User manual of the Coincidence data analysis software (ANACONDA 2)

Bart Oostenrijk, Benjamin Bolling, Lisa Rämisch

December 7, 2018

Contents

1 Introduction

This document shows how the data analysis package 'ANACONDA 2', or 'ANACONDA' can be installed and used, with the focus on new users that want to work with the command-line interface (CLI). ANACONDA allows the analysis of data recorded in experiments that use single-particle detectors (e.g. ions, electrons, photons ...), for example to study the correlation between these particles. These are often called 'coincidence' spectrocopic treatment methods. For more information of the content of the analysis package, the reader is referred to the CLI-description document.

2 Installation

End user installation:

- Download (or 'clone') the file 'ANACONDA.mltbx'
- Drag and drop the file into the matlab (command) window, and click install in the pop-up window.
- A gettingstarted (example) file can be found in the '/doc/' directory.

Developer installation:

- Copy the 'package' folder (included in the repository) to your local system.
- Add this folder to the path (only the folder, not the subfolders), either by the command:
  - addpath(fullfile('path','to','folder', ..., 'package'))
  - right-click on the folder in the file browser in MATLAB, and click 'add to path'.
- You are now ready to use all functions within the package. Note that the path might not be added automatically, every time you start up MATLAB. Click here for more help.

3 Getting started

The matlab toolbox comes with an interactive script (*/package/doc/GettingStarted.mlx*), or a plain matlab script (*/package/doc/GettingStarted_plain.m*) that shows a few simple commands and minimal working examples of functions and macros (=several functions bunched together).

Load first data After installing the package, one can start importing data. There are several data formats supported by the package. Depending on which format you want to treat, read one of the following functions by typing into the MATLAB command window:

```
>> edit IO.COBOLD.import_example ;% importing ASCII delimited
    datafiles from COBOLD PC. If you do not have the appropriate
    ASCII-delimited file format, contact Roentdek.
>> edit IO.DLT2ANA.import_example ;% importing the DLT data format
    from the Labview data acquisition software (by Erik Mansson, Lund
    University)
>> edit IO.EPICEA.import_example ;% Importing ASCII delimited
    datafiles from EPICEA. If you do not have the appropriate
    ASCII-delimited file format, contact the EPICEA (or PLEIADES
    beamline) experts.
```

At the end of importing the data, it is advised to save the data to a '.mat' (MATLAB) format. For example, in the case of the COBOLD import, one could write:

```
data = IO.COBOLD.import_example('file/to/path', 'filename'); %
    import the data
IO.save_exp(data, 'file/to/path', 'filename'); % save it to a binary
    '.mat' file
```

Create first metadata The data has now been loaded to memory, but in order to perform meaningful analysis, the package requires metadata as well. Metadata is literally 'data about the data'. For example, we need to know what the 'raw' data contains. This kind of information can be stored in a separate 'metadata' file. Each datafile must contain such a file (habitually in the same folder as the data, with the same name except the prefix 'md_' and extension '.m'). So, create a metadata called 'md_filename.m' with the file to the datafile that is called 'filename.mat'. This file 'md_filename.m' could look like this:

```
% Metadata file for 'filename'.
%% Defaults
% The default values are loaded from the package:
exp_md = metadata.defaults.exp.CIEL.md_all_defaults();
%% Custom values
% Here, we can customise these default values.
% For example, the mass-2-charge conversion factor:
exp_md.conv.det1.m2q.factor = 2500;
```

The metadata file is read as a matlab script when it is read, so all MATLAB functionalities are available. The name of the metadata struct has to be 'exp_md'.
After this file is written, the data and metadata can be loaded to memory by typing:

```
data  = IO.import_data('file/to/path', 'filename');%load data
mdata = IO.import_metadata('file/to/path', 'filename');%load metadata
```

4 Metadata

The metadata contains all the information *about* the data, such as conversion parameters, but also filter and plot parameters. The package comes shipped with 'default' parameters, which can be used when the data is first imported by the user. We have seen in the previous section how to start a first metadata file from those defaults. This section explains the different fields in the metadata, and what they can be used for. The metadata can be divided into different categories:

sample , e.g. atomic mass, expected fragment masses, constituent masses.

photon beam , e.g. the photon energy, intensity, duration, etc

spectrometer , e.g. the name, voltages and relevant dimensions of the used spectrometer are listed here.

detectors , e.g. the names and properties of the detectors are stored in here.

correct The parameters needed to execute corrections onto the raw data, before conversion. For example, translation in X and Y to move the centre of detection into the origin of the coordinate system.

calibrate The information needed to perform the calibrations. Note that these are not the actual calibration factors, they are stored in the 'convert' field.

fit The fitting parameters.

convert The conversion factors (sorted in terms of detectors), such as mass to charge conversion.

condition Defines conditions on data, that can later be applied in data visualization.

plot The user-preferred plotstyle.

The default values for each spectrometer are shipped with the package. The reader can inspect these by typing in the command window:

```
edit metadata.defaults.exp.‘setup_name’.‘metadata_category’
% For example:
edit metadata.defaults.exp.CIEL.plot
```

We will go through the least obvious categories and clarify the fields used in them.

4.1 Sample metadata

Here, one can provide information about the species under study. In the case of a measurement on single atoms, this is rather trivial. In the case of more complex samples, such as larger clusters of (different types of) molecules, there are more parameters that can help the interpretation. For example, the expected size of (molecular or cluster) fragments, information about the monomer(s) in the cluster, and whether protonation occurs in the formation of fragments. For example, in the case of pure ammonia clusters:

```
exp_md.sample.name              = 'NH$_3$  '; % [a.m.u.] mass of
    constituents of sample
exp_md.sample.type              = 'Cluster'; %sample type, can be
    'Molecule' or 'Cluster'.
exp_md.sample.T                 = 273-11.3; %[K] The temperature of
    the nozzle.
exp_md.sample.constituent.masses  = [17]; % [a.m.u.] mass of
    constituents of sample
exp_md.sample.constituent.names   = {'NH$_3$'}; % [] names of
    constituents of sample
```

```
exp_md.sample.monomer.fragment.masses  = [14 1]; % [a.m.u.] mass of
    components of fragments of sample
exp_md.sample.monomer.fragment.names  = {'N' 'H'}; % [] names of
    components of fragments
exp_md.sample.monomer.fragment.nof    = [1 3]; % [] number of
    components of fragments a single monomer can be composed of.
exp_md.sample.monomer.mass          =
    sum(exp_md.sample.monomer.fragment.masses.*exp_md.sample.monomer.fragment.nof);%
    [a.m.u] mass of sample fragment
exp_md.sample.fragment.protonation= 1;%[-] Protonation of fragments;
    1 means 1 proton attached.
exp_md.sample.fragment.sizes     = (1:20)'; % size is defined as the
    number of monomer units.
exp_md.sample.fragment.pure.masses    = exp_md.sample.fragment.sizes
    * exp_md.sample.constituent.masses +
    exp_md.sample.fragment.protonation*1; % The masses of the pure
    fragments (only relevant for multi-species clusters).
exp_md.sample.fragment.masses       =
    convert.cluster_fragment_masses(exp_md.sample);
```

On top of the sample name and mass, the sample's velocity before the photon interaction is defined here as well. This will later be used in the calculation of momentum:

```
exp_md.sample.m_avg         = mean(exp_md.sample.constituent.masses);
    % [a.m.u.] The average mass of the incoming sample.
exp_md.sample.v_direction     = [3 0 0]; % [] direction of the sample
    supply in X, Y, Z direction.
exp_md.sample.v_MB          = 1.0+03; % [m/s] average speed of the
    molecular beam
```

4.2 Detector metadata

Information of each detector used in the spectrometer is stored here. For example, the names of signals of each column of the detector data output, and the physical limits of the measurement variables. To see an example of detector metadata, execute the following MATLAB command:

```
edit metadata.defaults.exp.EPICEA.det
```

4.3 Correction metadata

The data, as imported, can contain abberations or artefacts that should be removed before converting the data to physically interprable variables. There are several correction routines programmed in the software, and more information can be found in the software description. To see an example of correction metadata, execute the following MATLAB command:

```
edit metadata.defaults.exp.EPICEA.corr
```

4.4 Spectrometer metadata

In order for the kernel to know how to treat the data, it needs to be able to interpret it. For this interpretation, the information about the spectrometer in use is essential. For example, the kernel can read the spectrometer name, and know from that what kind of method is used for the measurement of electrons and/or ions.

Furthermore, the user can specify the applied voltages to spectrometer elements, and the distances between them. Fro example, this can later be used in the conversion from (X, Y, T) to momentum variables. To see an example of spectrometer metadata, execute the following MATLAB command:

```
edit metadata.defaults.exp.EPICEA.spec
```

4.5 Condition metadata

In the condition metadata, the user can specify how to construct a 'filter' parts of the data. These definitions of conditions can later be applied in the construction of a histogram. The possible input fields of a condition are:

- *data_pointer* The 'address' of the signal on which to apply the condition. For example, 'h.det2.TOF', applies the condition to the Time Of Flight signal of detector 2. Note: the pointer can also be a MATLAB expression that combines different signals. For example, if one is interested to apply a condition to the difference in X- and Y-location, one can use 'exp.h.det2.X - exp.h.det2.Y' as a data pointer.

- *type* The type of condition; are you looking for only certain values in a signal ('discrete'), or for a range of values ('continuous');

- *value* If the condition is of discrete type: the values represent the exact value the signal should have. If the condition is of continuous type: the first colum represents the minimum value(s), the second represents the maximum value(s).
- *translate_condition* (only necessary for hit signals), specifies how the user wants to translate the filter from a hit filter to an event filter. Options: If all must be approved, 'AND' must be given for this field. If only one of all hits should be approved for the event to be approved, 'OR' should be given. If only the first hit needs to be approved by the condition, 'hit1' should be given (same for hit2).
- *invert_filter* (optional) logical: if 'true', the event filter will be inverted (true becomes false, false becomes true)

An example of the definition of conditions:

```
% Get rid of peaks in the mass spectrum that originate from oil in
    the vacuum pumps:
def.oil.type                = 'discrete';
def.oil.data_pointer        = 'h.det2.m2q_l';
def.oil.value               = [72; 73];
def.oil.translate_condition = 'AND';
def.oil.invert_filter       = true;
```

4.6 Plot metadata

The plot metadata describes the histogram plots to be shown when the plot macro is run (macro.plot). The metadata of a single plot can be composed by copying different metadata about 'signals'. For example, the metadata called 'i_TOF' describes the ion Time Of Flight signal:

```
%%%%%% ion TOF:
s.i_TOF.hist.pointer  = 'h.det2.TOF';% Data pointer, where the
    signal can be found.
% Histogram metadata:
s.i_TOF.hist.binsize  = 6;% [ns] binsize of the variable.
s.i_TOF.hist.Range    = [3000 10000];% [ns] range of the variable.
% Axes metadata:
s.i_TOF.axes.Lim   = s.i_TOF.hist.Range;% [ns] Lim of the axis that
    shows the variable.
s.i_TOF.axes.Label.String = 'TOF [ns]'; %The label of the variable
```

Then, this signal can be used to define the plot metadata. The plot metadata can have different fields:

- *hist* Contains information to form the histogram, such as the signal to look at, the binsize, etcetera. The possible fields are:
  - **data_pointer** (mandatory), the pointer where to find the data to use in the histogram. Example: 'h.det2.TOF', or in the case of two signals: {'h.det2.TOF', 'h.det2.R'}.
  - **binsize** (mandatory), the size of the bins to construct the histogram (in the unit of the data pointer). Example: binsize = 3
  - **Range** (mandatory), the extreme values over which the histogram should be made. Example: Range =

    $$23, 45$$

    .
  - **hitselect** (optional), used to indicate which hit number in an event should be shown (only for hit signals). For example, if the second and first hit need to be shown in coordinate X and Y, hitselect = {1, 2}. By default, the histogram shows all hits.
  - **dim** (mandatory), the dimension of the histogram. One signal can contain one to more dimensions, for example the three-dimensional momentum vector would imply dim = 3.
  - **Integrated_value** (optional), used to denote what value the integrated value of the entire histogram should be. For example, a normalization to one would imply an Integrated_value = 1
  - **Maximum_value** (optional) The maximum of the histogram count can be rescaled. If Integrated_value is defined, the Maximum_value is ignored.
  - **Intensity_scaling** (optional), allows for logarithmic ('log') or linear scaling. Default: 'linear'.
  - **Type** (optional) by default, the histogram is made in a 'cartesian' form, optionally it can be made in a 'ternary' grid.
  - **bgr** (optional), specify a signal to be used in a background subtraction.
  - **saturation_limits** (optional), specify histogram count values (minimum and maximum) at which the histogram count saturates.

By default, the histogram does not saturate. For example saturation_limits=

$$0, 50$$

.

– **ifdo.Solid_angle_correction** (optional), Solid angle histogram correction, used in angular (polar) plots. by default, ifdo.Solid_angle_correction = false.

- *axes* Contains information about the axes. The fields in this struct are copied to the axes handle of the actual axes. For example, axes.YTick = linspace(0, 1, 101).
- figure Contains information about the figure. The fields in this struct are copied to the figure handle of the produced figure. For example, figure.Name = 'Time Of Flight plot'.
- *GraphObj* Contains information about the Graphical Object (line, patch, etc.). The fields in this struct are copied to the Graphical Object handle of the produced Graphical Object. For example, GraphObj.Color = 'b'.
- *cond* (optional) The *cond* field can define a condition to be applied to the histogram. See section 4.5 for more info on how conditions are defined.

The ion TOF signal can be imported in a one-dimensional plot in the following way:

```
% Load some default plot signal metadata:
exp_md              = metadata.defaults.exp.CIEL.plot_signals;
% Use one of the signal metadata fields (i_TOF) to create a new plot
    metadata field:
d2.TOF              =
    metadata.create.plot.signal_2_plot({exp_md.plot.signal.i_TOF});
```

Which takes care of all mandatory plot.hist fields. We can add additional information on our plot preferences:

```
d2.TOF.hist.Integrated_value  = 1;
d2.TOF.hist.hitselect       = 2;
d2.TOF.GraphObj.ax_nr       = 1;
d2.TOF.axes.Position     = plot.ax.Position('Full');
% Axes properties:
d2.TOF.axes(1).YTick     = linspace(0, 1, 101);
```

```
d2.TOF.axes(1).YLim      = [0 0.01];
d2.TOF.cond         = exp_md.cond.e_KE;
```

TODO:
Give possible fields of plot metadata Give possible fields of condition metadata Give possible fields of histogram metadata