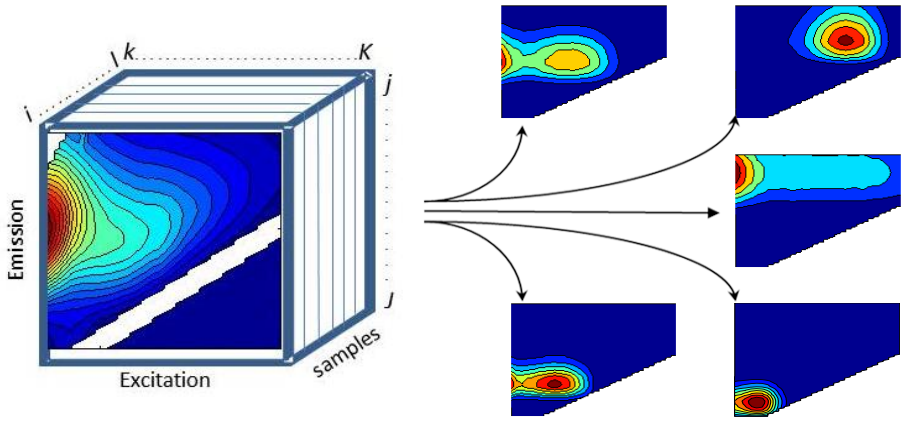
Appendix A.

Decomposition routines for Excitation Emission Matrices (drEEM version 0.1.0)

in

Murphy K.R., Stedmon C.A., Graeber D. and R. Bro, Fluorescence spectroscopy and multi-way techniques. PARAFAC, Anal. Methods, *in press*, DOI:10.1039/c3ay41160e.



Contents

1 Foreword

This document contains a tutorial on using the **drEEM** toolbox for implementing spectral correction and PARAFAC analysis of fluorescence excitation emission matrices (EEMs). It is published as an appendix to an article in the journal *Analytical Methods*, which is part of a Special Issue (web collection) on the topic of Tutorials in Advanced Data Analysis.

The tutorial and drEEM (decomposition routines for Excitation Emission Matrices) software can be cited as follows:

Murphy K.R., Stedmon C.A., Graeber D. and R. Bro, Fluorescence spectroscopy and multi-way techniques. PARAFAC, Anal. Methods, *in press*, DOI:10.1039/c3ay41160e.

This tutorial accompanies drEEM version 0.1.0. It is expected that periodic revisions will be made to the drEEM software and to this document. Consult the drEEM website for updates, discussions and bug fixes (http://models.life.ku.dk/drEEM).

2 Setting up MATLAB to run with drEEM and the *N*-way toolbox.

A MATLAB file containing a program or function has the file extension '.m' and is termed an mfile. In this tutorial, mfiles and folder names are shown in *italics*. Any statements to be typed in MATLAB's Command Window are written in `Courier` font. Note that MATLAB commands are case-sensitive.

To install drEEM (version 0.1.0, August 2013) and the N-way toolbox (version 3.20, July 2012):

i. Choose a place to locate the drEEM toolbox on your computer, for example C:\Program Files\MATLAB\R2012b\toolbox. Create a folder and name it drEEM. Extract the contents of the zip file "drEEM.zip" into the drEEM folder.

ii. Start MATLAB. Select [File] menu then click [Set path…] (in later versions of MATLAB, the Set Path icon is located in the environment tab). Click [Add] and navigate to the drEEM folder you created in Step 1 above. Click [Add] again and locate the N-way folder inside drEEM. This associates the drEEM and N-way toolboxes with MATLAB so that their functions can be called from within MATLAB. Click [Close] and [Save].

iii. If earlier versions of the toolboxes were installed previously on your computer, delete them to avoid conflicts. Also, if you have the DOMFluor toolbox installed (Stedmon and Bro, 2008, *Limnol Oceanogr Meth* 6, 572-579) delete the N-way subfolder that came packaged with DOMFluor.

iv. If a new version of MATLAB is installed, or if changes are made to the MATLAB path, these steps may need repeating.

Check that the toolboxes are operational by typing in MATLAB's Command Window using *lookforconflicts*.

```
lookforconflicts('drEEM')
lookforconflicts('nway')
```

If you receive a message confirming that no conflicts were found, this indicates that the toolbox is correctly installed and all mfiles within the toolbox should work properly. Conflicts will be listed if mfiles with the same name as those in the toolbox have higher precedence on your MATLAB path. If you wish to give drEEM mfiles precedence, change the order that folders are listed in the path so that drEEM ranks higher than the folder containing the conflicting mfile.

If you have the older DOMFluor toolbox installed and specified on the MATLAB path, try:

```
lookforconflicts('DOMFluor')
```

DOMFluor contains several files with the same names as drEEM except for capital letters. These should not cause conflicts because the names of all files in drEEM filenames are in lower case.

3 Creating a fully corrected EEM dataset from Raw data files.

Section 3 of this tutorial contains a demonstration of how to create a fully corrected three-dimensional dataset from raw data files according to standardised procedures (Murphy et al., 2010, *Environ Sci Technol*, 44, 9405–9412).

If you already know how to obtain corrected datasets or would prefer to go straight to the PARAFAC section of this tutorial, skip ahead to Section 4.

3.1 About the dataset

The demo dataset contains measurements made during four surveys of San Francisco Bay that took place in spring, summer, autumn and winter 2006 (Murphy et al. 2013, *J. Mar. Syst.* 111-112, 157-166). The dataset includes:

1. **Fluorescence EEMs** for samples and Milli-Q blanks, measured at 20 °C in a 1 cm quartz cell with a SPEX Fluorolog-3 from Horiba Jobin Yvon (Edison, New Jersey, USA). The experimental wavelength range was 230 to 455 nm in 5 nm intervals on excitation and 290 to 700 nm in 4 nm intervals on emission. Measurements were performed in ratio mode with 0.5 s integration times and 5-nm slit widths on the excitation and emission monochromators.

2. **Water Raman** scans of Milli-Q blanks obtained daily were also measured at 20 °C with the SPEX Fluorolog-3. Scans used an excitation wavelength of 275 nm and recorded emission at 1-nm intervals from 285-450 nm. Measurements were performed in ratio mode with 0.5 s integration times and 5-nm slit widths on the excitation and emission monochromators. Note that the water Raman scans were collected at Ex = 275 nm, not Ex = 350 nm as has been

recommended in recent studies (e.g. Lawaetz and Stedmon, 2007, *Appl. Spectrosc*. 63, 936-940, Murphy et al. *Environ. Sci. Technol.*, 2010, 44, 9405–9412). Use of wavelengths other than Ex = 350 nm is perfectly legitimate and common in studies that ultimately convert fluorescence intensities to Quinine-Sulfate (QSE) units.

3. **Absorbance** measured at 20 °C in either 5 or 10 cm quartz cells with a Cary 4E spectrophotometer. Measurements were obtained at 1-nm wavelength intervals from 290 to 750 nm, with ultrapure water as reference. Data were converted to absorbance in a 1-cm cell. Absorbances in the range 230 nm – 290 nm were estimated by linear back-extrapolation of the log-transformed spectra measured between 290-310 nm. Missing data were estimated using average values for samples with similar fluorescence characteristics.

4. **Quinine sulfate dilution series.** Fresh 5-point dilution series (1, 4, 10, 20, 100 ppb QSE) were created on three occasions in 2006. The slopes of each dilution series were determined at Ex/Em = 350/450 nm. Emission (water Raman) scans were obtained on the same days using excitation wavelengths of 275 nm and 350 nm.

In this demonstration, the above listed files will be used to obtain fluorescence intensities in Raman (275 nm) Units (RU_{275}) and QSE. The data will subsequently be converted to Raman (350 nm) Units (RU_{275} or simply RU). If you are following this tutorial to correct your own dataset and have water Raman scans at Ex = 350 nm, then the corrected data will already be in RU units and this last step will not be needed.

3.2 Getting started

There are three stages to creating a corrected EEM dataset from raw data files.

i. Import the raw data files
ii. Reconcile the various datasets
iii. Correct the EEMs

The steps for implementing this will described in this section, and interspersed with MATLAB code. For clarity, only part of the code is reproduced in this document. For convenience, the full code listing the steps described in this part of the tutorial can be found in *drEEM_demo* which is located in the *tutorial* folder.

If you originally placed the toolbox in a MATLAB subfolder, it will probably be write-protected. So that you can run the tutorial and write new datasets to disk, make a copy of just the demonstration dataset and tutorial code (i.e. the subfolders called '*demo'* and '*tutorials'*) and place them in a writeable folder somewhere else on your computer.

Assign the location of the demonstration dataset to a variable called demopath.

For example:

```
demopath= 'C:/MyData/MyMatlabStuff/MydrEEM/demo'
```

The drEEM toolbox includes the following functions to assemble corrected 3-way EEM datasets from the raw data files.

assembledataset	ramanintegrationrange	undilute
readineems	eemview	
readinscans	matchsamples	
fdomcorr	lookforconflicts	

A list and brief description of all routines in the drEEM toolbox is obtained by typing:

```
help drEEM
```

To obtain detailed help on any particular function in the toolbox, type "help" followed by the name of the function.

Note that some drEEM functions produce plots, which may be time consuming especially when analysing large datasets. For many such functions, plotting is optional. To cancel a function before it has run to completion, press CTRL + C.

3.3 Import the raw data files

3.3.1 Sample EEMs

Start by loading the sample EEMs. First make the *EEMs* subfolder in *demo* your current directory.

```
cd([demopath '/EEMs'])
```

We will now read in each EEM individually and assemble them into a 3-dimensional matrix using *readineems*. To obtain help on this function, type:

```
help readineems
```

As indicated in the help text, we need to specify input parameters that describe the type and arrangement of data in each EEM.

The files that will be imported are in .csv format, with data in cells A1 - AU105 (when opened using Excel). They look like the example below, with emission wavelengths in the first column, excitation wavelengths in the first row, and fluorescence in raw machine units.

	230	235	240	245	250	…
290	102227.09	54826.976	22455.805	12701.144	8099.207	…
294	95372.82	66821.293	29526.379	14523.594	11057.734	…
298	90069.495	68948.398	28893.105	17609.004	12116.621	…
302	112713.33	72283.838	34583.855	21498.965	17247.363	…
…	…	…	…	…	…	…

Therefore, for our EEM files, the parameters we need are:

```
filetype=1; ext = 'csv'; RangeIn='A1..AU105'; headers=[1 1];
```

We can decide whether or not to display each EEM as it is loaded and whether to save the data to file.

To speed things up, do not display each EEM: `display_opt=0;`

To save the output to disk: `outdat=1;`

Now we are ready to run *readineems* by entering the following as a single line of code.

```
[X,Emmat,Exmat,filelist_eem,outdata]=readineems(filetype,ext,RangeIn,headers,displa
y_opt,outdat);
```

The EEMs have been assembled into a matrix called X. By typing `size(X)` we see that X contains 224 samples, 99 emission wavelengths and 46 excitation wavelengths.

The filenames of the 224 EEMs loaded can be viewed by typing:

```
filelist_eem
```

Emmat and Exmat contain the Em and Ex headers. Since the rows of Emmat and Exmat are each identical, we can extract one row of excitation wavelengths (230:5:450 nm) and one column of emission wavelengths (290:4:682 nm) that apply to all samples:

```
Ex=Exmat(1,:);
Em=Emmat(:,1);
```

3.3.2 Blank (milliQ) EEMs

The procedure for loading the blank EEMs is analogous to what was done in Section 4.1 for the EEMs. First, change the current directory:

```
cd([demopath '/BlankEEMs'])
```

Now load the blank EEMs. Observe that new names are used here to distinguish the output variables from those created in Section 2.1.1.

```
[X_b,Emmat_b,Exmat_b,filelist_b,outdata_b]=readineems(filetype,ext,RangeIn,headers,
display_opt,outdat);
Exb=Exmat_b(1,:);
Emb=Emmat_b(:,1);
```

3.3.3 Water Raman Scans

Now import scans of clean water Raman peaks obtained on the same days as the samples. If no water Raman scans were collected, instructions are given at the end of this section for extracting scans from blanks:

```
cd([demopath '/Raman275'])
```

The water raman files are in csv format with emission wavelengths in the first column and fluorescence in the second column. They look something like this:

	0
285	6435.0542
286	6493.5874
287	6261.4917
288	5620.1382
289	6430.1496
290	6745.4675
...	...

To load univariate data (in two columns) rather than EEMs, use *readinscans*. See the help for full details. In the current case, emission wavelengths are in the first column and fluorescence in the second column. Data (excluding headers) are in cells A2-B167. Therefore the input variables are as following:

```
filetype='R275'; ext='csv'; RangeIn='A2..B167';
```

Choose whether to display each scan as it is loaded and/or save the data.

```
display_opt=1; outdat=1;
```

To generate a matrix of Raman scans called S_R, use:

```
[S_R,W_R,wave_R,filelist_R]=readinscans(filetype,ext,RangeIn,display_opt,outdat);
```

If no water Raman scans were collected, extract them from the Milli-Q blanks as follows:

```
W = [Emb'; squeeze(B(:,:,Exb==350))];
```

Note some important caveats. The resolution of the emission scan directly impacts the area under the Raman peak and hence the accuracy of the normalisation step. It is much better to collect separate high-resolution (every 0.5-1 nm on emission) Raman scans than rely upon a scan extracted from a low resolution blank. This is particularly the case when correcting data from less sensitive fluorometers (e.g. Cary Eclipse, Perkin Elmer). Also, make absolutely certain that the scan settings (e.g. Voltage, scan speed, integration times etc.) used are identical for the sample, blank and Raman scans, or risk ending up with serious scaling errors.

3.3.4 Absorbance Scans

Absorbance measurements can be used to correct the EEMs for inner filter effects according to the method described by Lackowicz (2006). The procedure for importing an Absorbance dataset is analogous to that demonstrated in 3.3.3.

To generate a matrix of Absorbance scans called S_abs, use:

```
cd([demopath '/Abs1cm'])

filetype='Abs'; ext='csv'; RangeIn='A1..B521';
display_opt=0; outdat=1;

[S_abs,W_abs,wave_abs,filelist_abs]=readinscans(filetype,ext,RangeIn,display_opt,ou
tdat);
```

3.3.5 Spectral correction files

The spectral correction factors are stored in *.csv files. They can be read in using MATLAB's built-in csvread.m function.

```
cd([demopath '/CorrectionFiles'])

Excor=csvread('xc06se06n.csv');
Emcor=csvread('mcorrs_4nm.csv');
```

3.3.6 Quinine sulfate dilution series

Fluorescence intensities in each EEM will be calibrated to the average slope of three separate dilution series made from quinine sulfate dissolved in acid, and measured at Ex/Em = 350/450 nm. This step is optional and can be omitted if the intention is to present data only in Raman Units.

The uncorrected slopes will be read from a sample log in step 2.1.7. They will be normalised to the area of the Raman peak in acid blanks from each dilution series. In this step, we will load the water Raman scans at Ex=275 nm and Ex = 350 nm into separate datasets.

```
cd([demopath '/QS/Raman275'])

filetype='QSuv275'; ext='csv'; RangeIn='A2..B75';
display_opt=1; outdat=1;

[S_qsuv275,W_qsuv275,wave_qsuv275,filelist_qsuv275]=readinscans(filetype,ext,RangeI
n,display_opt,outdat);
```

and

```
cd([demopath '/QS/Raman350'])

filetype='QSuv350'; ext='csv'; RangeIn='A2..B106';
display_opt=1; outdat=1;
```

```
[S_qsuv350,W_qsuv350,wave_qsuv350,filelist_qsuv350]=readinscans(filetype,ext,RangeI
n,display_opt,outdat);
```

3.3.7 Sample Log

A sample log will be used to specify the relationships between each EEM and the various other imported datasets, as well as any other numbers that may be important for obtaining corrected data, e.g. dilution factors, slopes of dilution series, etc.

Use MATLAB's built-in *xlsread* function to read the number columns of the log to LogNUM, and the text columns to LogTXT.

```
cd(demopath)
[LogNUM,LogTXT]=xlsread('SampleLog_PortSurveyDemo.xlsx','SF-6-P');
```

Read text from the worksheet 'SF-6-P' by specifying the appropriate columns of LogTXT, for example:

```
Log_EEMfile=LogTXT(:,7);Log_ABSfile=LogTXT(:,8);
Log_Blkfile=LogTXT(:,9);Log_RAMfile=LogTXT(:,10);
etc.
```

Numbers to be read in directly from the log must be extracted from the appropriate columns of LogNUM.

```
QSdata=LogNUM(:,13);
```

View the data imported in this step to confirm that each variable contains the right column of data.

This completes the data importation stage of the EEM correction process.

3.4 Reconcile the various datasets

Once all of the necessary data are in the workspace, it is necessary to define how the EEMs imported in step 2.1.1 correspond to each of the other datasets. EEMs and scans are matched according to instructions obtained from the sample log. The procedure is to (A) define how samples are paired, then (B) create new datasets containing matching samples.

EEMs are paired with their corresponding absorbance scans and Raman scans like this:

```
Pair_EEM_abs=[Log_EEMfile Log_ABSfile];
Pair_EEM_R=[Log_EEMfile Log_RAMfile];
etc.
```

The procedure is slightly different if numbers are to be obtained directly from the log (e.g. the slopes of the QS dilution series, dilution factors, etc.). For all such **number** data, pair them with EEMs like this:

```
Pair_EEM_log=[Log_EEMfile Log_EEMfile];
```

The matching is implemented using the function *matchsamples*. For example, to create an absorbance dataset Sabs containing only absorbance scans from S_abs that correspond with the EEMs in X, use:

```
Sabs=matchsamples(filelist_eem,filelist_abs,Pair_EEM_abs,X,S_abs);
```

Numbers in the sample log can also be matched. The dilution series slopes are stored in the variable that is named QSdata (see 2.1.7). They are matched with the EEMs stored in X like this:

```
eemsinlog=Log_EEMfile(2:end,:);
Sqs=matchsamples(filelist_eem,eemsinlog,Pair_EEM_log,X,QSdata);
```

Any other list of numbers imported from the log can be similarly matched with the EEMs, as specified by the final variable in this last line of code. Note that when using *matchsamples*, there must be an entry in the log for all of the samples in filelist_eem, but for the remaining datasets (e.g. Abs, Raman) files that are not listed in the Sample Log are simply ignored.

This completes the data reconciliation stage of the EEM correction process.

3.5 Correct the EEMs

The final stage is to use the reconciled datasets to correct the EEMs using *fdomcorrect*. As described in the help for this function, there are a number of different options available in terms of number and type of corrections that may be applied. As well as correcting for spectral bias, background signals and inner filter effects, each EEM is normalised to the area of a clean water Raman peak measured on the same day, in order to compensate for fluctuations in lamp intensity during the analytical period. If desired, the data are further calibrated to the slope of a Quinine Sulfate dilution series producing data in QSE units.

Several alternative methods for obtaining data in RU, RU_{275} and QSE using the demonstration dataset are as follows:

(1) Produce data in units of QSE and RU_{275} using daily Raman scans that tracked lamp variability. This is the most accurate available method for obtaining QSE intensities for this dataset; however, for this particular dataset the Raman-normalised fluorescence intensities will not be in the preferred RU_{350} unit. Had Raman scans been measured at Ex=350 nm rather than Ex=275 nm, then this method would instead produce data in RU_{350} units.

(2) Produce data in QSE and RU (RU_{350}) using 350 nm Raman scans extracted from the sample blanks. Since in this study sample blanks were collected infrequently and at relatively low emission wavelength resolution (every 4 nm), fluorescence intensities calculated by this method are relatively imprecise. However, from this process we will obtain QS/RU – a conversion factor between RU and QSE units. This factor should always be reported when available (Murphy et al. 2010).

(3) Combine methods (1) and (2) to get the best datasets in each RU and QSE units.

In methods 1 and 2, the basic command for calling *fdomcorrect* is the same.

For example:

```
fdomcorrect(X,Ex,Em,Emcor,Excor,W,RamOpt,A,B,[],Q,Qw);
```

The difference between methods 1 and 2 lies in the values of three variables: W, RamOpt and Qw. W contains the Raman scans for each sample, RamOpt specifies the extent and postition of the Raman scatter peak, and Qw contains the water Raman scans applicable to the quinine sulfate dilution series. RamOpt can be calculated for a particular dataset as described below, or in the case where the Raman scan was obtained at 350 nm excitation, the default value can be used which is applicable for a Raman peak centered at ~397 nm.

For method 1, recall that the Raman scans were collected at 275 nm and an appropriate peak integration range must be calculated. The 275 nm Raman peak appears at ~303 nm. Use the function called *ramanintegrationrange* to determine the range of emission wavelengths that best capture the 275 nm Raman peak, and hence the parameters needed for RamOpt.

```
[IR,IRmed,IRdiff] = ramanintegrationrange(RamMat,filelist_R,275,1800,6,0);
```

This reveals that the optimal wavelength range for integrating the 275/303 nm Raman peaks in our dataset is 288-320 nm. We are restricted in practice to using a range of 290-320 nm, because there are no emission correction factors available at wavelengths below 290 nm.

```
RamOpt=[275 290 320];
W_in=W(:,wave_R>=290);
Qw_in=Qw275(:,wave_qsuv275>=290);
[XcRU1, Arp1, IFCmat1, BcRU1, XcQS1, QS_RU1]
fdomcorrect(X_in,Ex,Em_in,Emcor,Excor,W_in,RamOpt,A,B_in,[],Q,Qw_in);
```

For method 2, extract Raman scans from the blank EEMs to produce data in RU and QSE. For reasons that were mentioned previously, fluorescence intensities calculated by this method may be imprecise for some samples. However, this method should produce accurate conversion factors between RU_{350} and QSE for the three dilution series.

```
W = [Emb'; squeeze(B(:,:,Exb==350))];    %Extracted scans
Qw=Qw350;
RamOpt=[];                               % default Raman integration range
[XcRU2, Arp2, IFCmat2, BcRU2, XcQS2,
QS_RU2]=fdomcorrect(X_in,Ex,Em_in,Emcor,Excor,W,RamOpt,A,B_in,[],Q,Qw);
QSonRU=mean(unique(QS_RU2));
```

Method 3 combines methods 1 and 2 to obtain the most accurate possible datasets in each QSE and RU. The most accurate QSE data are obtained by method 1. Method 2 produces accurate conversion

factors from QSE to RU. Therefore, to obtain intensities in RU, divide the QSE data obtained in (1) by the QS/RU conversion factor obtained in (2).

```
XcQS=XcQS1;         %Method 1
XcRU=XcQS1/QSonRU; %Method 1 and 2
```

The corrected datasets generated by method 1 and 2 are stored in the MATLAB workspace and written to separate subfolders in the demo/Corrected_EEMs folder. They are named XcRU1 and XcQSE1 (method 1) and XcRU2, and XcQSE2 (method 2).

In summary, the 'best available' corrected datasets that will be used in the remainder of this tutorial are:

1. XcQS: This was obtained using method 1, which involved the closest monitoring of the daily variations in lamp output
2. QSonRU: This conversion between QSE and RU is the mean value of QS_RU (88.96-90.05) obtained for three dilution series using method 2.
3. XcRU: This was obtained by dividing XcQS by QSonRU

Copies of the corrected datasets (XcQS, XcRU) and other relevant metadata (sample names, sites, analysis dates, ….) are found in the folder: demo/Corrected_EEMs/Backup.

If wanting to correct for samples that were diluted, use *undilute*. First make sure that the list of dilution factors applied corresponds with the order of EEMs in XcQS, which may be different from the order of samples in the Sample Log. None of the samples in the demonstration dataset were diluted (dilution factors in the Sample Log were all 1), so this step will have no effect and is shown only as a demonstration.

```
XcQS_df1=undilute(XcQS,dilfac);
```

End this section by saving the workspace or individual datasets as required.

4 PARAFAC analysis using drEEM and the *N-way* toolbox

4.1 Getting started

The drEEM toolbox includes the following functions directed toward dataset manipulation, PARAFAC modeling and visualisation.

assembledataset	loadingsandleverages	specsse
classinfo	lookforconflicts	spectralloadings
compare2models	modelout	splitanalysis
comparespectra	outliertest	splitds
compcorrplot	randinitanal	splitvalidation
describecomp	relcomporder	subdataset
eemview	smootheem	zap
fingerprint	scores2fmax	

Obtain a brief description of all the routines in the drEEM toolbox by typing:

```
help drEEM
```

To obtain detailed help on any particular function in the toolbox, type "help" followed by the name of the function.

For convenience, the full code listing the steps described in this tutorial can be found in the MATLAB script called *drEEM_tutorial* which is located in the demo folder.

4.2 Data Import

To begin the tutorial, load the corrected dataset from the demo folder `'//drEEM/demo/Corrected_EEMs/Backup'`, either from the MATLAB command line or using the import function pull-down menu. This imports eleven variables into your MATLAB workspace.

As a first step, we will assemble the variables we need into a single data structure using *assembledataset* :

```
mydata=
assembledataset(XcQS,Ex,Em_in,'QSE','site',sites,'rep',replicates,'longID',filelist
_eem,'ID',sampleID,'cruise',cruises,'date',dates,[]);
```

Note how the resulting dataset structure includes metadata (information describing the samples in X) in variables named site, rep ID, longID, cruise and date:

```
    Ex: [46x1 double]
    Em: [99x1 double]
```

```
              X: [224x99x46 double]
    IntensityUnit: 'QSE'
              nEx: 46
              nEm: 99
          nSample: 224
             site: {224x1 cell}
              rep: {224x1 cell}
           longID: {224x1 cell}
               ID: {224x1 cell}
           cruise: {224x1 cell}
             date: {224x1 cell}
```

Use *classinfo* to get summary information about the contents of each metadata field.

```
classinfo(mydata)
```

We can visualise the raw data using *eemview*. This function allows us to view contour plots of raw or modelled EEMs in a customised layout and scroll forward or backward through them. View the help for eemview to learn how to use it. It is possible to control the number of plots shown per page (e.g. [3 2] = 6 plots per page in 3 rows and 2 columns) and whether to number plots or label them according to the content of a metadata field (e.g. 'site'):

```
eemview(mydata,'X',[3 3],[],[],'site')
```

4.3 Preprocessing

Appropriate preprocessing is essential to obtaining reliable models. There are two main goals with preprocessing: (1) correct any systematic biases in the dataset, and (2) remove data likely to harm the model while retaining as much useful data as possible. Unfortunately, the best way to preprocess a dataset is not usually obvious from the outset, such that it is often necessary to iterate the preprocessing and modelling steps in order to arrive at a stable and satisfactory PARAFAC solution. This process is illustrated in Figure 2 of the tutorial paper.

When preprocessing a dataset with drEEM, it is recommended that preprocessing steps are implemented in the following order:

1. Resize the dataset to exclude noisy or contaminated Ex or Em wavelengths (if necessary).
2. Remove scatter peaks and parts of EEMs (e.g. using zap)
3. Remove outlier samples
4. Normalise the dataset

Although steps (1) and (2) can be switched in order, if smoothing is implemented before noisy data are removed, then the smoothing procedure could incorporate noisy data and be less successful as a result. Outlier data and samples should always be removed as the very last step or immediately prior to (optionally) normalising the dataset. This is for three reasons: (1) changing the Ex or Em wavelength range can change which samples are outliers, (2) if you remove any data from a normalised dataset you

will not be able to reverse the normalisation accurately, (3) to obtain accurate model scores for excluded samples during the Model Export phase (Section 4.6), a full dataset must be available that was preprocessed in exactly the same way as the modelled dataset.

The plots just displayed using *eemview* show non-trilinear variation (appearing as diagonal peaks) due to primary and secondary Raman and Rayleigh-Tyndall scatter. Before dealing with this, use *subdataset* to get rid of parts of the EEM that have more scatter than signal (Em>600) or that are noisy and/or likely to exert disproportionate leverage on the model (Ex<250).

```
SubData=subdataset(mydata,[],mydata.Em>600,mydata.Ex<250)
```

SubData has the same content as mydata, just with the chosen wavelengths removed. It also has a new field named SubData.i, which contains the original index of each sample. If we remove any samples, this field will track which samples from the original dataset still remain.

Next we will remove the scatter regions using *smootheem*. This function excises the scatter peaks then (optionally) interpolates across them. A number of side-by-side plots are produced by the function to make it possible to tune the input parameters so as to remove only the scatter-affected EEM regions.

View the help for this function to learn how to use it. Notice you have various options about how long (if at all) to display plots. Also, note that the function must be allowed to run to completion (i.e. all plots must be viewed if the plot-viewing option is switched on), otherwise Xs will not be created.

To pause between plotting each sample:

```
Xs=smootheem(SubData,[18 15],[15 15],[17 18],[19 18],'pause');
```

Or to smooth the entire dataset without plotting:

```
Xs=smootheem(SubData,[18 15],[15 15],[17 18],[19 18],0);
```

Once satisfied with the scatter removal, use *eemview* again to look for obvious outliers.

```
eemview(Xs,'X',[3 3],[],[],'site')
```

From plots of Xs, it is clear that there are two sites that represent extreme outliers relative to the others: site = ‘’ (no data) and site = ‘0A’. These samples represent field blanks and tributary measurements, respectively, whereas the rest of the samples came from the Bay proper. It was fine that we did not remove them earlier, because having samples with low signals can be useful when tuning the parameters used to locate scatter peaks. However, they are outliers to the Bay dataset which is likely to present problems for PARAFAC, so it is appropriate to remove them now. We can remove them by name using *subdataset*.

```
Xin=subdataset(Xs,{'site','','0A'},[],[]);
```

As an alternative to removing high-leverage samples (which could cause a significant loss of useful information, especially for a small dataset), it is possible to remove parts of samples containing faulty data using *zap*.

For example, sample 192 features a sharp peak in emission scans collected at excitation wavelengths of 345 and 350 nm only, which is characteristics of a fluorometer error. This is especially apparent in comparison to other EEMs (samples 191 and 193) which were collected at the same time and location.

```
eemview(Xin,'X',[1 3],176,[],[],[],[],[],20)
```

Remove just the faulty emission scans using *zap*.

```
Xin=zap(Xin,176,[],[345 350])
```

Confirm this took care of the problem.

```
eemview(Xin,'X',[1 3],176,[],[],[],[],[],20)
```

Notice that Xin now contains a field called Xin.Zap containing the input parameters for this last procedure. This information (together with data tracking other procedures implemented during model preprocessing, development and validation) will be carried forward and ultimately exported to Excel along with the validated model spectra.

4.4 Exploratory data analysis

The goal of exploratory data analysis is to get a feel for a dataset without investing a great deal of time and effort in obtaining the best possible solutions. This is the time to investigate the effects of different data preprocessing, by including or excluding samples and/or wavelengths, or varying the options for dealing with scatter peaks.

To start, use *outliertest* to generate preliminary models with 3 to 6 factors and identify samples that have an unusually high impact on the models. To speed up the modelling, we will exclude every second Ex and Em wavelength. We will create two tests, with and without the non-negativity constraint (which constrains the returned scores and spectra to having positive values).

```
Test1=outliertest(Xin,[2,2],3:7,'nonnegativity',[],'at once');
Test1u=outliertest(Xin,[2,2],3:7,'unconstrained',[],'at once');
```

View the PARAFAC components to check that they look approximately as expected. The non-negativity constraint is necessary to obtain reasonable spectra.

```
comparespectra(Test1,3:7)
comparespectra(Test1u,3:7)
```

Now look at correlations between the components in the various nonnegative models.

```
compcorrplot(Test1,4)
compcorrplot(Test1,5)
compcorrplot(Test1,6)
```

or

```
compcorrplot(Test1,6, 'site')
```

In any model with more than four components, the scores (related to concentrations) of some components are very strongly correlated suggesting that dilution is a dominant mechanism in the dataset. As is discussed in the main article, when the scores of two PARAFAC components are very highly correlated, it is very difficult for PARAFAC to accurately resolve their spectra. We will assist PARAFAC by normalising the dataset, which should greatly reduce the concentration-related colinearity and give low-concentration samples a chance to enter the model. This will help PARAFAC obtain the best possible estimates of each component's excitation and emission spectra. Once we have determined what the spectra should be in the final model, we will project the non-normalised data on the final (normalised) model and obtain the true (non-normalised) model scores.

First, develop a new dataset in which EEM intensities are normalised to unit norm. This will be the dataset modelled using PARAFAC. As the final step before exporting a validated model, we will reverse the normalisation in order to obtain the true scores.

```
Xpre=normeem(Xin)
```

Perform a new outlier test on the normalised data, and view the resulting PARAFAC components.

```
Test1p=outliertest(Xpre,[1,1],4:7,'nonnegativity',[],'at once');
spectralloadings(Test1p,4:7)
```

Compare the spectral loadings of the new model with the old model, for example:

```
compcorrplot(Test1p,6)
```

Now use *loadingsandleverages* to identify any samples or wavelengths that have an unusually high influence on the models. Unusual samples have the potential to introduce unique factors to a model. If they land on one split of our dataset but not another, they may make it difficult to obtain similar models using different fractions of the dataset. Wavelengths with high leverages will have more influence on spectral shapes than other wavelengths, which could distort the results.

```
loadingsandleverages(Test1p,5)
loadingsandleverages(Test1p,6)
loadingsandleverages(Test1p,7)
```

Samples 205 and 208 have high leverages and are clear outliers. Sample 49 and a few others are worth a closer look. High leverage samples are not always problematic – in practice, each should be assessed individually to determine the effect of removing them.

VERY IMPORTANT: If after normalising the dataset you decide to remove samples and/or parts of samples, return to Section 4.3 and repeat the steps taken to generate Xpre, after removing the outliers using the order of preprocessing steps described. In other words, make sure that normalising is the last thing you do to your dataset before you apply PARAFAC.

```
Xin=subdataset(Xin,[205,208],[],[]);
Xpre=normeem(Xin)
Test2=outliertest(Xpre,[2,2],5:7,'nonnegativity',[],'at once');
loadingsandleverages(Test2,5)
loadingsandleverages(Test2,6)
loadingsandleverages(Test2,7)
```

There are still some high leverage samples but none appear to be severe. Leave them in for the moment, or try to improve them using zap.

To view the error residuals:

```
eemview({Test2, Test2},{'X','Model6','error_residuals'},[1 3],49)
```

For example, look at the residuals for sample 7, indicating a problem around Ex=325 nm, and in a large number of samples, a sharp peak is seen at Ex = 280-295 nm and Em= 560-576 nm. This last peak in particular could disturb the modelling because it appears in numerous samples.

To view the residuals for the 5 and 6 component models in adjacent plots:

```
compare2models(Test2,5,6,0.05)
```

It is also possible to compare the residuals for 3 or more models at the same time. See *drEEM_tutorial* or *eemview* for examples.

We can tentatively evaluate the feasibility of the current PARAFAC models. Compare the spectra for the 5- to 7-component models. Do the components all look like fluorophores or are some components being used to model noise?

```
spectralloadings(Test2,5:7)
```

A useful indication of the number of components that should be in the model can be obtained by *specsse*, which shows the effect on model fit of adding more components, expressed as the sum of squared error (SSE) for each model plotted as a function of wavelength.

```
specsse(Test2,5:7)
```

Observe that the 6-component model is noticeably better (lower SSE) than the 5-component model. However, there is little difference in fit between the 6- and 7-component models. It is too soon to

conclusively determine the number of components in the model, because as discussed in the next section, the solution found by N-way's PARAFAC algorithm on any single run is not necessarily correct, particularly if the dataset is difficult to model and the convergence criterion is too lax. However, current indications for the demonstration dataset are that the solution has six components.

If you have arrived at this point in the tutorial using your own data, but have not yet obtained a fair indication of the likely number of components in the model, here are two possible paths forward:

1. Adjust and reiterate the preprocessing and exploratory modelling stages. Examine the model residuals and components for indications that contaminant peaks or residual scatter are affecting the models (e.g. very sharp peaks). Consider either excluding or re-including samples and/or wavelengths. Observe the order of preprocessing steps and start again from the top.
2. The components found by PARAFAC can vary between model runs and according to the criteria used to determine model convergence. Rerun the outlier test to see if the solutions change. If they do, the model is unstable and PARAFAC may settle on various local solutions instead of the global solution. See the tools in the next section for arriving at global solutions.

4.5 Refinement and Validation

Start by developing some new overall models. It is important to realise that PARAFAC analysis of a single dataset may produce variable results due to some procedures within the N-way PARAFAC algorithm involving random numbers. The N-way PARAFAC algorithm can therefore also converge on a local minimum in the data, instead of the true global minimum-error solution. This is especially the case when EEMs have a lot of missing data and/or when constraints are applied, regardless of how the model is initialised. When modelling does not produce a stable solution, it is an indicator that the model may have too many components and/or that more stringent (smaller) convergence criteria are needed.

To increase the chances of locating a robust solution that does not depend on the numbers used to initialise the models, a series models will be developed with each one initialised using a different random starting vector. The best (minimum error) solution will be retained as the overall model. Expect this process to take **much longer** than *outliertest*, depending on the convergence criterion used and the number of iterations requested.

To develop and compare a series of ten constrained 6-component models of a dataset with the default convergence criterion of 10^{-6}:

```
[LSmodel6,convg6,DSit6]=randinitanal(Xpre,6,10,'nonnegativity');
```

Or if applying a stricter convergence criterion, e.g. 10^{-8}:

```
[LSmodel6,convg6,DSit6]=randinitanal(Xpre,6,10,'nonnegativity',1e-8);
```

Also develop 5- and 7-component models. Choosing to make fewer runs will speed things up. Lowering the convergence criterion will slow it down, but may help PARAFAC to reach reasonable solutions. If randinitanal does not consistently produce very similar solutions (similar sum of squared residuals for each run), the model will be especially difficult to validate. See whether the model is made more stable by decreasing the convergence criterion.

```
[LSmodel6,convg6,DSit6]=randinitanal(Xpre,6,5,'nonnegativity',1e-8);
[LSmodel7,convg7,DSit7]=randinitanal(Xpre,7,5,'nonnegativity',1e-8);
```

The outputs of *randinitanal* for the six component model are:

1. LSmodel6: A data structure containing only the model that had the least squares solution. This model can be used to complete the split validation.
2. convg6: sum of squared error (SSE) and number of iterations until convergence for each model.
3. DSit6: A data structure containing results of all the model runs.

In addition, *randinitanal* produces plots of (A) the sum of squared errors and number of iterations for each model, and (B) a core consistency diagnostic plot. See the main article (and references within) for discussion of how to interpret core consistencies.

The core consistency of the six component model is found here:

```
LSmodel6.Model6core
```

To visualise the spectra from the least squares model:

```
spectralloadings(LSmodel6,6)
comparespectra(LSmodel6,6)
fingerprint(LSmodel6,6)
```

Compare the spectra for the components in the least squares model with those in the outlier test. Are they very different? If so, you may need to revisit any assumptions you have made on the basis of plotting the earlier outlier tests. Check the leverages of the least squares model for any last minute surprises and make sure you are satisfied with the residual error plots.

```
loadingsandleverages(LSmodel6,6)
eemview({LSmodel6, LSmodel6},{'X','Model6','error_residuals'},[2 3])
```

If you are working with a dataset that was normalised, before you export the model you will want to reverse the normalisation to obtain the unscaled scores.

```
LSmodel6r=normeem(LSmodel6,'reverse',6)
```

LSmodel6 and LSmodel6r are identical except that the latter contains the true (unscaled) model scores.

4.5.1 Split half analysis

Split half analysis is a very powerful technique for validating a PARAFAC model. However, it is computationally intensive and can take a **very long time**. The dataset is split into several fractions to see if the same model is obtained when modelling different groups of samples. Before attempting this, examine the loadings of the least squares model and consider whether the spectra are reasonable or not. If the spectra do not look reasonable, see if they can be improved by changing the convergence criterion and/or increasing the number of iterations. Also examine plots of the residuals for evidence of outliers or over-fitting. Often, problems that are invisible in the raw data can be very clearly seen in model residuals.

Appendix B to the tutorial which contains a pictorial depiction of split half analysis that may be helpful to refer to during the next part of this tutorial. The drEEM toolbox contains a flexible routine called *splitds* that enables datasets to be divided in a range of different ways. Samples can be split in random groups, in contiguous blocks, or by assigning alternate samples to different groups. Samples can also be split according to metadata categories in order to put e.g. different sites, cruises, etc. into different splits. Also, smaller splits can be combined using *splitds* in order to create larger splits having desirable characteristics.

splitds may seem a bit complicated at first. For help and examples, type:

```
help splitds
```

The function *splitds* can be used to implement the validation method suggested by Stedmon and Bro (2008, *Limnol Oceanogr Meth* 6, 572-579). This involves an alternating split style, in which starting from the first sample in the dataset each sample is assigned alternately to one of four splits, followed by a combination procedure where the four splits each containing a quarter of the dataset are combined in four different ways to produce four new 'halves' of the dataset. We will refer to this style of validation as '$S_4C_4T_2$' (Splits -4, Combinations -4, Tests -2). Here, we extend this method in order to assemble six different dataset 'halves' and produce three validation tests '$S_4C_6T_3$'.

```
S1=splitds(Xpre,[],4,'alternating',{[1 2],[3 4],[1 3],[2 4],[1 4],[2 3]})
```

In Appendix B to the tutorial, the dataset halves (split combinations) indicated by numbers in the above code are depicted using the letters A-D, with A=1, B=2, C=3, etc. So combination [1 2] is equivalent to combination [AB].

S1 now includes the original dataset, the six new datasets in S1.Split, and several new variables that document the splitting procedures used, i.e.:

```
                  Split: [1x6 struct]
    Split_NumBeforeCombine: 4
               Split_Style: 'alternating then combine'
     Split_NumAfterCombine: 6
        Split_Combinations: {'1 2' '3 4' '1 3' '2 4' '1 4' '2 3'}
             Split_nSample: [104 102 103 103 103 103]
```

Now use *splitanalysis* to generate separate 5- and 6-component PARAFAC models in each dataset split.

```
A1=splitanalysis(S1,5:6,'nonnegativity');
```

The next step is to perform a preliminary validation to see which split models are the same. In three tests we will compare the 1st and 2nd split-combinations (AB vs. CD), the 3rd and 4th (AC vs. BD) and the 5th and 6th combinations (AD vs. BC). Notice that by selecting these particular combinations, we ensure that in each test the dataset halves being compared have no samples in common.

```
splitvalidation(A1,5,[1 2;3 4;5 6],{'AB','CD','AC','BD','AD','BC'});
splitvalidation(A1,6,[1 2;3 4;5 6],{'AB','CD','AC','BD','AD','BC'});
```

If a model validates with *splitvalidation* it means that all of the components in the split models being compared in each test have found a match with Tucker correlation coefficient > 0.95. In this case, a message will appear requesting you complete the validation by specifying the overall least squares model. Please ignore the message for now, as this step will be done below.

Before we move on, note that it is also possible to use *splitvalidation* to compare all possible combinations of splits:

```
splitvalidation(A1,6);
```

The above line of code invokes the default option where every split is compared with each of the others. But be careful and don't lose sight of which samples went in to each split! You can NOT validate a model by comparing dataset halves that have samples in common (e.g. split combinations [AC] vs. [CD] or [BD] vs. [BC]).

If you have finished performing the split validation, you will probably have found that one or both models did not validate. In fact, if you try running *splitanalysis* several times, you may find that the solution is very unstable (it changes each time you run the analysis). When models do not validate but come close to doing so (e.g. some of the tests succeed but not others), *splitanalysis* can be run with the option to develop multiple models (keeping only the best one) and/or apply a different convergence criterion (the default is 10^{-6}). Note that depending on how many splits there are in the dataset, on the number of runs and the convergence criterion selected, this could take a long time. To cut down on processing time, specify more runs and/or lower convergence criteria only when modelling 'difficult' splits (those that are least stable or most unlike the others). In each case, only the solution with the lowest error will appear in the output.

For example, run the 5-component PARAFAC analysis five times, using a convergence criterion of 10^{-8}:

```
MyRunOptions=5
MyCC=1e-8
SaveResultsAs='A2_10-8_5comp'
A2a=splitanalysis(S1,5,'nonnegativity',MyRunOptions,MyCC,SaveResultsAs);
```

The 6-component PARAFAC solution is more stable. Run the analysis more times on difficult splits.

```
MyRunOptions=[5 5 3 3 3 3]
MyCC=[1e-8 1e-6 1e-6 1e-6 1e-6 1e-6]
SaveResultsAs='A2_10-6_6comp'
A2b=splitanalysis(S1,6,'nonnegativity',MyRunOptions,MyCC,SaveResultsAs);
```

Try the validation again:

```
splitvalidation(A2a,5);
splitvalidation(A2b,6);
```

If this worked, you are ready to perform the 5- and 6-component model validations a final time, using the overall (least squares) model as the final input variable in *splitvalidation*. This time, the function produces validation plots and retrieves the spectral correlation coefficients for each split against the overall model.

Note that **the validation dataset obtains scores from the overall model**. Therefore, if LSmodel6 was derived from a normalised dataset (e.g. Xpre), use the reversed model (i.e. LSmodel6r) when performing the validation. This will ensure that val6 contains the true model scores.

```
val5=splitvalidation(A2a,5,[1 2;3 4;5 6],{'AB','CD','AC','BD','AD','BC'},LSmodel5r);
val6=splitvalidation(A2b,6,[1 2;3 4;5 6],{'AB','CD','AC','BD','AD','BC'},LSmodel6r);
```

The contents of each validated model include fields tracking the results of the validation tests and the procedures used to obtain the validation. For example, val6 contains something similar to this:

```
          Val_ModelName: 'Model6'
         Val_Preprocess: 'Reversed normalisation to recover true scores'
             Val_Source: 'Model6it_5'
                Val_Err: 322.7535
                 Val_It: 692
               Val_Core: 2.2518
          Val_ConvgCrit: 1.0000e-06
        Val_Constraints: 'nonnegativity'
         Val_Initialise: 'random'
        Val_PercentExpl: 99.9713
           Val_CompSize: [69.9076 28.4569 29.3479 11.8407 10.0141 3.8042]
             Val_Result: 'Overall Result= Validated for all comparisons'
        Val_Comparisons: {'AB vs CD,'  'AC vs BD,'  'AD vs BC,'}
    Val_Comparisons_Num: [3x2 double]
           Val_Matches: {4x1 cell}
              Val_ExCC: {4x1 cell}
              Val_EmCC: {4x1 cell}
            Val_Splits: {'AB'  'CD'  'AC'  'BD'  'AD'  'BC'}
         Val_SplitsNum: [1 2 3 4 5 6]
```

4.6 Reporting and exporting models to Excel

PARAFAC components are often reported in terms of their peak positions. These can be calculated using *describecomp*.

For example:

```
describecomp(LSmodel6,6)
```

To convert raw PARAFAC model scores to F_{max}, representing the maximum intensity of each component in each sample in the same scale as the original EEMs, use *scores2fmax*.

```
[F6,scores6]=scores2fmax(val6,6)
```

Model results can be exported to Microsoft Excel using *modelout*. This will by default export the spectra of the chosen model, along with all the tracking information (information on data preprocessing, modelling criteria, and validation results) that are contained in the model structure being exported. It does not export the raw PARAFAC model scores, but calculates and exports F_{max} (the maximum intensities of the components in the original measurement scale).

For a basic export of the val6 model:

```
modelout(val6,6,'Tutorial_Models.xls');
```

There are two additional options for exporting models:

Option (1) Project a larger dataset on the model to obtain Fmax for all the samples in the larger dataset. This can be used to obtain scores for samples which were excluded (outliers) when building the model.

To obtain Fmax and loadings for all the samples:

```
[F,B,C,Ff,P6]=modelout(val6,6, 'Tutorial_Models.xls',Xs);
```

If projecting a dataset, beware of some caveats. First, it is essential that the new dataset differs from the modelled dataset only in the number of samples it contains. If it has different Ex or Em wavelengths, the projection will fail. Also, if it differs in the way it was preprocessed for scatter, the projection may fail or produce inaccurate results. Check in the Excel spreadsheet that Fmax for samples that were included in the model is consistent between the sheets titled ‘Model6Loading’ and ‘Model6FmaxProjected’. Second, even if the modelling is successful, Fmax may be inaccurate for outlier samples because the model is inappropriate. This is possibly the reason why they had to be excluded from modelling in the first place. Extra care must be taken when interpreting the scores for outlier samples.

Look at the fit for the projected dataset. It is very poor for many of the outliers!

```
eemview({P6,P6},{'X','Model5_P','error_residuals'},[3 3],[],[],[],[1 1 0.05]);
```

Option (2): Export metadata for the samples included in building the model.

For example, to export the metadata for val6:

```
metadata4export={'cruise','site','rep','longID','ID','date','i'}
modelout(val6,6,'PortSurvey_Models.xls',Xs,metadata4export);
```

Finally, note that when the *modelout* function fails to locate some tracking information, a warning is produced. Such warnings are for informational purposes, and do not necessarily indicate there was a problem with either the analysis or export. Many of the drEEM functions produce some tracking information that is carried forward to the validated model structure, so if any of those functions were omitted when creating a particular model (or if the model being exported was not validated using drEEM), their tracking information will be absent.

4.7 Incorporating hypothesis testing into model validation

Earlier in this tutorial, the 6-component model was validated using a straightforward alternating $S_4C_6T_3$ validation. However, there are many other ways that the model could be validated, and testing these may provide new insights regarding the usefulness and interpretation of the model.

It is often assumed that the best way to split a dataset is via a random process. Consider, however, that if dataset is split such that dissimilar samples are deliberately assigned to different splits, it should be harder to validate because the splits are less similar than they would be if samples were grouped randomly, or evenly as when alternating splits are created from a dataset that happens to be ordered in space or time. On the other hand, when identical models are obtained from different non-random and non-even splits, this provides much stronger evidence of the robustness of the model.

Here we will attempt two alternative validations of the 6-component PARAFAC model.

(A) Split the dataset as for a $S_4C_6T_3$ validation but keep replicate samples (collected at the same site on the same cruise) together in the same split.

```
newdata=splitds(Xin2,'cruise.site',[],'exact');
```

The number of unique combinations of cruise/site in the dataset is stored in newdata.Split_NumBeforeCombine

```
S=newdata.Split_NumBeforeCombine;
```

Create a series of alternating splits, where samples are in order assigned to one of four groups.

```
scomb={[1:4:S 2:4:S],[3:4:S 4:4:S],[1:4:S 3:4:S],[2:4:S 4:4:S],[1:4:S 4:4:S],[2:4:S
3:4:S]};
```

Combine the groups into six splits in preparation for three validation tests.

```
newdata=splitds(newdata,[],[],'combine',scomb);
```

Confirm that replicate samples were kept together in each Split (n=1:6) as intended.

```
[newdata.Split(1).cruise newdata.Split(1).site newdata.Split(1).rep]
```

Run the split analysis and validate the model.

```
cs6=splitanalysis(cruisesitecomb,6,'nonnegativity',5,[],'cs6');
mysplitnames={'1,2','3,4','1,3','2,4','1,4','2,3'};
vcs6=splitvalidation(cs6,6,[1 2;3 4;5 6],mysplitnames,LSmodel6);
```

(B) Split the data by cruise to see if each cruise dataset produces the same model.

```
cruisedata=splitds(Xin2,'cruise',[],'exact');
```

Confirm that Split n (n=1:4) contains only cruise n samples:

```
cruisedata.Split(1).cruise
```

A potential issue for modelling each cruise separately is that the four cruises each have very different sample sizes (n = 33, 35, 68, and 69).

```
cruisedata.Split_nSample
```

We therefore construct all possible combinations of cruises.

```
cruisecomb=splitds(cruisedata,[],[],'combine',{[1 2],[3 4],[1 3],[2 4],[1 4],[2
3]})
```

Now attempt to validate this model as previously shown.

```
mysplitnames={'fall+spr','sum+win','fall+sum','spr+win','fall+win','spr+sum'};
cc6=splitanalysis(cruisecomb,6,'nonnegativity',5,1e-8);
vcc6=splitvalidation(cc6,6,[1 2;3 4;5 6],{);
```

4.8 Troubleshooting split-validation failures

4.8.1 Extended plotting of split models

When split validation fails, it can be useful to view overlaid plots of components in different split models. For example, consider the combined-cruise model (vcc6).

First, take a closer look at the validation results. Results for the t^{th} test are located in vcc6.Val_Matches{t} and are reproduced in the matrix below (remember that individual results may vary). The matrix contains numbers indicating the components that were matched, and NaNs, representing components that were not correlated at p>0.95. For example, the first row of data indicates that in the first test (Split 1 vs Split 2) the fifth component in Split 1 did not find a match.

```
     1     2     3     4   NaN     5
     3     1     2     4     5     6
     1     2     3     4     6     5
```

To visualise this, we can use *spectralloadings* to plot the overlaid Split models.

```
spectralloadings(vcc6,6,1:6)
```

The plots are hard to interpret because the order that PARAFAC locates various components varies between splits. The way to fix this is to specify the order of components in each split, which can be calculated with the aid of *relcomporder*. First, however, we have to generate comparisons of every split model against every other model. Fortunately, this will be fast, and we can ignore/close the two plot windows generated in this step.

```
allcc6=splitvalidation(cruisecombmod,6)
```

Now determine the order of components in each model relative to the model in Split 1 using the function *relcomporder*.

```
r=relcomporder(allcc6,1)
```

This yields:

```
r =

     1     2     3     4     5     6
     1     2     3     4     6   NaN
     1     2     3     4     6   NaN
   NaN     3   NaN     4     6     5
     1     2     3     4     5     6
     1     2     3     4     6   NaN
```

It is necessary to replace the NaNs with component numbers using deductive reasoning or trial and error.

```
r(2,6)=5; r(3,6)=5; r(4,1)=2; r(4,3)=1; r(6,6)=5;
```

Then we can produce overlaid plots of the various split models, while correctly matching up the components.

```
spectralloadings(vcc6,6,1:6,r)
```

How many samples are in each split? Is sample size affecting the shapes of components?

```
vcc6.Split_nSample
```

4.8.2 Locating outliers in split models

Split models can be explored and optimised in exactly the same way as the main dataset model.

For example:

```
Split1mod= cruisecombmod.Split(1)
```

Use *loadingsandleverages* to investigate outlier influences upon particular split models.

```
loadingsandleverages(Split1mod,6)
```

It is possible to examine the residuals for just the 6-component model in Split 1:

```
eemview({Split1mod, Split1mod},{'X','Model6','error_residuals'},[2 3]);
```

Does removing the outlier sample 99 improve the six-component model for this Split?

```
Split1mod_R=subdataset(Split1mod,99,[],[]);
LS_Split1mod_R=randinitanal(Split1mod_R,6,5,'nonnegativity',1e-8);

spectralloadings(Split1mod,6)
spectralloadings(LS_Split1mod_R,6)
```

Get summary information for the split 4 combined cruise model:

```
classinfo(vcc6.Split(4)):
```