 1802.15
data:  (875 rows x 3 columns)
   1. y: y/(mu * m) [numeric] rng  -4.77 -3.77 ... 19.23
   2. x: x/(mu * m) [numeric] rng  -11.55 -10.55 ... 22.45
   3. spc: I / a.u. [AsIs matrix x 1272] rng  52.2573 52.5012 ... 1884.25 + NA
```

6 Writing your own Import Function

This section gives examples how to write import functions. The first example implements an import filter for an ASCII file format basically from scratch. The second example shows how to implement more details for an already existing import filter.

6.1 A new ASCII Import Function: scan.txt.PerkinElmer

The raw spectra of the `flu` data set (see also the respective vignette) are in Perkin Elmer's ASCII file format, one spectrum per file.

We need a function that automatically reads all files specified by a pattern, such as `*.txt`. In order to gain speed, the spectra matrix should be preallocated after the first file is read.

A short examination of the files (`flu*.txt` in directory `txt.PerkinElmer`) reveals that the actual spectrum starts at line 55, after a line containing `#DATA`. For now, no other information of the files is to be extracted. It is thus easier to skip the first 54 lines than searching for the line after `#DATA`.

A fully featured import function should support:

- Reading multiple files by giving a pattern
- hand further arguments to `scan`. This comes handy in case the function is used later to import other data types.
- Also skipping 54 lines would be a weird default, so we rather require it to be given explicitly.
- The same applies for the axis labels: they should default to reasonable settings for fluorescence spectra, but it should be possible to change them if needed.
- The usual log entry arguments should be supplied.
- A sanity check should be implemented: stop with an error if a file does not have the same wavelength axis as the others.
- Finally, if no file can be found, an empty *hyperSpec* object is a reasonable result: There is no need to stop with an error, but it is polite to issue an additional warning.

scan.txt.PerkinElmer.R

```
scan.txt.PerkinElmer <- function (files = "*.txt",  ..., label = list (),
                                  short = "scan.txt.PerkinElmer", user = NULL, date = NULL) {
  ##  set some defaults
  long <- list (files = files, ..., label = label)

  label <-  modifyList (list (.wavelength = expression (lambda / nm),
                              spc = "I[fl] / a.u."),
                        label)

  ## find the files
  files <- Sys.glob (files)

  if (length (files) == 0){
```

```
    warning ("No files found.")
    return (new ("hyperSpec"))
  }

  ## read the first file
  buffer <- matrix (scan (files [1], ...), ncol = 2, byrow = TRUE)

  ## first column gives the wavelength vector
  wavelength <- buffer [, 1]

  ## preallocate the spectra matrix:
  ##  one row per file x as many columns as the first file has
  spc <- matrix (ncol = nrow (buffer), nrow = length (files))

  ## the first file's data goes into the first row
  spc [1, ] <-  buffer [, 2]

  ## now read the remaining files
  for (f in seq (along = files)[-1]) {
    buffer <- matrix (scan (files [f], ...), ncol = 2, byrow = TRUE)

    ## check whether they have the same wavelength axis
    if (! all.equal (buffer [, 1], wavelength))
      stop (paste(files [f], "has different wavelength axis."))

    spc [f, ] <- buffer[, 2]
  }

  ## make the hyperSpec object
  new ("hyperSpec", wavelength = wavelength, spc = spc,
       data = data.frame (file = files), label = label,
       log = list (short = short, long = long, user = user, date = date))
}
```

Note how the labels are set. The label with the special name `.wavelength` corresponds to the wavelength axis, all data columns should have a label with the same name. The spectra are always in a data column called `spc`.

Thus,

```
> source ("scan.txt.PerkinElmer.R")
> scan.txt.PerkinElmer ("txt.PerkinElmer/flu?.txt", skip = 54)
```

```
hyperSpec object
   6 spectra
   2 data columns
   181 data points / spectrum
wavelength: lambda/nm [numeric] 405.0 405.5 ... 495
data:  (6 rows x 2 columns)
   1. file:  [factor] rng  txt.PerkinElmer/flu1.txt txt.PerkinElmer/flu2.txt ... txt.PerkinElmer/flu6.txt
   2. spc: I[fl] / a.u. [AsIs matrix x 181] rng  27.15000 32.34467 ... 677.4947
```

imports the spectra.

This function is not exported by *hyperSpec*. While it is already useful for importing files, it is not yet general enough to work immediately with new data. The the file header is completely ignored.

6.2 Deriving a More Specific Function: read.ENVI.Nicolet

The function `read.ENVI.Nicolet` is a good example for a more specific import filter derived from a general filter for the respective file type. Nicolet FT-IR Imaging software saves some non-standard

key-words in the header file of the ENVI data. These information can be used to reconstruct the x and y axes of the images. The units of the spectra are saved as well.

`read.ENVI.Nicolet` thus first adjusts the parameters for `read.ENVI`. Then `read.ENVI` does the main work of importing the file. The resulting *hyperSpec* object is post-processed according to the special header entries.

For using the function, see section **??** (p. **??**).

read.ENVI.Nicolet.R

```
read.ENVI.Nicolet <- function (..., # goes to read.ENVI
               # file headerfile, header
               x = NA, y = NA, # NA means: use the specifications from the header file if possible
               log = list (),
               keys.hdr2log = TRUE,
               nicolet.correction = FALSE) {

  ## set some defaults
       log <- modifyList (list (short = "read.ENVI.Nicolet",
                                                                    long = list (call = match.call ())),
                                                        log)
  ## the additional keywords to interprete must be read
       if (! isTRUE (keys.hdr2log))
               keys.hdr2log <- unique (c ("description", "z plot titles", "pixel size", keys.hdr2log))

  ## most work is done by read.ENVI
       spc <- read.ENVI (..., keys.hdr2log = keys.hdr2log,
                       x = if (is.na (x)) 0 : 1 else x,
                       y = if (is.na (y)) 0 : 1 else y,
                       log = log)

  ## get the header for post-processing
       header <-spc@log$long.description [[1]]$header

### From here on processing the additional keywords in Nicolet's ENVI header ************************

  ## z plot titles -------------------------------------------------------------------------------
  ## default labels
       label <- list (x = expression (`/` (x, micro * m)),
                                             y = expression (`/` (y, micro * m)),
                                             spc = 'I / a.u.',
                                             .wavelength = expression (tilde (nu) / cm^-1))

  ## get labels from header information
       if (!is.null (header$'z plot titles')){
               pattern <- "^[[:blank:]]*([[:print:]^,]+)[[:blank:]]*,.*$"
               tmp <- sub (pattern, "\\1", header$'z plot titles')

               if (grepl ("Wavenumbers (cm-1)", tmp, ignore.case = TRUE))
                       label$.wavelength <- expression (tilde (nu) / cm^(-1))
               else
                       label$.wavelength <- tmp

               pattern <- "^[[:blank:]]*[[:print:]^,]+,[[:blank:]]*([[:print:]^,]+).*$"
               tmp <- sub (pattern, "\\1", header$'z plot titles')
               if (grepl ("Unknown", tmp, ignore.case = TRUE))
                       label$spc <- "I / a.u."
               else
                       label$spc <- tmp
       }

  ## modify the labels accordingly
       spc@label <- modifyList (label, spc@label)

  ## set up spatial coordinates ------------------------------------------------------------------
       ## look for x and y in the header only if x and y are NULL
```

```
  ## they are in `description` and `pixel size`

  ## set up regular expressions to extract the values
        p.description <- "^Spectrum position [[:digit:]]+ of [[:digit:]]+ positions, X = ([[:digit:].-]+), Y = ([[:digit:
        p.pixel.size  <- "^[[:blank:]]*([[:digit:].-]+),[[:blank:]]*([[:digit:].-]+).*$"

        if (is.na (x) && is.na (y) &&
                        ! is.null (header$description)  && grepl (p.description, header$description ) &&
                        ! is.null (header$'pixel size') && grepl (p.pixel.size,  header$'pixel size')) {

                x [1] <- as.numeric (sub (p.description, "\\1", header$description))
                y [1] <- as.numeric (sub (p.description, "\\2", header$description))

                x [2] <- as.numeric (sub (p.pixel.size, "\\1", header$'pixel size'))
                y [2] <- as.numeric (sub (p.pixel.size, "\\2", header$'pixel size'))

    ## it seems that the step size is given in mm while the offset is in micron
                if (nicolet.correction) {
                        x [2] <- x [2] * 1000
                        y [2] <- y [2] * 1000
                }

    ## now calculate and set the x and y coordinates
                x <- x [2] * spc$x + x [1]
                if (! any (is.na (x)))
                        spc@data$x <- x

                y <- y [2] * spc$y + y [1]
                if (! any (is.na (y)))
                        spc@data$y <- y
        }
        spc
}
```

6.3 Deriving import filters for spc files

Please note that future changes inside the read.spc function are likely to occur. However, if you just post-process the *hyperSpec* object returned by `read.spc`, you should be fine.

References

[] Henrik Bengtsson and Jason Riedy. *R.matlab: Read and write of MAT files together with R-to-Matlab connectivity*, 2009. URL http://CRAN.R-project.org/package=R.matlab. R package version 1.2.6.

A File Import Functions by Format

Format	Manufacturer	Function	section	Notes
ASCII file formats				
ASCII long		`read.txt.long`	3, p. 3	
ASCII long	Renishaw (Raman)	`scan.txt.Renishaw`	??, p. ??	
ASCII long	Kaiser (Raman)	`read.txt.long`	??, p. ??	*Not* recommended, see discussion
ASCII long	Perkin Elmer (Fluorescence)	`read.txt.PerkinElmer`	??, p. ??	Reads multiple files, needs to be sourced.
ASCII wide		`read.txt.wide`	3, p. 3	
JCAMP-DX		-	3.2, p. 4	not available
JCAMP-DX	Renishaw (Raman)	-	3.2, p. 4	not available
binary file formats				
ENVI		`read.ENVI`	4.2, p. 5	
ENVI	Bruker (Infrared Imaging)	`read.ENVI`	??, p. ??	
ENVI	Nicolet (Infrared Imaging)	`read.ENVI.Nicolet`	??, p. ??	
hol	Kaiser (Raman)	-	??, p. ??	via Matlab
Matlab	Matlab	`R.matlab::readMat`	4.1, p. 4	
Opus	Bruker (Infrared Imaging)	-	??, p. ??	
spc		`read.spc`	4.3, p. 6	
spc	Kaiser (Raman Map)	`read.spc.KaiserMap`	??, p. ??	Reads multiple files
spc	Kaiser (Raman)	`read.spc`	4.3, p. 6	Reads multiple files
spc	Renishaw (Raman)	`read.spc`	??, p. ??	*Not* recommended, see discussion of ASCII files.

B File Import Functions by Manufacturer

Manufacturer	Format	Function	section	Notes
Manufacturers				
Bruker (Infrared Imaging)	ENVI	`read.ENVI`	??, p. ??	
Bruker (Infrared Imaging)	Opus	-	??, p. ??	
Kaiser (Raman)	ASCII long	`read.txt.long`	??, p. ??	*Not* recommended, see discussion
Kaiser (Raman)	hol	-	??, p. ??	via Matlab
Kaiser (Raman Map)	spc	`read.spc.KaiserMap`	??, p. ??	Reads multiple files
Kaiser (Raman)	spc	`read.spc`	4.3, p. 6	Reads multiple files
Matlab	Matlab	`R.matlab::readMat`	4.1, p. 4	
Nicolet (Infrared Imaging)	ENVI	`read.ENVI.Nicolet`	??, p. ??	
Perkin Elmer (Fluorescence)	ASCII long	`read.txt.PerkinElmer`	??, p. ??	Reads multiple files, needs to be sourced.
Renishaw (Raman)	ASCII long	`scan.txt.Renishaw`	??, p. ??	
Renishaw (Raman)	JCAMP-DX	-	3.2, p. 4	not available
Renishaw (Raman)	spc	`read.spc`	??, p. ??	*Not* recommended, see discussion of ASCII files.

C File Import Functions by Spectroscopy

Spectroscopy	Format	Manufacturer	Function	section	Notes
Fluorescence	ASCII long	Perkin Elmer	`read.txt.PerkinElmer`	??, p. ??	Reads multiple files, needs to be sourced.
Infrared Imaging	ENVI	Bruker	`read.ENVI`	??, p. ??	
Infrared Imaging	ENVI	Nicolet	`read.ENVI.Nicolet`	??, p. ??	
Infrared Imaging	Opus	Bruker	-	??, p. ??	
Raman	ASCII long	Renishaw	`scan.txt.Renishaw`	??, p. ??	
Raman	ASCII long	Kaiser	`read.txt.long`	??, p. ??	*Not* recommended, see discussion
Raman	hol	Kaiser	-	??, p. ??	via Matlab
Raman	JCAMP-DX	Renishaw	-	3.2, p. 4	not available
Raman	spc	Kaiser	`read.spc`	4.3, p. 6	Reads multiple files
Raman	spc	Renishaw	`read.spc`	??, p. ??	*Not* recommended, see discussion of ASCII files.
Raman Map	spc	Kaiser	`read.spc.KaiserMap`	??, p. ??	Reads multiple files

Index