2015 VLTI school

Cologne, 6–13 September

Practice session : AMBER data reduction

During this practice session, you will use the amdlib data-reduction package provided by the Jean-Marie Mariotti Center (JMMC) to reduce and calibrate AMBER observations. The final product of this data-reduction procedure can then be used in model-fitting software, such as LITPro that you will use during tomorrow session, or compared to physical models. amdlib software can be directly downloaded from the JMMC web site at the url : **http://www.mariotti.fr/data_processing_amber.htm**

1 First contact with amdlib and amber data

amdlib is a set of C routines that can be used in command line mode or with a yorick based interface. Yorick is an interpreted programming language designed for data analysis. It is comparable to IDL in term of capabilities but the syntax is slightly different. To launch yorick with the amdlib library, open a new terminal window and type "amdlib". Once you are under yorick, you can list amdlib common functions/commands by typing again :

```
amdlib
```

Almost all amdlib commands start with the prefix "amdlib". You can access to the help of a function by typing "help, function". For example, to obtain some help about the amdlibCreateLog function, just type :

```
help, amdlibCreateLog
```

amdlibCreateLog is a command creating a log file from files in a given directory (inputDir). As for all amdlib functions, if you don't specify the input directory (or input file for some other functions), the software will open a GUI to let you choose the proper directory or file(s). Try to type :

```
amdlibCreateLog
```

This will open amdlib file-and-directory-picker GUI. In this first part of the tutorial we will deal with some real AMBER data observed on a star. Thus, go to the data directory associated with the session. Once there, click on the "THISDDIR" button. The log file for this directory is now generated. Such log file help you identify the nature of all AMBER observations files. Now, reopen the file-and-directory-picker GUI by typing again amdlibCreateLog. All AMBER files in the directory are now shown in different colours depending on their type. Some additional information are also shown.

Question : What information are given in each of the 6 columns of the GUI?

Question : What are the 5 different types of AMBER raw files?

Before leaving the GUI by clicking on the "CANCEL" button, just look at the "Compute Log" button. This button will call the amdlibCreateLog function in the current directory without leaving the GUI. It will be useful when we will start to reduce data.

2 Looking closer at AMBER raw data

It is now time to look at what is inside a AMBER raw file. All AMBER raw and reduced files are in the common fits format that can contain both ASCII and/or BINARY tables and all start with the ASCII header. In the case of AMBER raw files they contains images of the detector in binary format. To display a raw file we will use the amdlibShowOiData function. But first we need to load both a bad-pixel map and a flat-field map that will be used to correct the raw data from detector-based biases. Just type

```
amdlibLoadBadPixelMap,inputBadPixelFile
```

Go to the cosmetic folder, and select the file "BPM_XXXXXX.fits". You should always choose the closest yet anterior BPM file. Do the same for the Flat Field map:

```
amdlibLoadFlatFieldMap,inputFlatFieldFile
```

and select "FFM_XXXXXX.fits". Now that both maps are load into memory (and will stay there as long as you don't quit yorick) you can display some AMBER raw files. First type :

```
amdlibShowRawData
```

In the interactive GUI, go to the data directory, and choose a star object file (in green). The program will then ask for a bias (i.e. DARK) file. Select the one taken just before the chosen OBJECT file. After that step, two windows open : the first one showing the first frame (i.e. image) of the selected file. You can change the image dynamics by modifying the low ("l") and high ("h") cut values in the Yorick console.

Question : Describe the image and spot the photometric and interferometric channels.

The second windows is showing some vertical and horizontal slices at the position given by the green lines on the first windows. You can modify the slices position by typing "m" and then clicking on the image window at the chosen position.

Question : Why is the vertical scale labelled as "wavelength" and not as "pixel number"?

The file you have selected contains several frames you can display. In the Yorick console, type the number of the frame you want to display. You can also display an animation of all the frames by typing "a".

Question : Why is the fringe pattern different for each frame?

Question : What is wavelength of the bright horizontal line visible in all channels?

Question : Can you give the name of this spectral feature?

To quit the interactive amdlibShowRawData tools, just type"q". Now let's look at an OBJECT file for the next star in the list.

Question : What is the main difference compared to the previous star data?

Question : Why was data on this star recorded just after the previous observation?

Finally, you can look at few of the 10 3P2V files (don't choose the first one). These files are recorded before the observations using an internal light source. Select the first 3P2V file as a bias file.

Question : Can you guess why are the 3P2V files recorded for?

3 From raw to reduced data

It is now time to reduce the data. The process of AMBER data reduction is done in three phases. First you compute the pixel to visibility matrix (P2VM) from the 3P2V files. Then, you compute the per-frame reduced data for the raw OBJECT files using the P2VM. Finally, you average the per frame reduced data taking into account each frame quality. Each step can be performed for a single file or for a full directory. The following table summarise these steps and the functions used to perform them.

Step	Per file function	Per directory function
Compute P2VM matrix	amdlibComputeP2vm	amdlibComputeAllP2vm
Compute per frame reduced data	amdlibComputeOiData	amdlibComputeAllOiData
Average frames data	amdlibPerformFrameSelection	amdlibPerformAllFrameSelection

You can check the parameters and options of these function using the "help" command. Let's start with our δ Cen data set. In our example, we will use the per-directory functions, with the standard parameters. First we'll compute all the P2VM in the δ Cen data directory :

```
amdlibComputeAllP2vm
```

Normally the path of the GUI should already be set to δ Cen data directory. Just click on the "THISDIR" button. As there is only one set of 3P2V files (10 in total) in the directory, only one P2VM will be computed. Now let's reduced the raw data using the P2VM. Just type

```
amdlibComputeAllOiData
```

You can see a new sub-directory named "XXX_P2VM". As evidenced by its name, it contains the newly computed P2VM. But right now, all you need to do is stay in the main data directory and click again on the "THISDIR" button. All the per-frame reduced data will be put in new sub-directory "XXX_OIDATA". Finally compute the average data on all frames within a file using

```
amdlibPerformAllFrameSelection
```

This time go to the "XXX_OIDATA" and click on "THISDIR". The average data will be put in a "XXX_OIDATA_AVG" sub-directory. Now that the data are fully reduced (but not yet calibrated) we can look at them using :

```
amdlibShowOiData
```

First go to the averaged data directory. You should see 10 files inside it. Click on the "Compute Log" button. 5 files will be coloured in blue, they concern the calibrator and 5 in green (science star). Click on a science star file. A new window summarising the data will open. It contains plots of the flux, square visibility, differential phases, and closure phase as a function of the wavelength. It also show the telescope configuration, the uv-plane coverage and some other important information : seeing, spectral mode, name of the star...

Question : Why is there 3 plots for the flux, 3 for the visibility, 3 for the differential phases and only one for the closure phase?

Look closer at the visibility and phases variations as a function of wavelength.

Question : What does the variation of visibility signal as a function of wavelength tells you? What about the phase signal?

Now, use the amdlibShowOiData on a calibrator file.

Question : Explain the main difference with the science target.

4 Calibrating your data

4.1 Getting calibrators diameters

Now that the data are reduced you need to calibrate them. This final step consist in evaluating the instrument + atmosphere response for a non-resolved object and correct the measured visibility and phases from this so called transfer function. That is why, for all interferometric observations that require absolute measurements (i.e. absolute visibility and closure phases), some unresolved stars, called calibrators, must be observed, preferentially twice, before and after the science target.

Unfortunately, it is often impossible to find a fully unresolved star bright and close enough to the science target. Nevertheless, a partially resolved target can be used as calibrator if its diameter has previously been measured. To get the diameter of the calibrators from various available database, just use the function :

`amdlibSearchAllStarDiameters`

Select the averaged-data directory, and click on "THISDIR". The diameter and uncertainty of the calibrator(s) will be written on the "amdlibCalibDataBase.txt" usually located in your home directory. If the diameter cannot be found in any known database, you will have to add it manually in the "amdlibCalibDataBase.txt" file. For example, the calibrator HD 110458 has a diameter of 1.65±0.02 mas that needs to be added in the database.

Question : Considering a wavelength of 2.2μm, what baseline length is needed to fully resolved this star?

Question : Is this star a good calibrator for our observation with a longest baseline of 130m?

4.2 Calibrating the data

Now that we know the calibrator diameter, we can compute the calibrated visibilities and phases. This can be done in two steps: first estimating the *transfer function*, i.e. the visibility function calculated on calibrators as if they had a diameter of 0 mas, and then interpolate this transfer function at the moment of the science observation.

```
amdlibComputeAllTransferFunction
```

This will create a new sub-directory "XXX_OIDATA_AVG_TRA" which contains all the transfer function files. One can visualize it with the function

```
amdlibShowTransferFunctionVsTime (or amdlibShowTransFuncTime)
```

for plotting it as a function of time.
Question : What are each of the four plots about?

Question : Does the transfer function bring information on how the night went well?

One can also plot the transfer function as a function of wavelength:

```
amdlibShowTransferFunctionVsWlen (or amdlibShowTransFuncWlen)
```

Question : Why is the transfer function variable as a function of wavelength?

One can play with the interpolation parameters `interpolStyle` and `interpolParam` (see details in the help). One happy with the results, the calibration itself can be performed with the function

```
amdlibCalibrateOiData
```

which accepts the same parameters `interpolStyle` and `interpolParam` as the previous vizualization function.
Finally, you can look at the calibrated data using the amdlibShowOiData function. The calibrated data are stored on a "XXX_OIDATA_AVG_PRO" sub-directory.

4.3 Calibrating the data (alternate method)

In amdlib, an alternate method has been implemented, which provides similar results: the function

```
amdlibCalibrateAllOiData,checkPlot=1,writeOutputFiles=1
```

computes the calibrated visibilities and phases. Select the averaged-data directory and click again of "THISDIR". Here, we use the function with two keywords, *writeOutputFile=1*, which means that the computed calibrated data will be saved to the disk, and *checkPlot=1*, which show some plots to verify the consistency and quality of the calibrated data.

Finally, the calibrated data are stored on a different directory "XXX_OIDATA_AVG_CAL".

5 Bonus : advanced processing for trickier datasets

If your data are "normally" good, you will never need to do anything more than these steps. Unfortunately, life is hard, and some data are hard to deal with. The directory "lowres" contains some a trickier data-set that need special "treatment". These data were taken in low-resolution mode so that they cover the whole J, H, and K bands.

First we can have a look at the raw data themselves (for a OBJECT file) using the *amdlibShowRawData* function.

Question : What are the two dark horizontal lines seen in the photometric and interferometric channels?

Question : Why aren't they at the same position in every channel?

These spectral shifts of the channels can affect the reduced data if they are not well characterised.

5.1 Wavelength calibration problems

Let's start by computing the P2VM without any option. Type :

```
amdlibComputeP2vm
```

Select the 10 3P3V files and click on "OK". It will use some default shifts that are printed on-screen during the computation. The amdlibComputeP2vm function will also calibrate the wavelengths table using the gaps between the J, H, and K bands. These two wavelength-calibrations may be biased for some data-sets.

You can check manually the spectral shifts using the *amdlibComputeP2vm* function with the keyword *specCalShifts* set to *"MAN"*

```
amdlibComputeP2vm,specCalShifts=''MAN''
```

First select the 10 3P2V files, then an OBJECT file, and finally a bias (DARK) file. Four windows open : three with the superposition of the interferometric channel and the each photometric channel, and the window 0 with control to modify the spectral shifts between the channels. Play with these shifts to improve the curves superposition for the three photometric channels.

Question : Are the offsets obtained with this method different from the default ones?

5.2 Data quality and frame selection

Now, compute the reduced data using the *amdlibComputeAllOiData* function. You can then use the *amdlibShowOiData* function to look at these non-averaged OIDATA files. Unlike for the averaged files, the function opens two windows. In the first one you can select a wavelength by left clicking on the flux plot. The visibility, flux, flux ratio and closure phase histograms from all frames within the file for this wavelength will be shown in the second window.

Question : What is roughly the mean value of the visibility for the three baselines in the J, H, and K band?

Question : What about the dispersion of the measurements?

Question : And what about the closure phase?

Question : Conclude on the respective quality of the J, H, and K band measurements

We can recompute the reduced data using the *band* keyword to select a band, for instance the K band

```
amdlibComputeAllOiData,band="K"
```

We can also split the bands into three different files using the *splitBands* keyword

```
amdlibComputeAllOiData,splitBands=1
```

Let's use the K band data only for the next steps. The *amdlibShowOiData* function also contains a plot of the pistons on the three baselines as a function of time. These data were taken using the VLTI fringe tracker FINITO which is supposed to lock the fringes with a precision of less than one microns.

Question : Did FINITO works well for this data-set?

The frames recorded with a high piston will affect the value of the average visibility and closure phases. The *amdlibPerformAllFrameSelection* function can be used to remove frames with too high piston, as well as frame with low SNR, or with too high flux ratio between the three beams.

To perform a frame selection with criteria we must use three keywords :

- *selCriterion*, the name of the criterion : "piston" for selection on piston value, "snr" for the fringe SNR, or "flux" for the flux SNR.
- *selType* which can be equal to "threshold" or "percent"
- *selValue*, the value of the frame selection criterion. write

For example to select the 20% of best frames in term of SNR you should write :

```
amdlibPerformAllFrameSelection,selCriterion="snr",selType="percent",
                                selValue=20
```

To keep frames with a piston of less than 15 microns use

```
amdlibPerformAllFrameSelection,selCriterion="piston",selType="threshold",
                                selValue=15
```

You can also use multiple criterion. For instance,

```
amdlibPerformAllFrameSelection,selCriterion=["flux","piston","snr"],
  selType=["threshold","threshold","percent"], selValue=[10,20,50]
```

will compute the average with the following consecutive selection criteria : on the flux SNR larger than 10, on piston with less than $\pm 20\mu$m, on fringe SNR with the 50% best frames.

For our example, we will keep the 80% frames with the best fringe SNR after removing the frames with a piston of more than $\pm 15\mu$m.

5.3 Time dependent transfer functions

Let's try to calibrate this tricky data-set. First, we need to look for diameters using the *amdlibSearchAllStarDiameters* function. Then we will use the *amdlibCalibrateOiData* (see previous sections).

Question : Compare the visibility transfer function obtained on the calibrator before and after the science target

Question : What can be the cause of such fluctuations?

There are two ways to deal with these fluctuation : the conservative and the optimistic ones.

If you are conservative, you can consider that you don't know the evolution of the transfer function between your two measurements. The estimated transfer function is, if you consider that the two measurements are independent, the average of these measurement. Its uncertainty is given by the standard deviation (σ) from the average. This is the result given by the amdlibCalibrateOiData if you specify the options `interpolStyle="polynom"` and `interpolParam=0`.

If you are optimistic, you can consider that you can "trace" the transfer function evolution between your measurements. For example, you can consider that it evolved linearly. In that case, the transfer function at the time of the science star observation is given by the linear interpolation between the two nearest calibrators. Finally, the uncertainty on the transfer function is equal to the mean uncertainty on all the calibrator measurements. To use such optimistic transfer function with linear interpolation in amdlib, just type :

```
amdlibCalibrateOiData ,interpolStyle="lines";
```