Calibration of Quinine Fluorescence Emission Vignette for the Data Set `flu` of the R package hyperSpec

Claudia Beleites <Claudia.Beleites@chemometrix.gmbh>
DIA Raman Spectroscopy Group, University of Trieste/Italy (2005 – 2008)
Spectroscopy · Imaging, IPHT, Jena/Germany (2008 – 2017)
ÖPV, JKI, Berlin/Germany (2017 – 2019)
Arbeitskreis Lebensmittelmikrobiologie und Biotechnologie, Hamburg University, Hamburg/Germa
Chemometric Consulting and Chemometrix GmbH, Wölfersheim/Germany (since 201

November 27, 2020

Reproducing the Examples in this Vignette

The spectra files are shipped with *hyperSpec*. This allows reproduction of the whole vignette (source code and spectra files are in the package's documentation directory and its `rawdata` subdirectory). For reproducing the examples in a live session, the full file names of the spectra can be found with the command:
`> list.files (system.file("doc", package = "hyperSpec"), pattern = "flu[1-6][.]txt").` Note that loading the package, and some definitions e.g. of the color palettes are executed in `vignettes.def`.

Contents

This vignette gives an example how to

- write an import function for a spectrometer manufacturer's proprietary ASCII files,
- add further data columns to the spectra, and
- set up a linear calibration (inverse least squares).

The data set `flu` in *hyperSpec* consists of 6 fluorescence emission spectra of quinine solutions. They were acquired during an student practicum and were kindly provided by M. Kammer.

The concentrations of the solutions range from 0.05 mg/l to 0.30 mg/l. Spectra were acquired with a Perkin Elmer LS50-B fluorescence spectrometer at 350 nm excitation.

1 Writing an Import Function

The raw spectra are in Perkin Elmer's ASCII file format, one spectrum per file. The files are completely ASCII text, with the actual spectra starting at line 55.

The function to import these files, `read.txt.PerkinElmer`, is discussed in the "FileIO" vignette, please refer to that document for details.

It needs to be `source`d before use:

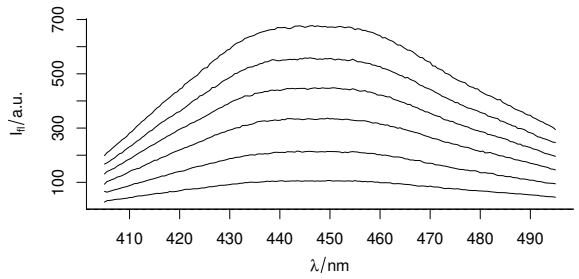
```
> source ("read.txt.PerkinElmer.R")
> flu <- read.txt.PerkinElmer (Sys.glob ("rawdata/flu?.txt"), skip = 54)
```

Now the spectra are in a *hyperSpec* object and can be examined e.g. by

```
> flu
```

```
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. spc: I[fl]/"a.u." [matrix181] 27.150 66.801 ... 294.65
   2. filename: filename [character] rawdata/flu1.txt rawdata/flu2.txt ... rawdata/flu6.txt
```

```
> plot (flu)
```



2 Adding further Data Columns

The calibration model needs the quinine concentrations for the spectra. This information can be stored together with the spectra, and also gets an appropriate label:

```
> flu$c <- seq (from = 0.05, to = 0.30, by = 0.05)
> labels (flu, "c") <- "c / (mg / l)"
> flu
```

```
hyperSpec object
   6 spectra
   3 data columns
   181 data points / spectrum
wavelength: lambda/nm [numeric] 405.0 405.5 ... 495
```

```
data:  (6 rows x 3 columns)
   1. spc: I[fl]/"a.u." [matrix181] 27.150 66.801 ... 294.65
   2. filename: filename [character] rawdata/flu1.txt rawdata/flu2.txt ... rawdata/flu6.txt
   3. c: c / (mg / l) [numeric] 0.05 0.10 ... 0.3
```

```
> save (flu, file = 'flu.rda')
```

Now the *hyperSpec* object `flu` contains two data columns, holding the actual spectra and the respective concentrations. The dollar operator returns such a data column:

```
> flu$c
```

```
[1] 0.05 0.10 0.15 0.20 0.25 0.30
```

3 Dropping data columns

`read.txt.PerkinElmer` added a column with the file names that we don't need. It is therefore deleted:

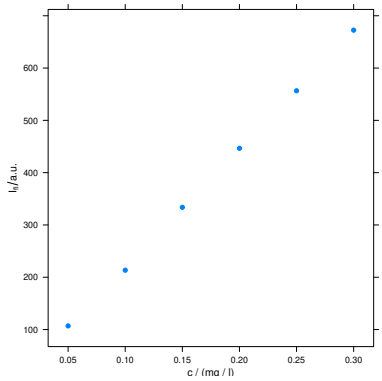
```
> flu$filename <- NULL
```

4 Linear Calibration

As R is developed for the purpose of statistical analysis, tools for a least squares calibration model are readily available.

The original spectra range from 405 to 495 nm. However, the intensities at 450 nm are perfect for a univariate calibration. Plotting them over the concentration is done by:

```
> plotc (flu[,,450])
```



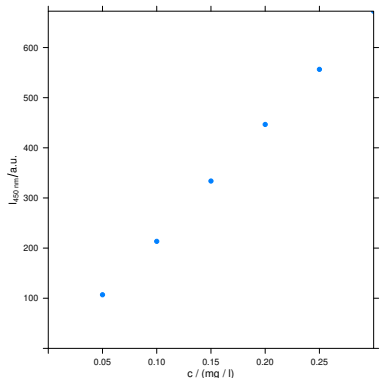
The square bracket operator extracts parts of a *hyperSpec* object. The first coordinate defines which spectra are to be used, the second which data columns, and the third gives the spectral range.

We discard all the wavelengths but 450 nm:

```
> flu <- flu [,,450]
> labels (flu, "spc") <- expression (I ["450 nm"] / a.u.)
```

The plot could be enhanced by annotating the ordinate with the emission wavelength. Also the axes should start at the origin, so that it is easier to see whether the calibration function will go through the origin:

```
> plotc (flu, xlim = range (0, flu$c), ylim = range (0, flu$spc))
```



The actual calibration is a linear model, which can be fitted by the R function `lm`. `lm` needs a *formula* that specifies which data columns are dependent and independent variables.

The normal calibration plot gives the emission intensity as a function of the concentration, and the calibration function thus models $I = f(c)$, i. e. $I = mc + b$ for a linear calibration. This is then solved for c when the calibration is used.

However, R's linear model is a quite strict in predicting: a model set up as $I = f(c)$ will predict the intensity as a function of the concentration but not the other way round. Thus we set up an inverse calibration model[1]: $c = f(I)$. The corresponding `R` formula is `c ~ I`, or in our case `c ~ spc`, as the intensities are stored in the data column `$spc`:

In addition, `lm` (like most R model building functions) expects the data to be a *data.frame*.

There are three abbreviations that help to get the parts of the *hyperSpec* object that are frequently needed:

`flu[[]]`	returns the spectra matrix. It takes the same indices as `[]`.
`flu$.`	returns the data as a *data.frame*
`flu$..`	returns a *data.frame* that has all data columns but the spectra

```
> flu[[]]

        450
[1,] 106.95
[2,] 213.50
[3,] 333.78
[4,] 446.63
[5,] 556.52
[6,] 672.53

> flu$.

     450    c
1 106.95 0.05
2 213.50 0.10
3 333.78 0.15
4 446.63 0.20
5 556.52 0.25
6 672.53 0.30

> flu$..
```

[1] As we can safely assume that the error on the concentrations is far larger than the error on the instrument signal, it is actually the correct type of model from the least squares fit point of view.

```
     c
1 0.05
2 0.10
3 0.15
4 0.20
5 0.25
6 0.30
```

Putting this together, the calibration model is calculated:

```
> calibration <- lm (c ~ spc, data = flu$.)
```

The `summary` gives a good overview of our model:

```
> summary (calibration)

Call:
lm(formula = c ~ spc, data = flu$.)

Residuals:
        1         2         3         4         5         6
-0.000987  0.002052 -0.000960 -0.000701  0.000864 -0.000267

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 3.85e-03   1.25e-03    3.09    0.037 *
spc         4.41e-04   2.87e-06  153.60  1.1e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.00136 on 4 degrees of freedom
Multiple R-squared:     1,        Adjusted R-squared:     1
F-statistic: 2.36e+04 on 1 and 4 DF,  p-value: 1.08e-08
```

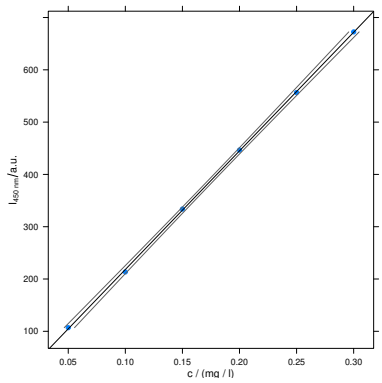
In order to get predictions for new measurements, a new *data.frame* with the same independent variables (in columns with the same names) as in the calibration data are needed. Then the function `predict` can be used. It can also calculate the prediction interval. If we observe e.g. an intensity of 125 or 400 units, the corresponding concentrations and their 99 % prediction intervals are:

```
> I <- c (125, 400)
> conc <- predict (calibration, newdata = list (spc = as.matrix(I)), interval = "prediction",
+                  level = .99)
> conc

       fit     lwr      upr
1 0.058943 0.05133 0.066556
2 0.180149 0.17338 0.186922
```

Finally, we can draw the calibration function and its 99 % confidence interval (also via `predict`) together with the prediction example. In order to draw the confidence interval into the calibration graph, we can either use a customized panel function:

```
> int <- list (spc = as.matrix(seq (min (flu), max(flu), length.out = 25)))
> ci <- predict (calibration, newdata = int, interval = "confidence", level = 0.99)
> panel.ci <-  function (x, y, ...,
+                        intensity, ci.lwr, ci.upr, ci.col = "#606060") {
+    panel.xyplot (x, y, ...)
+    panel.lmline (x, y,...)
+    panel.lines (ci.lwr, intensity, col = ci.col)
+    panel.lines (ci.upr, intensity, col = ci.col)
+ }
> plotc (flu, panel = panel.ci,
+        intensity = int$spc, ci.lwr = ci [, 2], ci.upr = ci [, 3])
```



Or, we can add the respective data to the *hyperSpec* object. The meaning of the data can be saved in a new extra data column that acts as grouping variable for the plot.

First, the spectral range of `flu` is cut to contain the fluorescence emission at 450 nm only, and the new column is introduced for the original data:

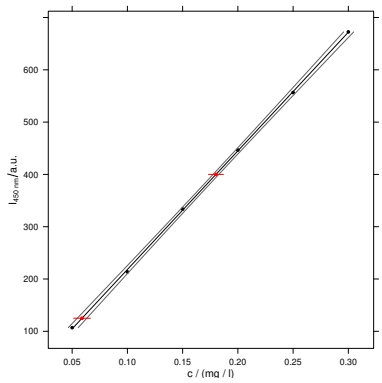
```
> flu$type <- "data points"
```

Next, the calculated confidence intervals are appended:

```
> tmp <- new ("hyperSpec", spc = as.matrix(seq (min (flu), max(flu), length.out = 25)),
+                          wavelength = 450)
> ci <-  predict (calibration, newdata = tmp$., interval = "confidence", level = 0.99)
> tmp <- tmp [rep (seq (tmp, index = TRUE), 3)]
> tmp$c <- as.numeric (ci)
> tmp$type <- rep (colnames (ci), each = 25)
> flu <- collapse (flu, tmp)
```

Finally, the resulting object is plotted. Our prediction example is handled by another customized panel function:

```
> panel.predict <- function (x, y, ...,
+                   intensity, ci, pred.col = "red", pred.pch = 19, pred.cex = 1) {
+    panel.xyplot (x, y, ...)
+    mapply (function (i, lwr, upr, ...) {
+                   panel.lines (c (lwr, upr), rep (i, 2), ...)
+               },
+            intensity, ci [, 2], ci [, 3], MoreArgs = list (col = pred.col))
+    panel.xyplot (ci [, 1], intensity, col = pred.col, pch = pred.pch, cex = pred.cex, type = "p")
+ }
> plotc (flu, groups = type, type = c("l", "p"),
+        col = c ("black", "black", "#606060", "#606060"),
+        pch = c (19, NA, NA, NA), cex = 0.5,
+        lty = c (0, 1, 1, 1),
+        panel = panel.predict,
+        intensity = I,
+        ci = conc,
+        pred.cex = 0.5)
```



Session Info

```
R version 3.6.3 (2020-02-29)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.5 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/libopenblasp-r0.2.20.so

locale:
 [1] LC_CTYPE=de_DE.UTF-8       LC_NUMERIC=C               LC_TIME=de_DE.UTF-8
 [4] LC_COLLATE=C               LC_MONETARY=de_DE.UTF-8    LC_MESSAGES=de_DE.UTF-8
 [7] LC_PAPER=de_DE.UTF-8       LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=de_DE.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] tools     grid      stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] hyperSpec_0.99-20201127 xml2_1.3.2              ggplot2_3.3.2           lattice_0.20-41

loaded via a namespace (and not attached):
 [1] magrittr_2.0.1      tidyselect_1.1.0    munsell_0.5.0       colorspace_2.0-0
 [5] R.cache_0.14.0      R6_2.5.0            jpeg_0.1-8.1        rlang_0.4.8
 [9] dplyr_1.0.2         gtable_0.3.0        png_0.1-7           R.oo_1.24.0
[13] latticeExtra_0.6-29 withr_2.3.0         ellipsis_0.3.1      lazyeval_0.2.2
[17] digest_0.6.27       tibble_3.0.4        lifecycle_0.2.0     crayon_1.3.4
[21] R.rsp_0.44.0        RColorBrewer_1.1-2  purrr_0.3.4         vctrs_0.3.5
[25] R.utils_2.10.1      testthat_3.0.0      glue_1.4.2          compiler_3.6.3
[29] pillar_1.4.7        generics_0.1.0      scales_1.1.1        R.methodsS3_1.8.1
[33] pkgconfig_2.0.3
```